Indiana University Southeast

Feasibility Report #3

TrafficLouisville Software Architecture Specification

Capstone Team: **So Much For Subtlety**

John Brown

Mason Napper

Aaron O'Brien

10/29/2024

Table of Contents:

1.1 Overview
1.2 Individual Contributions
1.3 Subsystem Decomposition
1.4 Hardware/Software Mapping
1.5 Persistent Data Management
1.6 Access Control and Security
1.7 Global Software Control
1.8 Boundary Conditions

1.1 System Overview

Traffic Louisville is a multi-component, combined hardware/software system that delivers near real-time information about the density of vehicle traffic on Louisville highways through an online map interface. It achieves this through accessing CCTV images publicly available through TRIMARC, a project of KYTC. It then analyzing the images with machine learning models that perform object detection, and after calculating the relative density of traffic at a geographic location, sends this information to be displayed on a custom heat-map of Louisville hosted on a webpage. TrafficLouisville is not intended to be a commercial system or to be used in critical applications. It is designed as part of an academic project that illustrates the potential of utilizing machine learning for near-real time applications.

1.2 Individual Contributions

The members of the development team So Much For Subtlety include John Brown, Mason Napper, and Aaron O'Brien. We are grateful for help received from our capstone professor, Dr. Ronald Finkbine and our sponsor, Chris Sexton. We would also like to thank the Kentucky Transportation Cabinet Office of Information Technology, both for providing a valuable public information resource and for their guidance in our project.

Team Member Responsibilities:

John Brown: Team Leader, responsible for developing the backend services required for the project. For this particular report, responsible for documenting the overview, subsystem decomposition, and hardware/software mapping.

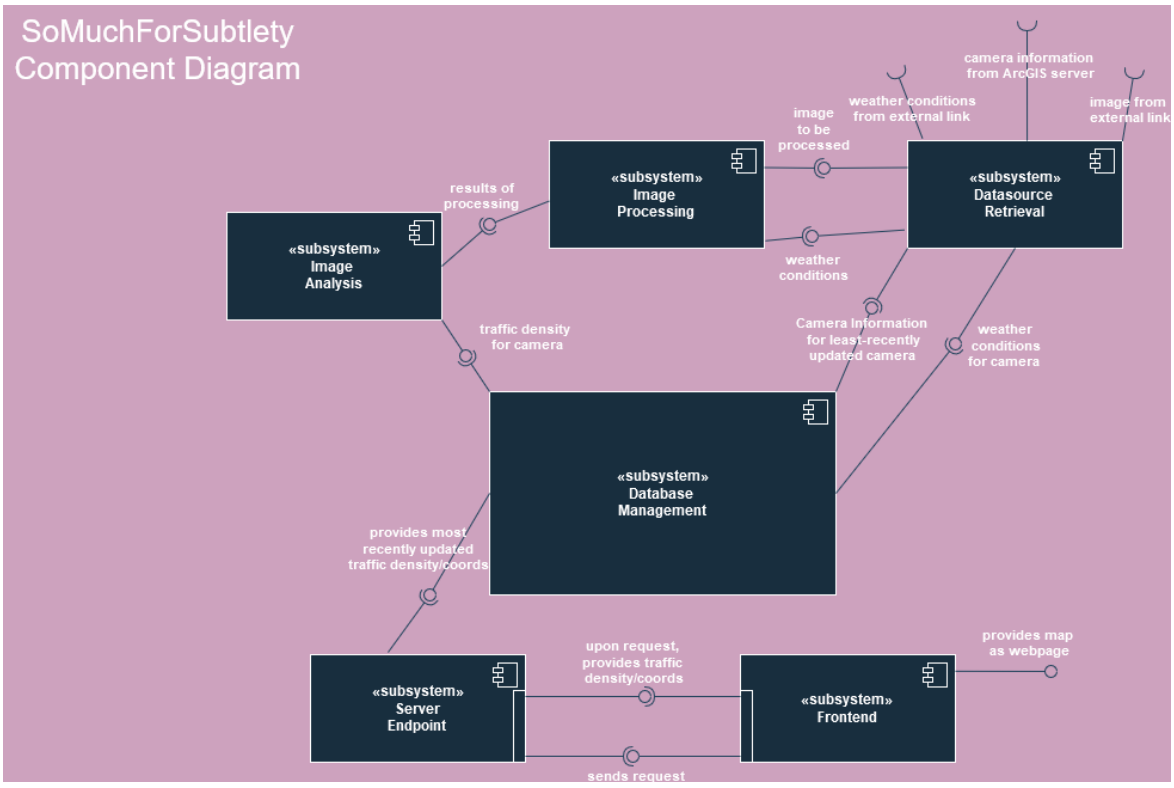
Mason Napper: Responsible for developing the public facing interface of Traffic Louisville, also contributes hardware to the project. For this particular report, responsible for documenting the global software control.

