LAMINA
STUDIOS

Data Science Internship

Individual Weekly Task Documentation

Week #4:
Smart Warehousing & Inventory Management

Jimalyn B. Del Rosario

Table Contents

I. My Summarized Daily Logs
II. Lecture Report
III. Individual Hands-on Activity
IV. Team Task Progress Report

I. My Summarized Log

Encompasses five weekdays from Monday to Friday with 8-hours spent per day

DAY 16
Week 4
Monday
08/04/2025

[08:00 AM - 05:00 PM]

- Reviewed new tasks for the week in 5-week Data Science Interns Document
- Updated my local repository for documentation of previous weeks' tasks
- Created a schedule to allocate days and hours for all tasks required in Week 4
- Started watching first batch of Week 4 Lectures:
  - RFID Smart Warehousing Cloud Helps You Make a Smart Inventory Decision
  - What is Real-Time Data Analytics
  - Real Time Analytics in IoT Scenarios
  - The Role of AI and IoT in Modern Retail
- Created a Week 4 Individual Weekly Task Documentation file
- Wrote lecture report e.g. what I took or learned from the video lectures
- Met with Dats Team on discord to have recalibration with team task distribution

DAY 17
Week 4
Tuesday
08/05/2025

[08:00 AM - 05:00 PM]

- Continued the second batch of Week 4 Lectures:
  - Supply Chain Optimization: Leveraging Data and Emerging Technologies,
  - Can Inventory Optimization Improve Order Preparation Processes in Warehouses?,
  - Inventory Optimization
- Wrote lecture report e.g. what I took or learned from the video lectures
- Met with my Insights team mate to discuss our distribution of tasks for future team tasks.
- Read the individual guide files for Week 3 Team Task and Week 4 Individual Hands-on Activity

DAY 18
Week 4
Wednesday
08/06/2025

[08:00 AM - 05:00 PM]

- Started on Week 4 Individual Hands-on Activity Smart Warehouse & Inventory Management with Barcode Scanning, Visualization via Metabase & Chart.js
- Prepared Mock Inventory dataset through AI generation
- Set up PostgreSQL database using given schema
- Set up full architecture and secured communication between all components
  - Built flask microservice
  - Integrated Laravel API with flask
  - Tested POST on localhost:500/rfid-scan route using POSTMAN
  - Attempted Chart.js visualization but faced errors
  - Synced inventory database in Metabase
  - Created Metabase dashboards
- Compiled screensnaps of each step taken to build the system
- Wrote the documentation document for Week 4 Hands-on activity

DAY 19
Week 4
Thursday
08/07/2025

[08:00 AM - 05:00 PM]

- Started on Week 3 Team Task Forecasting Delivery Time with Proximity-Based Notifications
- Set up full architecture and secured communication between all components
    - Cleaned and prepared historical delivery dataset
    - Created util.py to train and ETA prediction model using Random Forest and saved the trained model as delivery_eta_model.pkl
    - Created index.html file to have a web view of real-time distance calculation. It takes in coordinates and will display the ETA with its respective Status
    - Created a flask API to return ETA predictions in index.html
- Met with the Data Science Team to realignment
- Worked on BONUS tasks for Week 3 Team Task
    - Created worker.py for redis queue function to handle GPS data poling
    - Created new version of files for Traccar integration to replace manual GPS assignment
        - Set up Traccar Client on my phone device and Traccar UI in my desktop to have connection
        - Created get_gps.py to get live location from traccar api
        - Created new app.py with /live_eta routing that imports get_latest_position function
        - Created new index.html to fetch from the /live_eta route
    - Worked on Metabase integration of the database
        - Created delivery_db in pgAdmin
        - Import historical_deliveries.csv
        - Created new column travel_time_minutes
        - Converted time columns in time date format and updated content of travel_time_minutes column
        - Connected to Metabase and created dashboard
- Compiled screensnaps of each step taken to build the system
- Wrote the documentation document for Week 4 Hands-on activity

DAY 20
Week 4
Friday
08/08/2025

[08:00 AM - 05:00 PM]

- Worked on compiling all activity logs for the past days
- Compiled all documentations in the Weekly Documentation file
- Worked on the notarization of my Internship Requirements
- Updated MyHours to log all hours for Week 4
- Collected all the links to lecture videos I watched
- Collected all the links to the deliverables
- Organized all the deliverable files
- Submitted all the deliverables to the drive folder.

II. Lecture Report

Week 4 Lectures

- RFID Smart Warehousing Cloud Helps you Make a Smart Inventory Decision

  This lecture talked about what happens after checking raw materials. Once they pass quality control, they're sent to the warehouse. Warehousing involves three main things: **storing, tracking inventory, and distributing.** With smart warehousing using CFM's cloud system, everything becomes more efficient. RFID labels let the system register and track items in real time without manual work. There's no need for scanning or stock taking by hand. Machines handle tracking and calculations, so decisions can be made faster. The data helps improve how fast goods move through the system and lowers costs.

- What is Real-Time Data Analytics

  This lecture explained what real-time analytics is and why it's important. Real-time data means using and analyzing information as soon as it's available. It discusses how companies are now realizing that data loses value quickly, so acting fast is key. **Real-time analytics helps businesses make faster decisions, improve services, and automate processes.** One example is how banks detect fraud by monitoring card activity instantly. Apps also use real-time data to show users the right ads at the right moment, like offering travel insurance when someone's at the airport. It's also the tech behind things like self-driving cars and autonomous ships, which rely on constant sensors and camera data. To use this, companies need live data feeds and the systems to process it right away. The good part is that services like AWS, Microsoft, and IBM make this tech available even for smaller businesses.

