**Software Architecture Document**

1. **Introduction**
   1. **Project Overview**

TraS (Traffic Sign Smart Detector) is a web-based system designed to detect and classify traffic signs from video recordings of urban routes. The system utilizes a service-oriented architecture (SOA) to process uploaded videos, extract frames, detect traffic signs using AI, and enrich the detection results with semantic knowledge. The goal of TraS is to assist urban planners, autonomous vehicle developers, and traffic management authorities in analyzing road conditions and improving transportation safety.

* 1. **Objectives**

The primary objectives of TraS are:

1. Automated Traffic Sign Detection: Accurately identify traffic signs from video recordings using AI-powered object detection.
2. Scalability and Modularity: Implement a microservice-based architecture to allow independent scaling of processing units.
3. Semantic Knowledge Integration: Use an OWL-based ontology to enrich detected traffic signs with additional metadata and regulations.
4. Efficient Video Processing: Split videos into frames and process them asynchronously to optimize detection speed.
5. User Interaction and Reporting: Provide an interface for users to upload videos, view detections, and report incorrect results to improve the AI model.
6. Cloud-Based Storage and Processing: Utilize cloud services for video storage, data management, and computational tasks.
   1. **Scope**

TraS is a microservice-based web application that processes urban traffic videos to detect and classify traffic signs. The system follows a service-oriented architecture (SOA) and consists of multiple services working together.

1. In Scope (Features and functionalities)

a. Video Upload and Processing: Users upload videos that are processed in the cloud. b. Frame Extraction: The system extracts frames from uploaded videos for analysis.

c. Traffic Sign Detection: A YOLO-based AI model detects signs such as speed limits, stop signs, and traffic lights.

d. Semantic Enrichment: Detected traffic signs are linked to an OWL ontology for contextual information.

e. Data Storage and Management: Videos, frames, and detection results are stored in cloud-based storage and databases.

f. User Interaction: Users can view detected signs, navigate through frames, and report incorrect detections.

1. Out of Scope(What this system does not cover)

a. Real-time Traffic Sign Detection: The system processes pre-recorded videos, not live camera feeds.

b. Vehicle Detection and Tracking: The system focuses on traffic signs, not vehicle behavior analysis.

c. Road Condition Analysis: The system does not detect potholes, lane markings, or road quality.

* 1. **High-Level Architecture Summary**

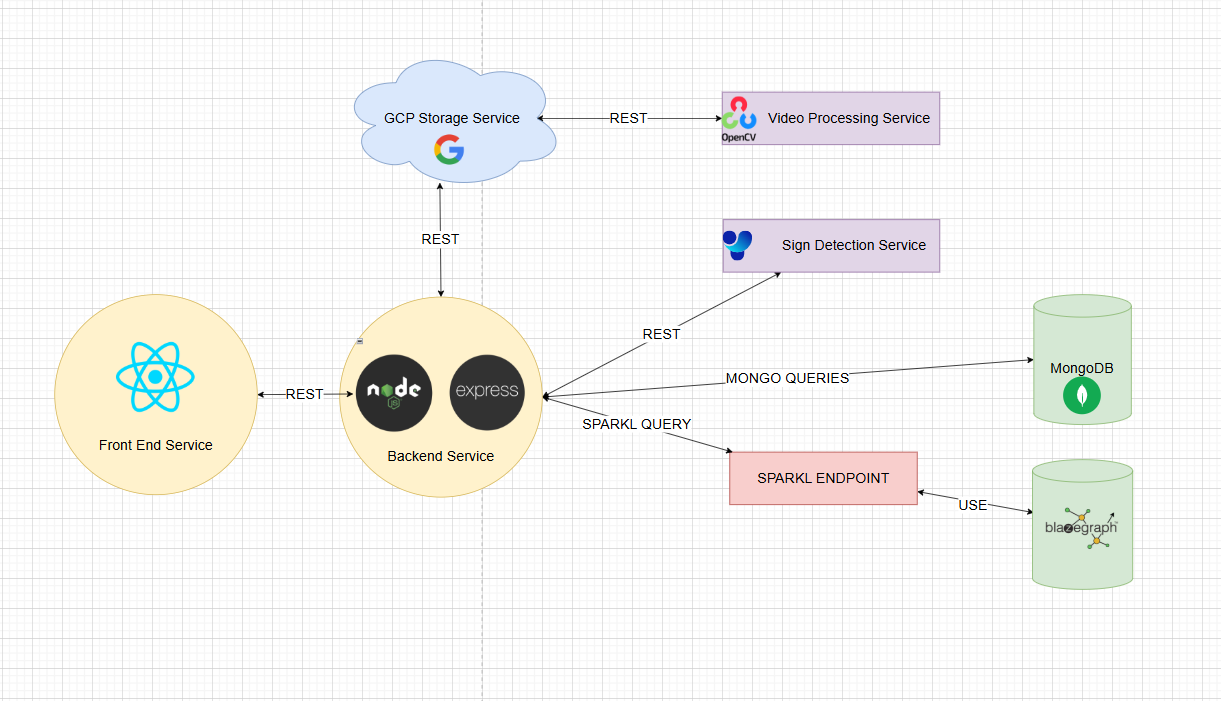
TraS follows a service-oriented architecture (SOA) to ensure modularity, scalability, and maintainability. The system is composed of independent services that interact through well-defined APIs.

