**Software Requirement Specification (SRS)**

**Project:** Vehicle Detection & Number Plate Recognition Using Image Processing & Deep Learning

**Table Of Content**

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, Abbreviations

1.4 References

1.5 Overview

1. Overall Description

2.1 Product Perspective

2.2 Product Functions

2.3 User Classes and Characteristics

2.4 Operating Environment

2.5 Design and Implementation Constraints

2.6 Assumptions and Dependencies

1. Specific Requirements

3.1 External Interface Requirements

3.2 Functional Requirements

3.3 Non-Functional Requirements

3.4 Logical Database Requirements

3.5 Design Constraints

3.6 Security and Privacy Requirements

1. Use Cases

4.1 UC-1: Upload Video and Run Detection

4.2 UC-2: Live Feed Detection

4.3 UC-3: Admin — Change Model & Parameters

4.4 UC-4: Download Results & Reports

1. System Architecture and Data

5.1 High-Level Components

5.2 Data Flows

5.3 Database Schema

5.4 Output Formats

1. Test and Acceptance Criteria

6.1 Acceptance Tests

6.2 Unit and Integration Tests

6.3 Performance Tests

1. Requirements Traceability Matrix
2. Risks, Mitigations, and Future Work
3. Appendices

9.1 Implementation Notes

1. UML Diagrams

10.1 Use Case Diagram

10.2 Class Diagram

10.3 Sequence Diagram — Upload & Detect

10.4 Sequence Diagram — Live Feed

1. **Introduction:**
   1. **Purpose:**

This document specifies the software requirements for a system that detects vehicles in video or image streams, localizes license plates, reads plate text with OCR, and exposes results via a web UI and API. It is written according to IEEE-830 conventions to be used by developers, test engineers, and project evaluators.

**1.2. Scope:**

The system provides: video upload and live-feed ingestion, frame extraction, vehicle detection (baseline Haar Cascade + optional YOLO family), plate localization, OCR processing (Tesseract or similar), a web UI to run and monitor jobs, exportable results (CSV/JSON), and model evaluation reports. Out of scope: integration with law-enforcement databases, multi-camera tracking, production-level scaling.

**1.3. Definitions, Acronyms, Abbreviations**

* OCR — Optical Character Recognition
* ROI — Region of Interest
* FPS — Frames per Second
* UI — User Interface
* API — Application Programming Interface
* COCO — Common Objects in Context dataset
* mAP — mean Average Precision

**1.4. References:**

OpenCV, PyTorch/TensorFlow, Tesseract (implementation libraries)

**1.5. Overview**

Sections 2–7 describe the system in increasing detail: overall description, functional & external interfaces, nonfunctional requirements, use cases, system architecture & data, test & acceptance criteria, UML diagrams and appendices.

1. **Overall Description:**
   1. **Product Perspective**

Standalone web application with backend AI inference pipeline. The backend runs detection/OCR and provides REST APIs. The UI displays annotated frames, allows configuration, and downloads.

**2.2. Product Functions**

* Upload videos / specify live feed URL
* Configure model & parameters (detection threshold, FPS)
* Run detection pipeline (frame extraction → detection → plate crop → OCR)
* Stream live annotated frames to UI
* Export results and evaluation reports

**2.3. User Classes and characteristics**

* User / Operator: uploads video or starts live feed, views results. Technical but not necessarily ML expert.
* Admin: configures models, retention policy, users.
* Developer / Tester uses backend APIs and test harnesses.

**2.4. Operating Environment**

* Backend: Ubuntu 20.04+ (production) / Windows 10 (dev)
* Python 3.9+, OpenCV, PyTorch/TensorFlow, Tesseract OCR
* Browser: Chrome/Firefox modern versions
* Hardware: CPU with at least 4 logical cores; optional GPU (NVIDIA) recommended for YOLO experiments

**2.5. Design & implementation constraints**

* File upload size limit (configurable; default 500MB)
* Privacy rule: plate data stored only temporarily (configurable retention)
* Must be modular: detection, OCR, UI decoupled

