Technical Report — TripLens Project

1. Problem Framing and Dataset Analysis

Dataset Description

Our project aims to visualize and analyze urban mobility patterns using a dataset containing trip information such as:

- Trip ID
- Vendor ID
- Pickup & Dropoff location (latitude, longitude)
- Pickup & Dropoff time
- Trip duration and others

The data set provided was not clear with improper data and not organized even to the extent when u see it to analyse it would be a hard task. This is where we decided to do our magic and split the data and organize it in proper and clear way

Challenges

During initial inspection, we identified:

- Unclear or so called missing dataset in like vendor detail.
- Integration of map in our front end

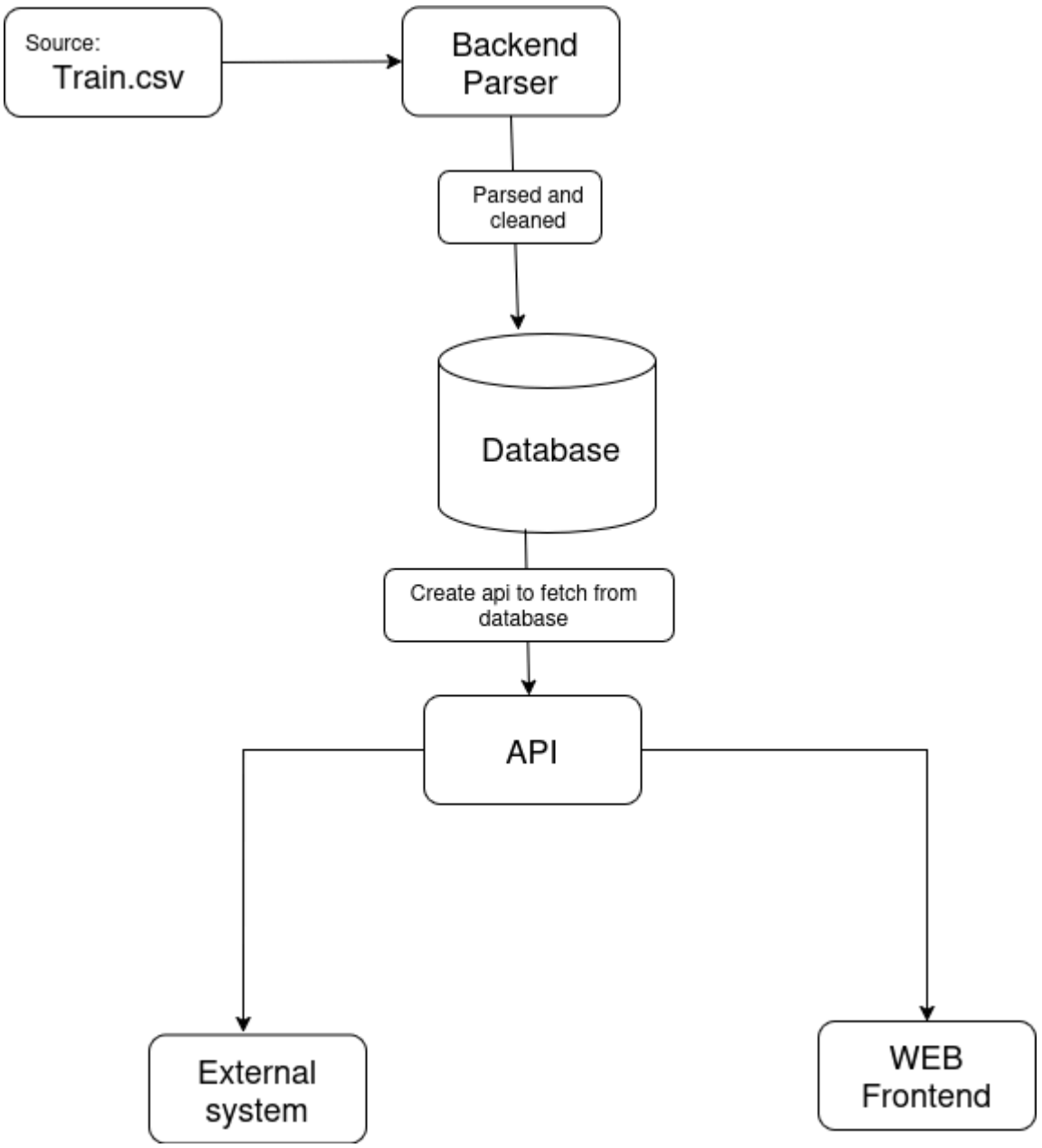
Assumptions Made

We created vendor table and assigned vendor name and descriptions

---

2. System Architecture and Design Decisions

Architecture Overview



We created the file that is going to process our csv file and create a clear database and after creating it we made an api that is going to be used for both our front end and external systems wanting to use our design using fastapi.