- Real Time Analytics in IoT Scenarios

  In this talk, the speaker discusses how IoT, edge computing, and Azure services work together in real-world scenarios. What stood out to me was how everything is connected: traffic cameras, edge devices, cloud storage, and analytics, all streaming and processing data in real time. He showed how traffic data (like license plates and car speeds) can be captured at the edge, processed locally, then sent to the cloud for deeper analysis or automation, like issuing speeding tickets. He explained the difference between edge and cloud: edge is for fast, local processing (like recognizing a license plate), and cloud is for storage, analysis, and long-term actions. Tools like IoT Hub, Azure Stream Analytics, and Logic Apps all play a role, and surprisingly, most of it requires little or no code just setting up configurations and queries. I liked how stream analytics queries were used to detect speeding by measuring time between two cameras, and how the data triggered workflows automatically. Also, Databricks was used for more complex data analysis, like spotting suspicious vehicles. **My takeaway is that IoT's real power is in the data and how it's used, not just the devices.**

- The Role of AI and IoT in Modern Retail

  This lecture focused on why companies struggle with data. The main idea was that businesses often collect too much data without knowing how to use it. A lot of it ends up sitting unused, or gets analyzed too late to be useful. What companies really need is to make decisions fast, based on up-to-date data, not old reports from months ago.
  He also pointed out that dashboards and reports don't really help people take action unless there's a clear signal in all the noise. **Just throwing more data at people doesn't work. Instead, companies should focus on outcomes, figuring out what actions to take based on data, not just staring at charts.** He gave the example of a factory where workers see a graph but don't know what it means or what to do. The big takeaway is that real-time, actionable insights are way more useful than just collecting and reporting data.

- Supply Chain Optimization: Leveraging Data and Emerging Technologies

  In this video, Lorenza's team worked on optimizing inventory for small fair trade shops. These shops have little space and limited management tools, so restocking is tough. They built a two-step system: one part tracks stock levels, the other suggests when and what to restock. To predict demand, they used different forecasting models and picked the most accurate. For the optimization, they used linear programming because it's fast, transparent, and works even with small datasets. The goal was to boost profits while keeping stock low.

- Can Inventory Optimization Improve Order Preparation Processes in Warehouses?

  This lecture explained how inventory optimization can make order preparation in warehouses much more efficient. By reducing both stock shortages and overstocking, it ensures that the right items are always available when needed. This helps workers prepare orders faster and with fewer delays. Organized inventory systems, especially when using SKUs, also speed up item retrieval and improve overall workflow. Another key point was that inventory optimization improves accuracy. With better visibility and fewer manual errors, there are fewer returns and happier customers. Tools like real-time analytics and warehouse management systems help managers plan better and optimize picking methods. Techniques like wave picking or multi-order processing reduce walking time and boost efficiency. In the end, aligning inventory levels with warehouse layout and smart planning leads to faster, more reliable, and more cost-effective order fulfillment.

- Inventory Optimization

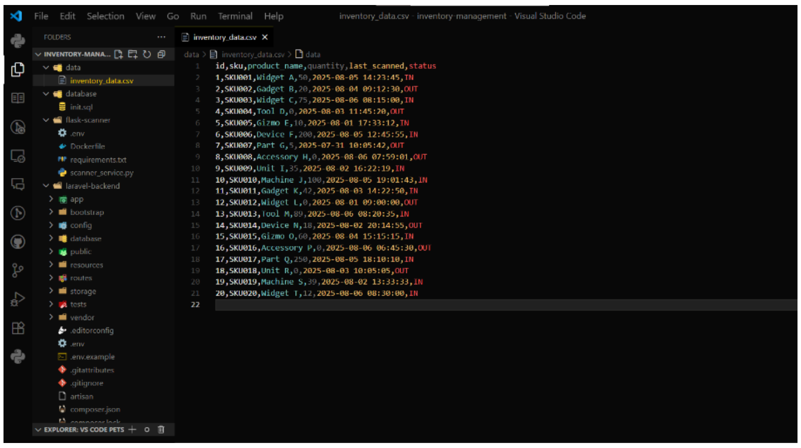
  This lecture helped me really understand how complex inventory optimization can get, especially in large businesses. Before this, I mostly thought of inventory as just making sure there's enough product in stock. But now I see it's about finding a balance between having enough to meet demand, keeping costs down, and managing all the uncertainty that comes with supply and demand. What stood out to me was how inventory isn't something you look at on its own it has to fit into the bigger picture with demand planning, supply planning, and overall business goals. One of the many things I found really interesting was how they talked about modeling uncertainty. I hadn't really thought about using things like demand variability or supply delays in such a structured way. The idea of safety stock being calculated based on service levels and data made it feel less like guessing and more like something strategic. The MEIO (multi-echelon inventory optimization) concept was new to me too managing inventory across different parts of the supply chain, not just in one warehouse. It made me realize how much coordination is actually required behind the scenes. I also liked the part about the Enterprise Knowledge Graph and the Graph Cube. I wasn't familiar with those before, but they made sense once explained. Basically, it's a way to map out how everything is connected in the supply chain, products, locations, timelines, and then use that to plan smarter. It sounded like a way to make better, faster decisions with real data, not just gut feeling or rough estimates.

  The platform demo was helpful in showing how this all works in practice. I liked seeing how users could set service level goals, simulate different stocking strategies, and actually see the impact on cost and availability. The case study at the end was a nice touch too as it showed that these tools aren't just for show, but actually lead to big improvements, like cutting working capital by 20%. This session really changed how I look at inventory management. It's not just about keeping shelves full, it's about using data and smart systems to plan better, respond faster, and run things more efficiently. I'll definitely keep this perspective in mind moving forward.

