Smart Way Traffic - Software Design & Lifecycle Document (SDLC)

---

1. Project Overview

Smart Way Traffic is an intelligent traffic management system designed to analyze traffic flow, detect congestion, and identify emergency situations using AI-powered video analysis. The system provides real-time visualization and detailed reporting to optimize urban traffic control.

---

2. Technology Stack

A. Frontend (User Interface)

- **Framework**: React (v18+) with Vite for fast build tooling.
- **Styling**: Tailwind CSS for utility-first responsive design.
- **State Management**: React Hooks (`useState`, `useEffect`, `useRef`) for local state and stream handling.
- **Icons**: `react-icons` (Feather Icons).
- **Communication**: Fetch API for REST endpoints, `EventSource` for Real-time SSE (Server-Sent Events).

B. Backend (API & Orchestration)

- **Runtime**: Node.js with Express.
- **Database**: Supabase (PostgreSQL) for relational data storage `uuid`, `jsonb` support.
- **Authentication**: Supabase Auth (Google OAuth / Email).
- **AI Integration**: `openai` npm package configured for **OpenRouter** (GPT-4o) for high-level scene analysis.
- **File Handling**: `multer` for video upload management.

C. AI Engine (Computer Vision Core)

- **Language**: Python (3.11+).
- **Framework**: FastAPI for high-performance async API.
- **Vision Model**: Ultralytics YOLOv8 (Nano model) for object detection.
- **Tracking**: YOLOv8 `model.track` for persistent object tracking (ID assignment).
- **Image Processing**: OpenCV (`cv2` headless) for frame manipulation and drawing.
- **Concurrency**: `asyncio` and `sse_starlette` for streaming analysis data to frontend.

---

3. System Architecture

```
graph TD
   User[User / Admin] -->|Interacts| FE[React Frontend]

   subgraph "Application Layer"
       FE -->|REST API (Upload/Save)| BE[Node.js Backend]
       FE -->|SSE Stream (Visualize)| AI[Python AI Engine]
   end

   subgraph "Data Layer"
       BE -->|Read/Write| DB[(Supabase PostgreSQL)]
       BE -->|Uploads| FS[Local File System / Storage]
   end

   subgraph "Intelligence Layer"
```

```
        AI -->|Load Model| YOLO[YOLOv8 Model]
        BE -->|Analyze Snapshot| GPT[OpenRouter / GPT-4o]
    end

    AI -.->|Snapshot| FS
    BE -.->|Read Snapshot| FS
```

---

4. Functional Workflows

4.1. Video Upload & Processing

1. User selects a traffic video file on the **Dashboard/Simulation Page**.
2. Frontend uploads file to Backend `POST /api/videos`.
3. Backend saves file to `uploads/` and creates a record in `videos` table in Supabase.
4. Video becomes available in the selection dropdown.

4.2. Simulation & Visualization (Real-Time)

1. User selects a video and clicks **"Visualize"**.
2. Frontend opens an `EventSource` connection to AI Engine: `GET /api/live-detect-sse/?file=...`.
   AI Engine:
   - Reads video frame-by-frame using OpenCV.
   - Runs **YOLOv8 Tracking** on each frame to detect vehicles (Car, Bus, Truck, Motorcycle) and Signal Lights.
   - Assigns unique IDs to track total volume.
   - Encodes processing frame to Base64.
   - Streams JSON data (Frame + Counts + Status) back to Frontend.

   Frontend:
   - Renders Base64 frame to `<img src>`.
   - Updates Live Metrics (Vehicle Count, Congestion Status).

   User Controls:
   - **Pause**: Freezes the UI updates while maintaining state.
   - **Stop**: Ends the simulation and prepares data for saving.

4.3. Detailed Analysis & Reporting

1. On completion (or Stop), AI Engine captures a **Snapshot** of the busiest frame.
2. Frontend sends aggregated stats to Backend: `PUT /api/videos/:id/analysis`.
   Backend:
   - Reads the Snapshot image.
   - Sends Snapshot + Stats to **OpenRouter (GPT-4o)** via visual prompt.
   - **Prompt**: "Analyze for accidents, infrastructure (lanes), and signal compliance."
4. **OpenRouter** returns a natural language report.
5. Backend saves the report and stats into `traffic_logs` table.
6. Frontend displays the "AI Analysis Popup" immediately.

---

5. Database Schema

Table: `videos`

- `id` (UUID): Primary Key
- `filename` (Text): Original name

- `filepath` (Text): Storage path
- `status` (Text): processing / processed
- `analysis_summary` (JSONB): Basic counts

Table: `traffic_logs`

- `id` (UUID): Primary Key
- `video_id` (UUID): FK to videos
- `vehicle_count` (Int): Total traffic volume
- `detailed_analysis` (JSONB): Contains full `ai_report` text, congestion level, and breakdown.
- `emergency_detected` (Boolean): True if Ambulance/Firetruck/Accident found.
- `created_at` (Timestamp): Log time.

---

6. Directory Structure

```
smartway-traffic/
■■■ ai_engine/           # Python Service
■   ■■■ detector.py      # YOLO Logic & Tracking
■   ■■■ main.py          # FastAPI Routes
■   ■■■ requirements.txt
■■■ backend/             # Node.js Service
■   ■■■ index.js         # Express Server & OpenRouter integration
■   ■■■ schema.sql       # Database definitions
■   ■■■ uploads/         # Video storage
■■■ smarttraffic-frontend/ # React App
    ■■■ src/
    ■   ■■■ pages/       # Dashboard, Simulation, Profile
    ■   ■■■ components/  # Reusable UI
    ■   ■■■ App.jsx      # Routing
    ■■■ vite.config.js
```