We added demo data for the vendors and separated the locations into its own table to create a relational database.

We used techniques like skipping and limiting queries to improve speed of database queries and api calls and allow for pagination of data.

We also implemented google map functionality with the longitudes and latitudes mapped for improved user experience.

3. Components Description

3.1 Data Source – `Train.csv`

- Format: CSV
- Contains raw data to be processed and stored.
- Uploaded or placed in a specific directory on the server.

3.2 Backend Parser

- Language: Python
- Responsibilities:
    - Load CSV file.
    - Validate and clean data (e.g., assigning vendor values, type conversions).
    - Insert parsed data into the database.
- Modules used: `pandas`, `sqlite3, os`.

3.3 Database

- Type: SQLite (local development)
- Tables: Designed to match the structure of `Train.csv` (e.g., trips, users, locations depending on dataset) but in a clear and understandable way.
- Features:
    - Indexes for faster querying.
    - Foreign key constraints enabled.
    - Primary keys to ensure unique records.

3.4 API Layer

- Framework: FastAPI
- Endpoints Examples:
    - `localhost:8080/trip` → Fetch all records
    - `localhost:8080/trip/{id}` → Fetch specific record
    - `localhost:8080/trip/month/6=...` → Fetch by filtering data by months **eg:** of response:


```
{"total_trip": 234316,"skip": 6, "limit": 20,"month_name": "June",
"average_trip_duration": 1013.3672263097698, "total_passenger":
388853, "data": [{"Trip_id": "id3321406","vendor_name": "SafeBoda",
"trip_pickup_date": "2016-06-03 08:15:05", "trip_dropoff_date":
"2016-06-03 08:56:30", "trip_passenger_count": 1,
"location_pickup_longitude": -73.95523071289062,
"location_pickup_latitude": 40.77713394165039,
"location_dropoff_longitude": -73.78874969482422,
"location_dropoff_latitude": 40.64147186279297,
"trip_store_and_fwd_flag": "N", "trip_trip_duration": 2485 }, {
"Trip_id": "id2104175", "vendor_name": "Yego Taxi",
"trip_pickup_date": "2016-06-20 23:07:16", "trip_dropoff_date":
"2016-06-20 23:18:50", "trip_passenger_count": 1,
"location_pickup_longitude": -73.95843505859375,
"location_pickup_latitude": 40.713191986083984,
"location_dropoff_longitude": -73.9495391845703,
"location_dropoff_latitude": 40.68025207519531,
"trip_store_and_fwd_flag": "N", "trip_trip_duration": 694} ]}
```


    - `localhost:8080/vendor =...` → Fetch all vendors
- Features:
    - Returns data in JSON format.
    - Supports filtering, pagination, and search(eg: if you check this example of json we added, they consist of limit and skip(which will help us during the pagination on the frontend and faster processing)).
    - Includes error handling.

3.5 Web Frontend

- Technology: HTML / CSS / JavaScript
- Responsibilities:
    - Fetch data from the API using `fetch()`.
    - Display data in a clean, user-friendly table and cards.
    - Include search and filter UI if implemented.
- Features:
    - Real-time updates (if API is dynamic).
    - Responsive design for different devices.

3.6 External Integrations (Optional)

- Third-party systems can consume the API directly using its public endpoints.
- Allows future integrations without changing the core architecture.

4. System Flow

1. CSV file is uploaded or placed in the system.
2. Parser cleans and validates data.
3. Clean data is inserted into the database.
4. API exposes endpoints to access the data.
5. Web frontend requests data via API and renders it to users.
6. Optional external systems can also consume the same API.

5. Non-Functional Requirements

Category	Description
Performance	API response time < 500ms for typical queries.
Scalability	Frontend and API are decoupled → supports future scaling.
Maintainability	Clear separation of concerns across components.
Security	Input validation and basic error handling implemented.

Portability	Can run locally or be deployed on a server with minimal configuration.