III. Individual Hands-on Activity

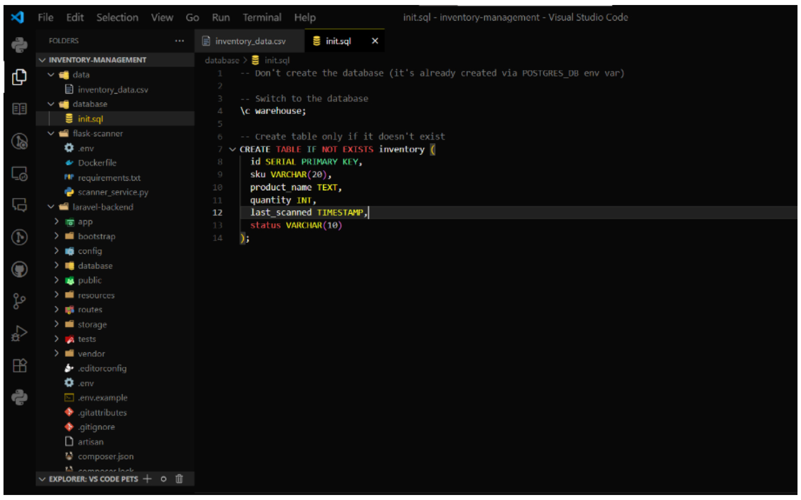
Smart Warehouse & Inventory Management with Barcode Scanning, Visualization via Metabase & Chart.js

- STEP 1: Prepare Mock Inventory Dataset

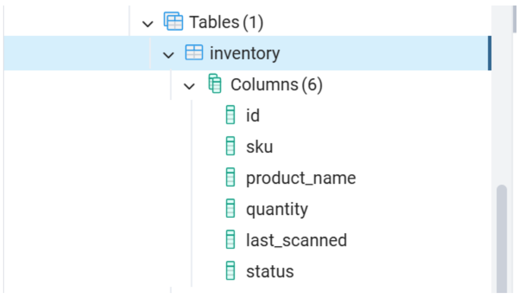


```
id,sku,product_name,quantity,last_scanned,status
1,SKU001,Widget A,50,2025-08-05 14:23:45,IN
2,SKU002,Gadget B,20,2025-08-04 09:12:30,OUT
3,SKU003,Widget C,75,2025-08-06 08:15:00,IN
4,SKU004,Tool D,0,2025-08-03 11:45:20,OUT
5,SKU005,Gizmo E,10,2025-08-01 17:33:12,IN
6,SKU006,Device F,200,2025-08-05 12:45:55,IN
7,SKU007,Part G,5,2025-07-31 10:05:42,OUT
8,SKU008,Accessory H,0,2025-08-06 07:59:01,OUT
9,SKU009,Unit I,35,2025-08-02 16:22:19,IN
10,SKU010,Machine J,100,2025-08-05 19:01:43,IN
11,SKU011,Gadget K,42,2025-08-03 14:22:50,IN
12,SKU012,Widget L,0,2025-08-01 09:00:00,OUT
13,SKU013,Tool M,89,2025-08-06 08:20:35,IN
14,SKU014,Device N,18,2025-08-02 20:14:55,OUT
15,SKU015,Gizmo O,60,2025-08-04 15:15:15,IN
16,SKU016,Accessory P,0,2025-08-06 06:45:30,OUT
17,SKU017,Part Q,250,2025-08-05 18:10:10,IN
18,SKU018,Unit R,0,2025-08-03 10:05:05,OUT
19,SKU019,Machine S,39,2025-08-02 13:33:33,IN
20,SKU020,Widget T,12,2025-08-06 08:30:00,IN
```

- STEP 2: Set Up PostgreSQL Database

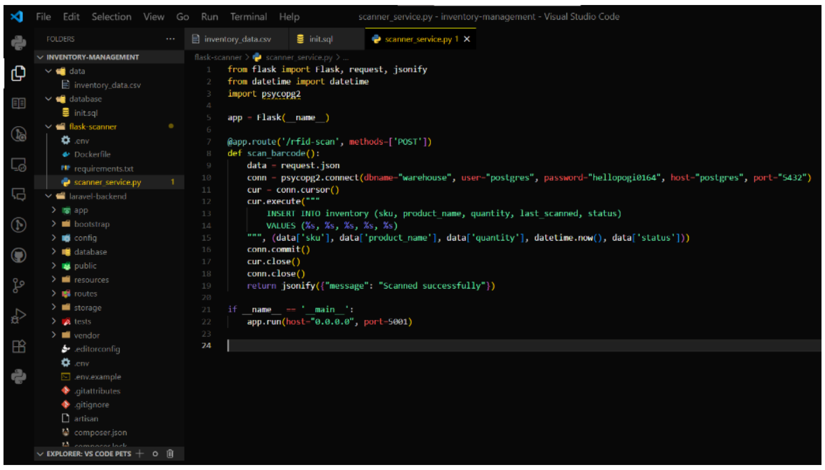


```sql
-- Don't create the database (it's already created via POSTGRES_DB env var)

-- Switch to the database
\c warehouse;

-- Create table only if it doesn't exist
CREATE TABLE IF NOT EXISTS inventory (
  id SERIAL PRIMARY KEY,
  sku VARCHAR(20),
  product_name TEXT,
  quantity INT,
  last_scanned TIMESTAMP,
  status VARCHAR(10)
);
```