**2.6. Assumptions & dependencies**

* Pretrained model weights available (downloadable)
* COCO dataset used for detection baseline and evaluation
* Tesseract supports license plate character sets used in demos

1. **Specific Requirements:**
   1. **External Interfaces Requirements**
      1. **Web UI**

* POST ‘/upload’ - accepts video file form-data (max-size configurable)
* POST ‘/live’ - accepts JSON to start a live job
* GET ‘/session/{id}/stream’ - server-sent events or WebSocket streaming frames with overlay data
* GET ‘/session/{id}/results’ - returns JSON summary / CSV
  + 1. **REST API (backend)**
* POST ‘/api/run’ - start job; input: file or feed URL; output: { session\_id }
* GET ‘/api/status/{session\_id}’ - job status & progress
* GET ‘/api/results/{session\_id}’ - JSON or download link for CSV/JSON report
* POST ‘/api/config’ - admin to update model/config
  + 1. **Model and libraries interfaces**
* PyTorch/TensorFlow model loader, OpenCV for frames, Tesseract for OCR
  1. **Functional Requirements**

**FR-1 - Video Upload & Validation**

* Description: Accept video uploads (MP4, AVI). Validate format, size.
* Priority: High
* Fit criterion: Upload succeeds and returns session\_id in ≤ 10s for a 100MB file on typical network.

**FR-2 - Live Feed Ingestion**

* Description: Accept RTSP/HTTP/webcam streams and initiate detection.
* Priority: High
* Fit: Connect to RTSP and begin streaming frames within 5s for valid URL.

**FR-3 - Frame Extraction**

* Description: Extract frames at configurable FPS (default 5 FPS for processing).
* Priority: High
* Fit: For a 60s video at 5 FPS, 300 frames extracted.

**FR-4 - Vehicle Detection**

* Description: Detect vehicles in frames; return bounding boxes + model + confidence.
* Priority: High
* Fit: Provide [{x, y, w, h, confidence, class}] per detected object.

**FR-5 - License Plate Localization**

* Description: Given vehicle bbox, detect plate bbox inside ROI.
* Priority: High
* Fit: Provide plate bbox with confidence.

**FR-6 - OCR (Plate Recognition)**

* Description: Run OCR on plate crops; return text & confidence; provide preprocessing options (denoise, binarize).
* Priority: High
* Fit: Output text string and numeric confidence.

**FR-7 - Annotated Streaming**

* Description: Stream frames to UI with overlay (vehicle boxes, plate boxes, OCR text).
* Priority: High

**FR-8 - Model Selection & Parameters**

* Description: Admin/UI can choose model (Haar / YOLO / custom), set thresholds, set FPS, and preprocessing toggles.
* Priority: Medium

**FR-9 - Results Export**

* Description: Export per-session JSON/CSV containing timestamped detection rows.
* Priority: High
* Fit: CSV with columns: session\_id, timestamp, frame\_no, vehicle\_id, vehicle\_bbox, vehicle\_conf, plate\_bbox, plate\_conf, ocr\_text, ocr\_conf, model

**FR-10 - Model Evaluation Report**

* Description: Provide precision/recall/mAP/FPS for models on chosen dataset subset.
* Priority: Medium
* Fit: Deliver a downloadable PDF/JSON report with metrics and sample images.

**FR-11 - Logging & Retention**

* Description: Log per-session results; configurable retention (default 7 days).
* Priority: Medium

**FR-12 - Error Handling & Retry**

* Description: Graceful error messaging and retries for transient stream failures.
* Priority: High
  1. **Non-Functional Requirements**

**NFR-1 - Performance and Throughput**

* Real-time mode: the system should process at least 8 FPS on a target machine with GPU acceleration (e.g., NVIDIA GTX/RTX class).
* CPU baseline: system should achieve at least 2 FPS on a modern 4-core CPU for small models (e.g., HAAR / tiny-yolo).
* Batch mode: average processing latency per frame (extraction → detection → plate detection → OCR) should target ≤ 0.5 seconds on a modern CPU for lightweight models.