Aaron O'Brien – Responsible for developing the machine learning component of our project through training models and governing their output, also contributes hardware for the project. For this particular report, responsible for documenting access control and security, and boundary controls.

1.3 Subsystem Decomposition

The following subsystems comprise our project:

- **Data Source Retrieval** - Downloading an image at a fixed rate from CCTV camera links, retrieving camera metadata from ArcGIS REST API, retrieving weather conditions that affect image processing, responsible for error handling that may arise from external factors
- **Image Processing** - Applying appropriate model for detecting and classifying vehicles within retrieved images using the YOLOv8 model, will potentially account for camera orientation and zooming through TBD
- **Image Analysis** – Converting object detection results into traffic counts, storing traffic data and calculating density
- **Database Management**– Stores information retrieved from data source, current and historical traffic counts to allow for calculating traffic density, and supports CRUD operations from other subsystems to keep them independent.
- **Server Endpoint** – Receive GET request from frontend and respond by retrieving information from database
- **Frontend** – Sending request to endpoint, receiving it and displaying it at the correct position on the heat map, displaying other relevant information



1.4 Hardware-Software Mapping

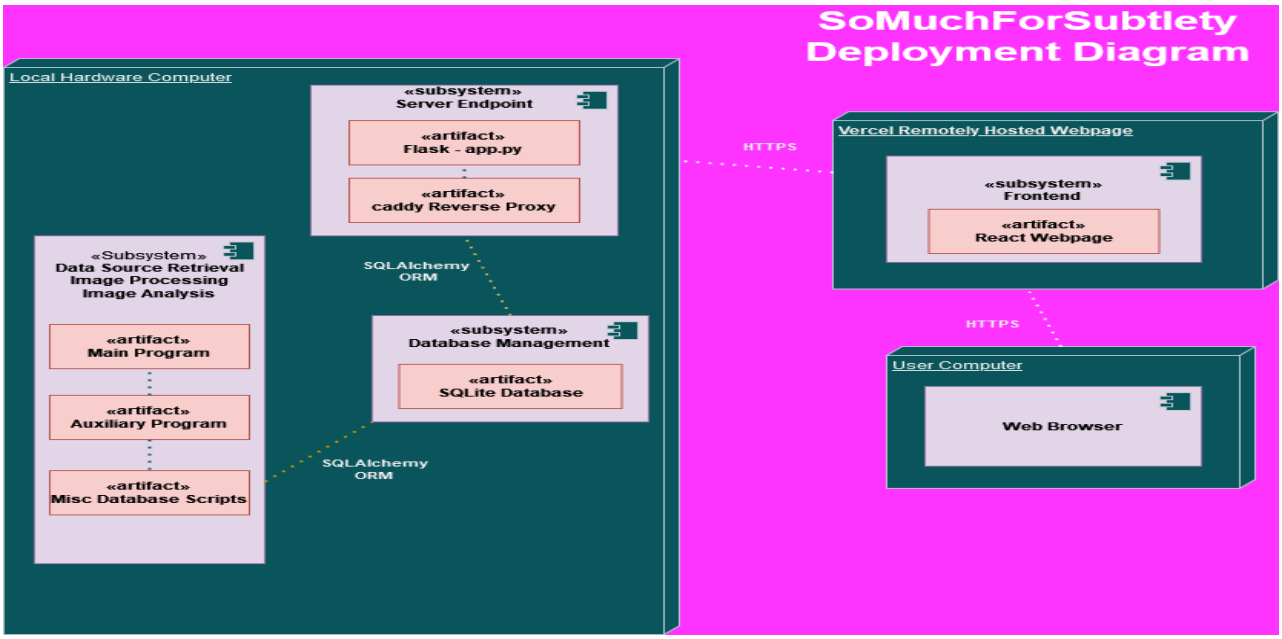
The system will accomplish the preceding system functions through the following hardware/software components:

Running on local, privately owned hardware:

- **Local Processing** – The data source retrieval, image processing, and image analysis functions will be accomplished by two programs run on local hardware: the main program and the auxiliary program. The main and auxiliary programs will be written and run as Python programs. The image processing will be accomplished with Yolov8, a machine learning model. Python scripts to create the database, populate, and periodically update the database are also available in local processing.
- **Local Database** – A database will be maintained on local hardware that fulfills the database functions of the system. The database will be an SQLite database.
- **Local Server Endpoint** – A backend API server run as a separate program on local hardware fulfills the server endpoint function. The backend API server will be a Python- Flask server.

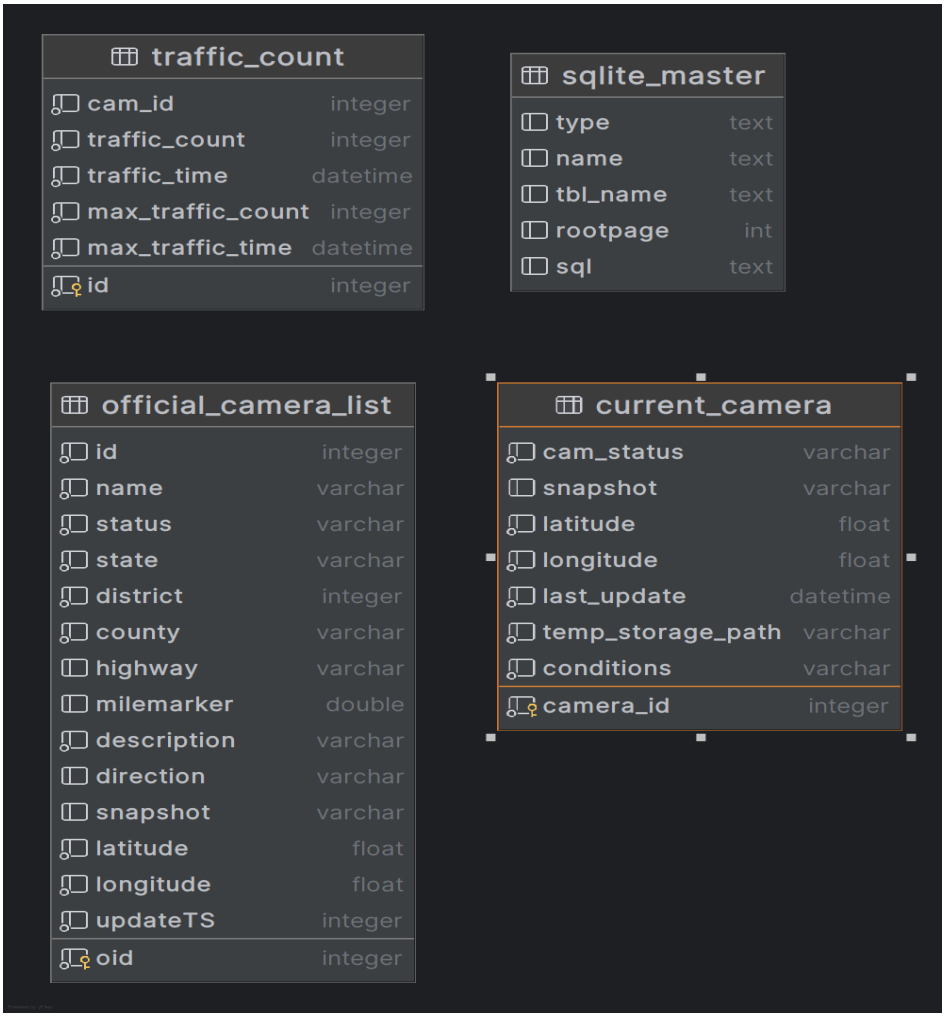
Running remotely on a cloud hosting platform:

Remote Frontend – An application running on a cloud platform that hosts a public web page will accomplish the frontend functions. The application will be a React application hosted on Vercel, a cloud-based hosting platform.