* 1. Frontend Service – A web interface that allows users to upload videos, navigate through frames, and view detected traffic signs.
  2. Backend Service – Acts as the central controller, handling user requests, managing video processing tasks, and integrating with other services.
  3. Video Processing Service – Extracts frames from uploaded videos and stores them in cloud storage.
  4. YOLO Detection Service – Uses a trained YOLO model to detect traffic signs in frames and returns detection results.
  5. MongoDB Service – Stores metadata about uploaded videos, extracted frames, and detected traffic signs.
  6. OWL Knowledge Service– Provides semantic enrichment for detected traffic signs using an ontology-based SPARQL endpoint.
  7. Cloud Storage Service (GCP Storage) – Stores raw videos and extracted frames for processing. The system is designed to be asynchronous and event-driven, with microservices communicating via REST APIs and database queries to ensure efficient processing and scalability.

1. **System Architecture**
   1. **Overview**

The TraS system follows a service-oriented architecture (SOA) composed of multiple independent services that interact through well-defined APIs. The architecture is designed for scalability, modularity, and efficient traffic sign detection from video recordings. This section describes the high-level system design, including:

1. The core services that make up the system.
2. The interaction between these services.
3. The data flow from video upload to traffic sign detection.
4. The technologies used for different components. The following diagram provides an overview of the TraS architecture.
   1. **System Architecture Diagram**

****

The diagram above illustrates the high-level architecture of TraS, which follows a service-oriented approach. Each component is an independent service that interacts with others through well-defined APIs. The key interactions are:

a. The Frontend Service communicates with the Backend Service to handle user requests.

b. The Backend Service uploads videos to GCP Storage and delegates processing tasks.

c. The Video Processing Service extracts frames and stores them in GCP Storage

d. The YOLO Detection Service detects traffic signs in frames and returns results.

e. The MongoDB Service stores video metadata, frames, and detection results.

f. The OWL Knowledge Service provides additional semantic information about detected signs. This architecture ensures scalability, modularity, and efficient video processing.

* 1. **System Components**

The TraS system consists of multiple independent services that work together to process videos, detect traffic signs, and provide enriched data. Below is a description of each component.

a) Frontend Service

* A web-based interface where users can upload videos, navigate frames, and view detected traffic signs.
* Sends requests to the backend for processing and data retrieval.
* Receives processed results, including detected signs and semantic information.

b) Backend Service

* Acts as the central coordinator of the system.
* Manages video uploads, task delegation, and data retrieval.
* Interacts with other microservices via REST API and database queries.
* Enriches detection results with semantic data from the OWL Knowledge Service.

c) Video Processing Service

* Extracts frames from uploaded videos for analysis.
* Stores extracted frames in cloud storage (GCP Storage).
* Returns frame metadata (IDs and count) to the backend.

d) YOLO Detection Service

* Runs the trained YOLO model to detect traffic signs in frames.
* Receives frame IDs from the backend and processes them.
* Sends detected signs (bounding boxes, sign type, confidence) back to the backend.

e) MongoDB Service

* Stores metadata for uploaded videos and extracted frames.
* Keeps a record of detected traffic signs and their properties.
* Enables efficient retrieval of stored information for frontend display.

f) OWL Knowledge Service

* Provides semantic information about traffic signs using an OWL ontology.
* Supports SPARQL queries to enrich detection results with legal regulations and descriptions.

g) GCP Storage Service

* Stores raw videos and extracted frames for processing.
* Ensures scalability and high availability of stored media files.
  1. **Data Flow and Interactions**

The TraS system follows a structured data flow where each service interacts based on predefined roles. The process is divided into key steps to ensure efficient traffic sign detection and semantic enrichment.

a) Video upload and storage

* The frontend sends a request to the backend to upload a video.
* The backend stores the video in GCP Storage and records its metadata in MongoDB.
* The backend notifies the video processing service about the new video.