**NFR-2 - Accuracy and Reliability**

* Vehicle detection baseline: the initial Haar cascade model should reach at least 75% detection accuracy on a held-out test subset; advanced models (YOLO) should exceed this baseline.
* OCR accuracy: recognize at least 70% of plates clearly visible in normal lighting conditions; this depends on preprocessing and plate style.
* System reliability: the system must recover from transient stream failures (network interruptions) with automated retries (configurable) and must not crash on malformed frames.

**NFR-3 - Scalability and Portability**

* The application design must allow horizontal scaling of inference components (moving to GPU-backed workers) without significant code rework.
* The solution must be runnable on both Linux (Ubuntu 20.04+) and Windows 10+ for demonstration purposes.

**NFR-4 - Usability and Accessibility**

* Typical user (operator) should be able to start a detection run (upload + run) within 2 minutes of landing on the UI.
* UI should provide clear progress indicators and error messages and should be usable on modern desktop browsers (Chrome, Firefox).

**NFR-5 - Maintainability and Modularity**

* Codebase must be modular: separate components for VideoProcessor, InferenceEngine, Storage, and WebUI, each with clear interfaces and unit tests.
* Adding a new detection model should require minimal integration code (load new weights and register model name).

**NFR-6 - Security and Privacy**

* Admin endpoints must require authentication; role-based access control must be in place for configuration and session management.
* Default data retention for uploaded videos, frames, and OCR results is 7 days (configurable). Automated deletion processes should be implemented to purge data after retention expires.
* The option to anonymize or blur plate numbers in saved images to comply with privacy concerns during demos.

**NFR-7 - Operational and Logging**

* Comprehensive logging for sessions, errors, model loads, and performance metrics (FPS, per-frame latency) must be recorded and available for diagnostics.
* Exportable logs and session summaries (CSV/JSON) for debugging and evaluation purposes.

**NFR-8 - Legal and Compliance**

* The system shall include documentation and warnings about legal restrictions on license plate recognition and data retention; the demo deployment must not share plate data externally unless explicitly authorized.
  1. **Logical Database Requirements**

A relational DB (SQLite/Postgres) with these tables (minimum): users: user\_id (pk), username, role, hashed\_password sessions: session\_id (pk), user\_id, type {upload|live}, source, start\_time, end\_time, model, status frames: frame\_id (pk), session\_id, frame\_no, timestamp, path detections: detection\_id (pk), frame\_id, vehicle\_bbox, vehicle\_conf, plate\_bbox, plate\_conf, ocr\_text, ocr\_conf, model

* 1. **Design Constraints**
* Modular pipeline: VideoProcessor, InferenceEngine, Storage, UI components decoupled.
* Must be runnable locally for demo (no cloud required).
* Ability to swap models by name or weight file.
  1. **Security and Privacy Requirements**
* Authentication required for admin endpoints.
* Retention policy default 7 days; configurable.
* Optional anonymization: blur plate text in saved images.

1. **Use Cases:**
   1. **Upload Video and Run Detection**

**ID**: UC-1

**Actors**: User (Operator)

**Description**: Upload a video file and run detection pipeline; view annotated frames & download results.

**Preconditions**: User is on Upload page; file <= allowed size and supported format.

**Main flow**:

* User selects file and clicks Upload.
* UI sends file via POST /api/run with params {model, fps, thresholds}.
* Server returns { session\_id }.
* Server extracts frames from chosen FPS.
* For each frame: VideoProcessor → InferenceEngine (detect vehicles → detect plates → OCR).
* Server saves detection rows and streams annotated frames to UI.
* UI displays live progress and overlays boxes/text.
* After the job is completed, the user downloads the results CSV/JSON.

**Alternate flows**:

A1: User cancels upload at step 1 → Server deletes partial upload and returns status canceled.

A2: User chooses evaluation mode instead of run → Server runs evaluation and returns metrics.

**Exception flows**:

E1: Unsupported file format → UI shows error and lists supported formats; job not created.

E2: Model loading failure (e.g., corrupted model weights) → Server returns error and logs.