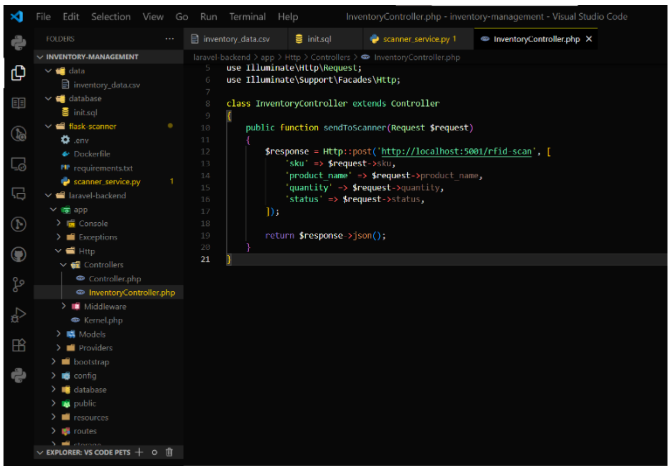
```
Tables (1)
  inventory
    Columns (6)
      id
      sku
      product_name
      quantity
      last_scanned
      status
```

- STEP 3: Build Flask Microservice for Barcode Scanning

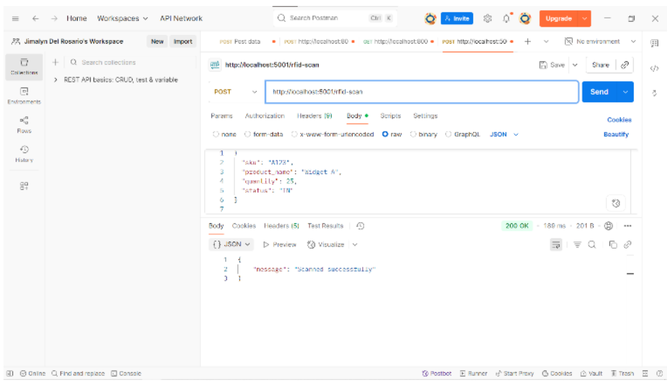


```python
from flask import Flask, request, jsonify
from datetime import datetime
import psycopg2

app = Flask(__name__)

@app.route('/rfid-scan', methods=['POST'])
def scan_barcode():
    data = request.json
    conn = psycopg2.connect(dbname="warehouse", user="postgres", password="hellopogi0164", host="postgres", port="5432")
    cur = conn.cursor()
    cur.execute("""
        INSERT INTO inventory (sku, product_name, quantity, last_scanned, status)
        VALUES (%s, %s, %s, %s, %s)
    """, (data['sku'], data['product_name'], data['quantity'], datetime.now(), data['status']))
    conn.commit()
    cur.close()
    conn.close()
    return jsonify({"message": "Scanned successfully"})

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5001)
```

- STEP 4: Integrate Laravel API with Flask



```php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Http;

class InventoryController extends Controller
{
    public function sendToScanner(Request $request)
    {
        $response = Http::post('http://localhost:5001/rfid-scan', [
            'sku' => $request->sku,
            'product_name' => $request->product_name,
            'quantity' -> $request->quantity,
            'status' => $request->status,
        ]);

        return $response->json();
    }
}
```



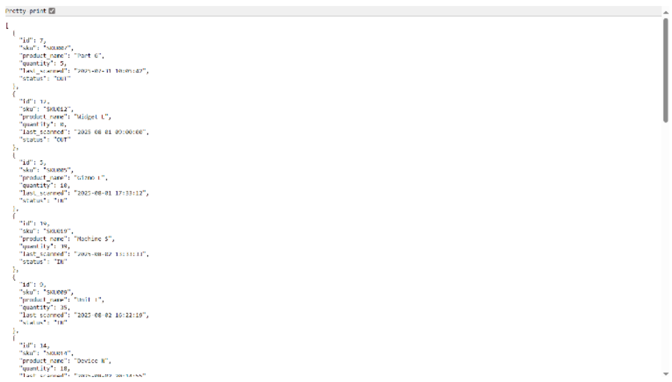
POST http://localhost:5001/rfid-scan

```json
{
  "sku": "A123",
  "product_name": "Widget A",
  "quantity": 25,
  "status": "IN"
}
```

200 OK · 189 ms · 201 B

```json
{
    "message": "Scanned successfully"
}
```

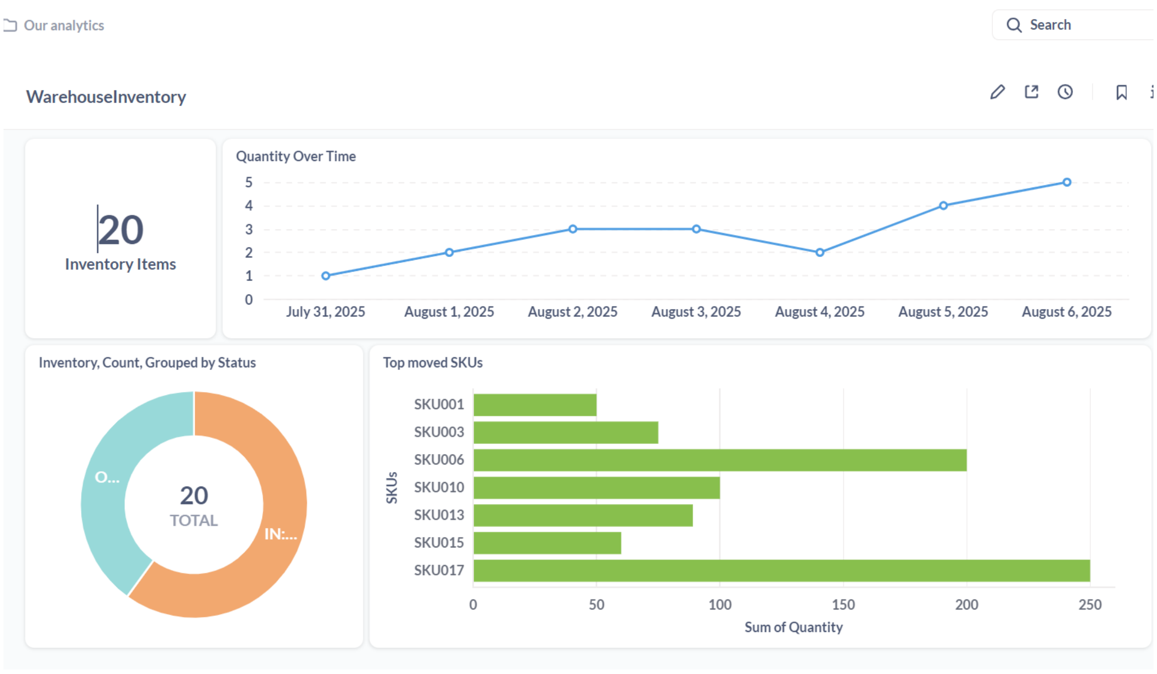
- STEP 5: Visualize with Chart.js (Frontend)