1.5 Persistent Data Management

Persistent data in TrafficLouisville includes a database that contains camera metadata and current/historical traffic camera records, as well as temporary images saved to a directory. The data is stored locally in a SQLite database that has three tables. The first table describes the current rotation of cameras, with a camera id, status, coordinates, time of last update, a temporary storage path for a downloaded picture, and current conditions. The second table describes an hourly update of the official camera data from KYTC ArcGIS server. The third table consists of records of traffic counts for a particular camera id, as well as the max traffic count for that camera so density can be calculated. The temporary images directory contains a single image for each camera, the last one that was accessed. Each time a new picture for a camera is downloaded, it replaces the old one. There is no long-term storage of pictures. The table 'sqlite_master' in the diagram below can be ignored.



1.6 Access Control and Security

Users:

- **Operations:** Users access TrafficLouisville through a public webpage, where they can view the traffic density heatmap but have no ability to interact with or modify backend data.
- **Authentication**: No authentication is required for users as the webpage is a simple display and does not involve user data or accounts.

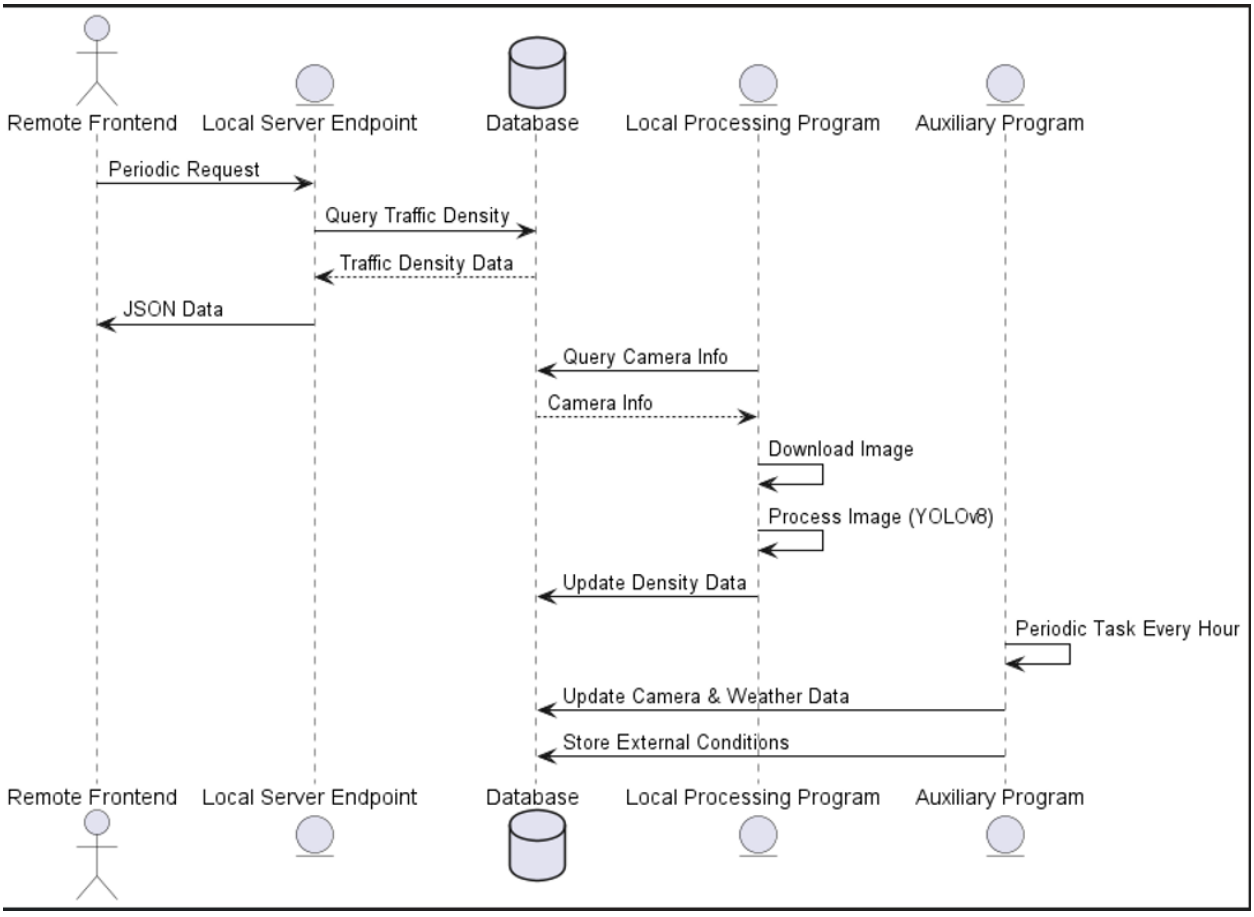
System Administrators:

- **Operations:** System administrators have the privilege to access the local server and may:
  - o Monitor the Flask server.
  - o Monitor processes and their resource consumption.
  - o Manage the local database by manually updating records.
  - o Set up new traffic cameras to be processed.
  - o Restart backend processes if necessary.
  - o Configure reverse proxy settings.
- **Authentication:** Authentication for system administrators is required for server access.Specifically, a secure TeamViewer ID and password is required for administrators to remotely access the server.

Security Provisions: The local server endpoint is protected by a reverse proxy restricting exposure to public internet threats, and all traffic between the frontend and backend is routed over HTTPS to prevent unauthorized access. Additionally, Vercel provides secure access controls for managing the frontend.

1.7 Global Software Control

TrafficLouisville utilizes procedural and event-driven control flow. The main program operates under a procedural loop, processing a camera and generating results repeatedly in a cycle every ten seconds. If a cycle takes less than ten seconds, the procedure will wait until ten seconds have elapsed, if a cycle is more the control flow continues as normal. The remote frontend is a mixed procedural/event-driven control flow, with the frontend sending requests on a timer but waiting for a response. The local server endpoint operates on an event-driven control flow, waiting for a request from the frontend.



1.8 Boundary Conditions

1. System Startup, Initialization, and Shutdown

- **Startup and Initialization:**

  o The main and auxiliary programs will automatically start on the server and initiate the traffic monitoring loop and populating the database. The main program loops every 10 seconds, and the auxiliary program loops every hour.

  o The Flask server for the backend API will launch with the main server startup, run continuously, and listen for requests from the frontend routed through a reverse proxy.

- **Shutdown:**

  o When shutting down, both the main and auxiliary programs will stop their processing loops gracefully, ensuring that database connections close cleanly.

  o The local server endpoint will also shut down and release any open ports or server resources.

2. Error Handling and Exceptions

- **Error Responses:**

  o For errors during camera image retrieval, weather data fetching, or database access, each function in the main and auxiliary programs includes error handling. When encountering errors such as unavailable traffic camera URLs or database connection errors, the system will skip to the next item in its loop or temporarily pause processing to avoid crashes.

  o In the event of repeated errors, the system may log errors for later review allowing administrators to address persistent issues.

3. Maintenance and Data Migration

- **Periodic Maintenance:**

  o Daily: Monitor the system for any errors logged by the main and auxiliary programs ensure that the Flask server is functioning smoothly.

  o Weekly: Conduct a manual review of the data stored in the SQLite database to ensure there are no inconsistencies, outliers, or unusual behavior.

  o Monthly: Verify the local hardware's performance, especially CPU utilization during Yolov8 model runs, and check for any updates or issues with dependencies like Flask, SQLite, Ultralytics, and the YOLOv8 model.

  o Yearly: Evaluate the overall architecture and determine if any hardware or software components need an upgrade.

- **Server Migration and Bulk Data Transfer:**

  o Migration: Every 3-4 years, as part of the hardware upgrade cycle, the system can be migrated to a new server by reinstalling the necessary software components and transferring all code and database files.

  o Data Dump/Load: The SQLite database supports export functionalities, allowing bulk dumping of all data to a file format, such as CSV or JSON, which can then be re-imported into a new SQLite database on an upgraded server.