* 1. **Live Feed Detection**

**ID**: UC-2

**Actors**: User

**Description**: Connect to a live camera (RTSP) and run real-time detection.

**Preconditions**: Camera reachable and credentials provided (if needed).

**Main** **flow**:

* User provides RTSP URL and params; clicks Start.
* Server connects, streams frame, performs detection/OCR, and streams overlay to UI.
* User clicks Stop to end session.

**Exceptions**:

Camera unreachable → show connection error and retry option.

Intermittent disconnect → server attempts N retries; if fail, marks session failed.

* 1. **Admin – Change Model & Parameters**

**ID**: UC-3

**Actors**: Admin

**Description**: Admin logs in, changes detection model, sets thresholds and retention.

**Preconditions**: Admin authenticated.

**Main** **flow**: Admin posts to /api/config with new model name and parameters; server validates and saves config for subsequent runs.

**Exceptions**: Unauthorized access → 403 response.

* 1. **Download Results & Reports**

**ID**: UC-4

**Actors**: User

**Description**: Download session results as JSON/CSV and evaluation report (if available).

**Main** **flow**: User opens session page and clicks Download; server returns file stream.

1. **System Architecture and Data:**
   1. **High-level components**

Web UI, API Server, Video Processor, Inference Engine, Storage, Model Evaluator

* 1. **Data Flows**

Video → Frame extraction → Preprocess → Vehicle detector → Plate detector → OCR → Save detections → Stream overlays → Export reports

* 1. **Database Schema (example)**

{

"detection\_id": 123,

"session\_id": "sess-001",

"frame\_no": 45,

"timestamp": "2025-08-10T12:02:10Z",

"vehicle\_bbox": [x, y, w, h],

"vehicle\_conf": 0.92,

"plate\_bbox": [x, y, w, h],

"plate\_conf": 0.87,

"ocr\_text": "ABC1234",

"ocr\_conf": 0.78,

"model": "yolov5s"

}

* 1. **Output Formats**

session\_id, timestamp, frame\_no, vehicle\_bbox, vehicle\_conf, plate\_bbox, plate\_conf, ocr\_text, ocr\_conf, model

1. **Test and Acceptance Criteria:**
   1. **Acceptance tests**

* AT-1: Upload 60s sample video → system completes processing; returns CSV; success if ≥ 80% of visible vehicles detected & ≥ 70% of visible plates OCR readable (values adjustable).
* AT-2: Start live feed for 2 minutes → system remains stable, and stream displays overlays; no fatal crashes.
* AT-3: Admin changes model from Haar to YOLO → subsequent runs use chosen model.
  1. **Unit / Integration tests**
* Unit tests for frame extractor, detector wrapper, OCR wrapper.
* Integration tests: small sample videos through full pipeline.
  1. **Performance tests**
* Measure FPS for each model under CPU/GPU; log throughput and latency.

1. **Requirements Traceability Matrix:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Req ID** | **Short Req** | **Mapped Use Case(s)** | **Test Case(s)** |
| FR-1 | Upload video | UC-1 | AT-1 |
| FR-2 | Live feed | UC-2 | AT-2 |
| FR-4 | Vehicle detection | UC-1, UC-2 | AT-1 |
| FR-6 | OCR | UC-1, UC-2 | AT-1 |
| FR-9 | Export results | UC-4 | AT-1 |

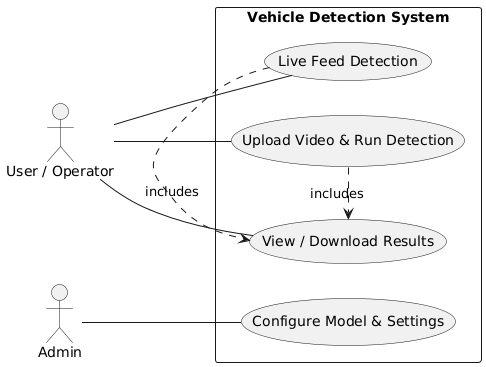
1. **Riska, Mitigation, and Future Work:**

* Risk: Poor OCR for non-standard plates or poor lighting.  
   Mitigation: Add preprocessing steps (contrast, morphological operations), try multiple OCR engines, consider fine-tuning plate-text dataset.
* Risk: Low throughput on CPU.  
   Mitigation: Provide “lite” model variants (tiny-YOLO), allow lowering FPS, support GPU deployment.
* Future work: Multi-camera tracking, vehicle re-identification, GDPR-compliant privacy modes, database linkage for authorized uses.