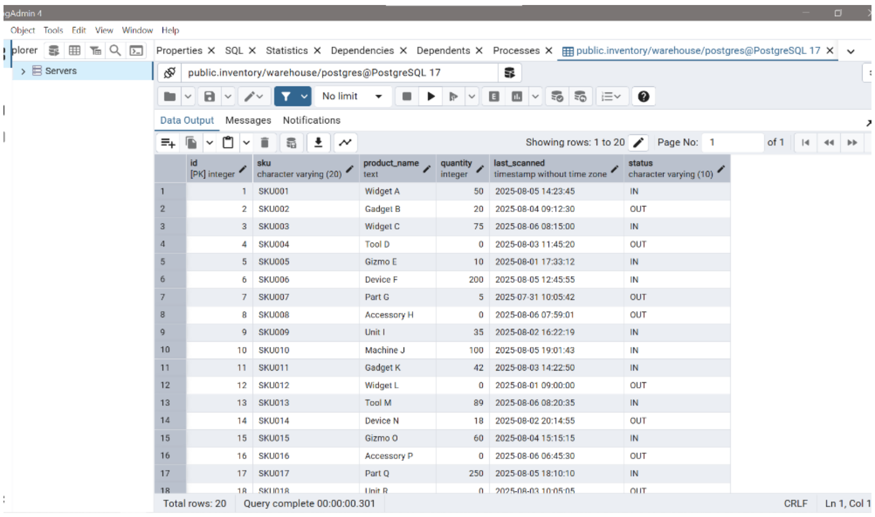
Pretty-print ☑

```
[
  {
    "id": 7,
    "sku": "SKU007",
    "product_name": "Part G",
    "quantity": 5,
    "last_scanned": "2025-07-31 10:05:42",
    "status": "OUT"
  },
  {
    "id": 12,
    "sku": "SKU012",
    "product_name": "Widget L",
    "quantity": 0,
    "last_scanned": "2025 08 01 09:00:00",
    "status": "OUT"
  },
  {
    "id": 5,
    "sku": "SKU005",
    "product_name": "Gizmo E",
    "quantity": 10,
    "last_scanned": "2025-08-01 17:33:12",
    "status": "IN"
  },
  {
    "id": 19,
    "sku": "SKU019",
    "product name": "Machine S",
    "quantity": 39,
    "last_scanned": "2025-08-02 13:33:33",
    "status": "IN"
  },
  {
    "id": 9,
    "sku": "SKU009",
    "product_name": "Unit I",
    "quantity": 35,
    "last scanned": "2025-08-02 16:22:19",
    "status": "IN"
  },
  {
    "id": 14,
    "sku": "SKU014",
    "product_name": "Device N",
    "quantity": 18,
```

- STEP 6: Set Up Metabase Dashboard



- Inventory Dataset



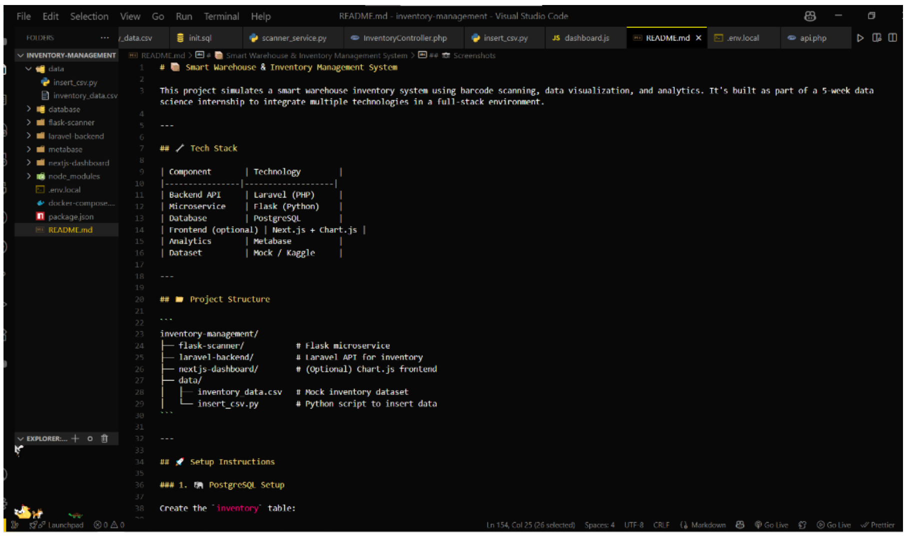
public.inventory/warehouse/postgres@PostgreSQL 17