b) Frame extraction

* The video processing service extracts frames from the uploaded video.
* Extracted frames are stored in GCP Storage.
* The service returns a list of frame IDs and frame count to the backend.
* The backend stores this metadata in MongoDB and sends it to the frontend.

c) Frame selection and traffic sign detection

* The frontend allows users to navigate through frames and select one for detection.
* When a frame is selected, the frontend requests the backend to run detection.
* The backend retrieves the frame from GCP Storage and sends it to the YOLO detection service.
* The YOLO detection service analyzes the frame and returns detected signs with bounding boxes and confidence scores.
* The backend stores detection results in MongoDB.

d) Semantic enrichment

* The backend queries the OWL Knowledge Service using SPARQL to get additional information about detected signs.
* The OWL service responds with legal meanings and regulations associated with the detected traffic signs.
* The backend integrates this semantic information with the detection results and sends the final response to the frontend.

e) User interaction and feedback

* The frontend displays detected signs and enriched information.
* Users can review, confirm, or report incorrect detections.
* Reported detections are logged in MongoDB for future model improvements.
  1. **Technology Stack**

The TraS system is built using a combination of modern technologies to ensure scalability, efficiency, and modularity. Each component is developed using technologies best suited for its specific function.

a) Frontend

* Developed using React .
* Communicates with the backend using REST APIs.

b) Backend

* Implemented using Node.js with Express for handling API requests.
* Manages communication between frontend and microservices.
* Handles video uploads, processing requests, and semantic enrichment.

c) Video Processing Service

* Developed in Python using OpenCV for extracting frames from videos.
* Stores extracted frames in GCP Storage.

d) YOLO Detection Service

* Uses YOLOv8, trained on a custom dataset for traffic sign detection(traffic lights, speed signs and stop signs).
* Implemented in Python using the Ultralytics YOLO package.
* Processes frames received from the backend and returns detected traffic signs.

e) MongoDB Service

* Stores video metadata, extracted frames, and detection results.
* Ensures fast retrieval of stored data for frontend visualization.

f) OWL Knowledge Service

* Uses an RDF triplestore such as Blazegraph.
* Provides semantic enrichment using SPARQL queries.

g) Cloud Services

* GCP Storage for storing raw videos and extracted frames.
* GCP Compute Engine or Cloud Run for deploying backend and microservices.
  1. **Deployment Architecture**

The TraS system is deployed in a cloud environment to ensure scalability, fault tolerance, and efficient processing. The deployment follows a microservices-based architecture with containerized services running on cloud infrastructure.

a) Infrastructure

* The backend and microservices are deployed on **Google Cloud Run** for auto-scaling and efficient resource utilization.
* MongoDB is hosted on **MongoDB Atlas** for managed database services.
* The OWL Knowledge Service is hosted on **a cloud-based RDF triplestore** such as Blazegraph.
* Videos and extracted frames are stored in **Google Cloud Storage (GCS)** for persistent and scalable storage.

b) Containerization

* Each microservice is packaged as a **Docker container** for easy deployment and isolation.
* Containers are managed using **Cloud Run** for automatic scaling and fault recovery.

c) API Gateway

* The backend acts as an **API gateway**, routing requests between the frontend and microservices.
* All microservices communicate with the backend using **REST APIs**.

d) Communication and Data Flow

* The backend interacts with the **video processing and YOLO detection services** asynchronously.
* The OWL Knowledge Service is queried using **SPARQL** to retrieve semantic traffic sign data.
* MongoDB stores all metadata and is accessed by the backend for retrieving detection results.

e) Security and Access Control

* Cloud Storage and database access are restricted to authorized services only.

The deployment model ensures that TraS remains **scalable, fault-tolerant, and optimized for efficient processing** while leveraging cloud infrastructure for performance and reliability.

**Future Improvements**

The TraS system is designed with scalability and modularity in mind, allowing for future enhancements to improve performance, accuracy, and usability. Below are potential areas for improvement:

a) **Real-Time Processing**

* Implement real-time video processing to detect traffic signs from live camera feeds instead of pre-recorded videos.
* Optimize the AI model to run efficiently on edge devices for real-time inference.

b) **Improved AI Model**

* Train the YOLO model with a larger dataset covering more diverse weather and lighting conditions.
* Use model optimization techniques such as quantization to improve inference speed.
* Implement a self-learning mechanism where user feedback refines detection accuracy.

c) **Enhanced Semantic Enrichment**

* Expand the OWL ontology to include more traffic regulations, country-specific rules, and road conditions.
* Improve SPARQL queries to provide more context-aware information.

d) **Better User Interaction**

* Allow users to edit detected traffic sign data and contribute to improving detection accuracy.