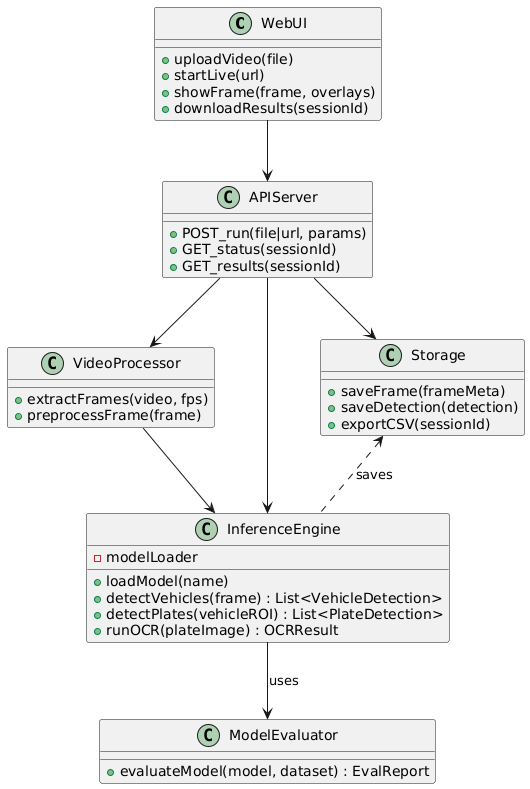
1. **Appendices:**
   1. **Implementation notes**

* Languages: Python 3.9+.
* Key libs: OpenCV, PyTorch/TensorFlow, Tesseract OCR, Flask/Django, Celery (optional) for background jobs.

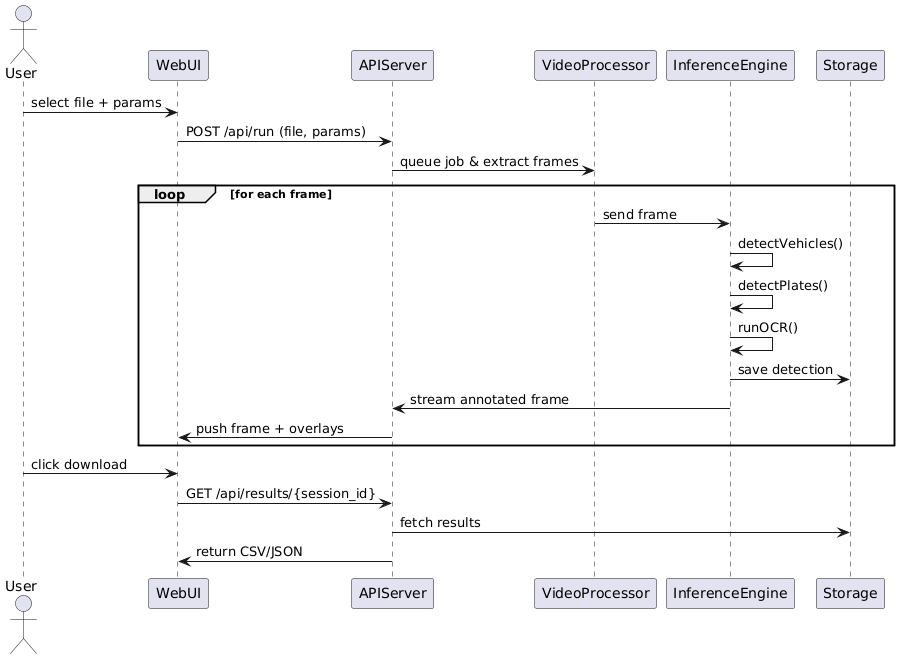
1. **UML Diagrams:**
   1. **Use Case Diagram**



* 1. **Class Diagram**



* 1. **Sequence Diagram – Upload & Detect**



* 1. **Sequence Diagram – Live Feed**