Showing rows: 1 to 20 Page No: 1 of 1

	id [PK] integer	sku character varying (20)	product_name text	quantity integer	last_scanned timestamp without time zone	status character varying (10)
1	1	SKU001	Widget A	50	2025-08-05 14:23:45	IN
2	2	SKU002	Gadget B	20	2025-08-04 09:12:30	OUT
3	3	SKU003	Widget C	75	2025-08-06 08:15:00	IN
4	4	SKU004	Tool D	0	2025-08-03 11:45:20	OUT
5	5	SKU005	Gizmo E	10	2025-08-01 17:33:12	IN
6	6	SKU006	Device F	200	2025-08-05 12:45:55	IN
7	7	SKU007	Part G	5	2025-07-31 10:05:42	OUT
8	8	SKU008	Accessory H	0	2025-08-06 07:59:01	OUT
9	9	SKU009	Unit I	35	2025-08-02 16:22:19	IN
10	10	SKU010	Machine J	100	2025-08-05 19:01:43	IN
11	11	SKU011	Gadget K	42	2025-08-03 14:22:50	IN
12	12	SKU012	Widget L	0	2025-08-01 09:00:00	OUT
13	13	SKU013	Tool M	89	2025-08-06 08:20:35	IN
14	14	SKU014	Device N	18	2025-08-02 20:14:55	OUT
15	15	SKU015	Gizmo O	60	2025-08-04 15:15:15	IN
16	16	SKU016	Accessory P	0	2025-08-06 06:45:30	OUT
17	17	SKU017	Part Q	250	2025-08-05 18:10:10	IN
18	18	SKU018	Unit R	0	2025-08-03 10:05:05	OUT

Total rows: 20 Query complete 00:00:00.301

- README guide



README.md - inventory-management - Visual Studio Code

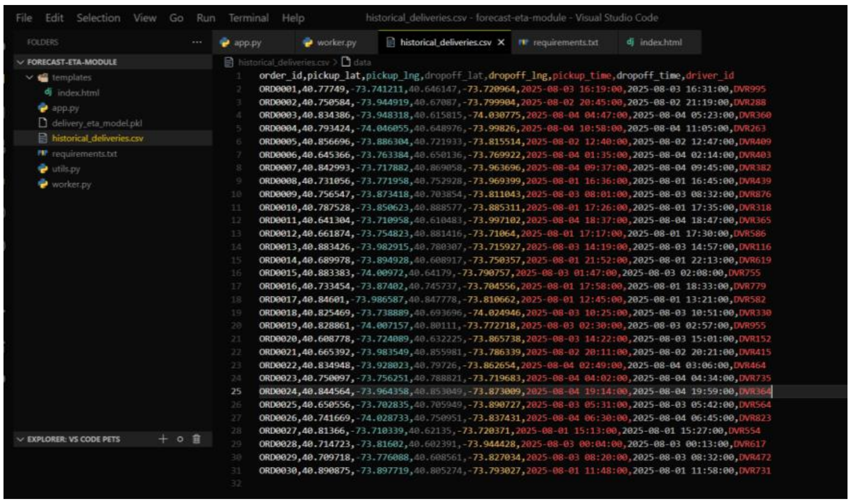
```
# 📦 Smart Warehouse & Inventory Management System

This project simulates a smart warehouse inventory system using barcode scanning, data visualization, and analytics. It's built as part of a 5-week data
science internship to integrate multiple technologies in a full-stack environment.

---

## 🛠 Tech Stack

| Component         | Technology         |
|-----------------|-------------------|
| Backend API       | Laravel (PHP)      |
| Microservice      | Flask (Python)     |
| Database          | PostgreSQL         |
| Frontend (optional) | Next.js + Chart.js |
| Analytics         | Metabase           |
| Dataset           | Mock / Kaggle      |

---

## 📁 Project Structure

```
inventory-management/
├── flask-scanner/          # Flask microservice
├── laravel-backend/        # Laravel API for inventory
├── nextjs-dashboard/       # (Optional) Chart.js frontend
├── data/
│   ├── inventory_data.csv  # Mock inventory dataset
│   └── insert_csv.py       # Python script to insert data
```

🚀 Setup Instructions

1. 🐘 PostgreSQL Setup

Create the `inventory` table:
```

IV. Team Task Progress Report

This is for Week 3: Forecasting Delivery Time with Proximity-Based Notifications

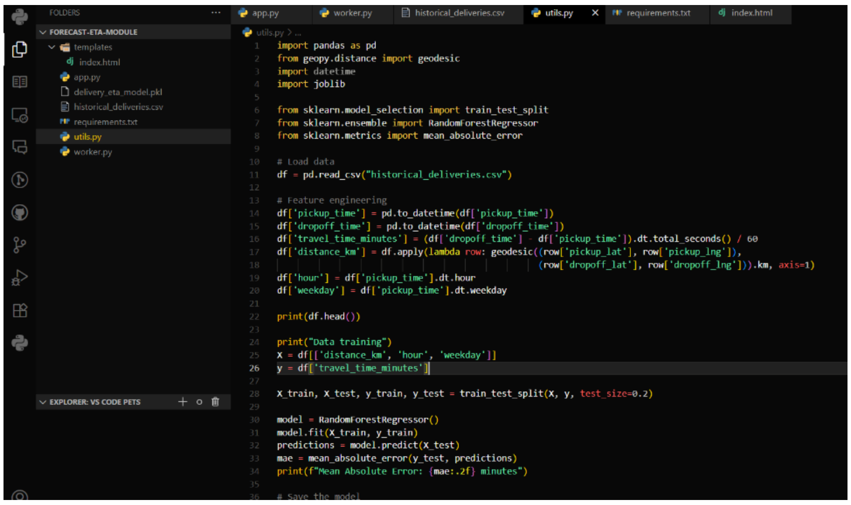
- Predicts delivery ETA based on historical data,
- Sends notifications to users based on proximity,
- Uses geolocation tracking with Haversine/geopy.

- STEP 1: Clean and prepare historical delivery datasets.



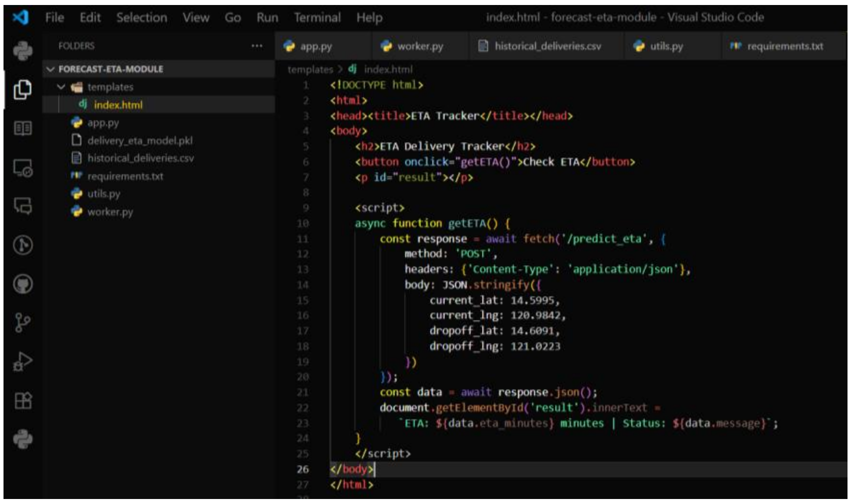
```
order_id,pickup_lat,pickup_lng,dropoff_lat,dropoff_lng,pickup_time,dropoff_time,driver_id
ORD0001,40.77749,-73.741211,40.646147,-73.720964,2025-08-03 16:19:00,2025-08-03 16:31:00,DVR995
ORD0002,40.750584,-73.944919,40.67087,-73.799904,2025-08-02 20:45:00,2025-08-02 21:19:00,DVR288
ORD0003,40.834386,-73.948318,40.615815,-74.030775,2025-08-04 04:47:00,2025-08-04 05:23:00,DVR360
ORD0004,40.793424,-74.046055,40.648976,-73.99826,2025-08-04 10:58:00,2025-08-04 11:05:00,DVR263
ORD0005,40.856696,-73.886304,40.721933,-73.815514,2025-08-02 12:40:00,2025-08-02 12:47:00,DVR409
ORD0006,40.645366,-73.763384,40.650136,-73.769922,2025-08-04 01:35:00,2025-08-04 02:14:00,DVR403
ORD0007,40.842993,-73.717882,40.869058,-73.963696,2025-08-04 09:37:00,2025-08-04 09:45:00,DVR382
ORD0008,40.731056,-73.771958,40.752928,-73.969399,2025-08-01 16:36:00,2025-08-01 16:45:00,DVR439
ORD0009,40.756547,-73.873418,40.703854,-73.811043,2025-08-03 08:01:00,2025-08-03 08:32:00,DVR876
ORD0010,40.787528,-73.850623,40.888577,-73.885311,2025-08-01 17:26:00,2025-08-01 17:35:00,DVR318
ORD0011,40.641304,-73.710958,40.610483,-73.997102,2025-08-04 18:37:00,2025-08-04 18:47:00,DVR365
ORD0012,40.661874,-73.754823,40.881416,-73.71064,2025-08-01 17:17:00,2025-08-01 17:30:00,DVR586
ORD0013,40.883426,-73.982915,40.780307,-73.715927,2025-08-03 14:19:00,2025-08-03 14:57:00,DVR116
ORD0014,40.689978,-73.894928,40.608917,-73.750357,2025-08-01 21:52:00,2025-08-01 22:13:00,DVR619
ORD0015,40.883383,-74.00972,40.64179,-73.790757,2025-08-03 01:47:00,2025-08-03 02:08:00,DVR755
ORD0016,40.733454,-73.87402,40.745737,-73.704556,2025-08-01 17:58:00,2025-08-01 18:33:00,DVR779
ORD0017,40.84601,-73.986587,40.847778,-73.810662,2025-08-01 12:45:00,2025-08-01 13:21:00,DVR582
ORD0018,40.825469,-73.738889,40.693696,-74.024946,2025-08-03 10:25:00,2025-08-03 10:51:00,DVR330
ORD0019,40.828861,-74.007157,40.80111,-73.772718,2025-08-03 02:30:00,2025-08-03 02:57:00,DVR955
ORD0020,40.608778,-73.724089,40.632225,-73.865738,2025-08-03 14:22:00,2025-08-03 15:01:00,DVR152
ORD0021,40.665392,-73.983549,40.855981,-73.786339,2025-08-02 20:11:00,2025-08-02 20:21:00,DVR415
ORD0022,40.834948,-73.928023,40.79726,-73.862654,2025-08-04 02:49:00,2025-08-04 03:06:00,DVR464
ORD0023,40.750097,-73.756251,40.788821,-73.719683,2025-08-04 04:02:00,2025-08-04 04:34:00,DVR735
ORD0024,40.844564,-73.964358,40.853049,-73.873009,2025-08-04 19:14:00,2025-08-04 19:59:00,DVR364
ORD0025,40.650556,-73.702835,40.705949,-73.890727,2025-08-03 05:31:00,2025-08-03 05:42:00,DVR564
ORD0026,40.741669,-74.028733,40.750951,-73.837431,2025-08-04 06:30:00,2025-08-04 06:45:00,DVR823
ORD0027,40.81366,-73.710339,40.62135,-73.720371,2025-08-01 15:13:00,2025-08-01 15:27:00,DVR554
ORD0028,40.714723,-73.81602,40.602391,-73.944428,2025-08-03 00:04:00,2025-08-03 00:13:00,DVR617
ORD0029,40.709718,-73.776088,40.608561,-73.827034,2025-08-03 08:20:00,2025-08-03 08:32:00,DVR472
ORD0030,40.890875,-73.897719,40.805274,-73.793027,2025-08-01 11:48:00,2025-08-01 11:58:00,DVR731
```

- STEP 2: Train an ETA prediction model using linear regression or random forest.



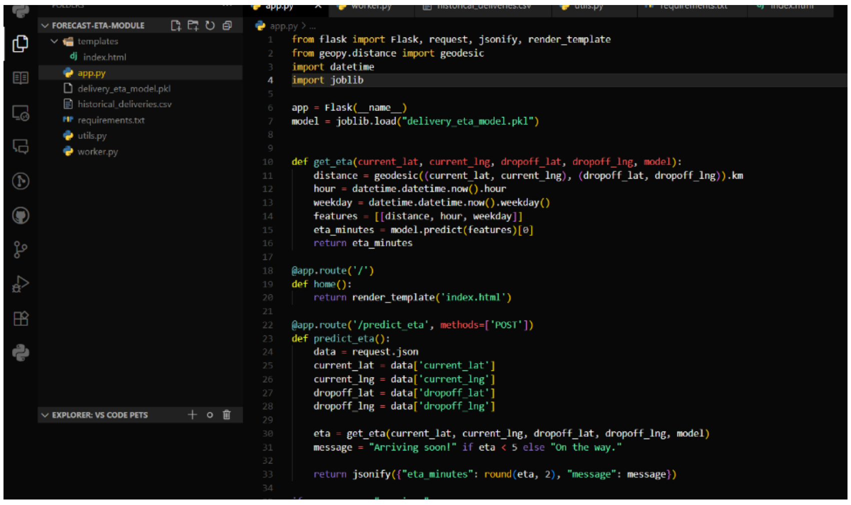
```python
import pandas as pd
from geopy.distance import geodesic
import datetime
import joblib

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Load data
df = pd.read_csv("historical_deliveries.csv")

# Feature engineering
df['pickup_time'] = pd.to_datetime(df['pickup_time'])
df['dropoff_time'] = pd.to_datetime(df['dropoff_time'])
df['travel_time_minutes'] = (df['dropoff_time'] - df['pickup_time']).dt.total_seconds() / 60
df['distance_km'] = df.apply(lambda row: geodesic((row['pickup_lat'], row['pickup_lng']),
                                                  (row['dropoff_lat'], row['dropoff_lng'])).km, axis=1)
df['hour'] = df['pickup_time'].dt.hour
df['weekday'] = df['pickup_time'].dt.weekday

print(df.head())

print("Data training")
X = df[['distance_km', 'hour', 'weekday']]
y = df['travel_time_minutes']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = RandomForestRegressor()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
mae = mean_absolute_error(y_test, predictions)
print(f"Mean Absolute Error: {mae:.2f} minutes")

# Save the model
```

- STEP 3: Calculate real-time distance using coordinates.



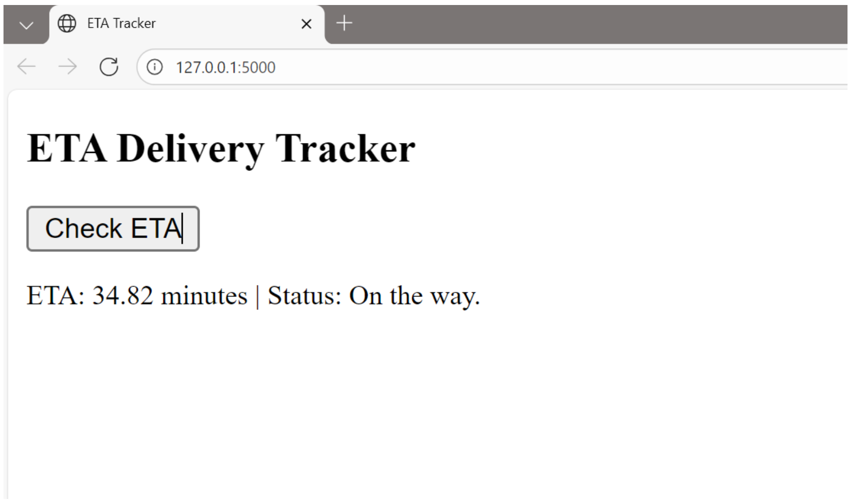
```html
<!DOCTYPE html>
<html>
<head><title>ETA Tracker</title></head>
<body>
    <h2>ETA Delivery Tracker</h2>
    <button onclick="getETA()">Check ETA</button>
    <p id="result"></p>

    <script>
    async function getETA() {
        const response = await fetch('/predict_eta', {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify({
                current_lat: 14.5995,
                current_lng: 120.9842,
                dropoff_lat: 14.6091,
                dropoff_lng: 121.0223
            })
        });
        const data = await response.json();
        document.getElementById('result').innerText =
          `ETA: ${data.eta_minutes} minutes | Status: ${data.message}`;
    }
    </script>
</body>
</html>
```

- STEP 4: Create a Flask API to return ETA predictions and proximity notifications.



```python
from flask import Flask, request, jsonify, render_template
from geopy.distance import geodesic
import datetime
import joblib

app = Flask(__name__)
model = joblib.load("delivery_eta_model.pkl")


def get_eta(current_lat, current_lng, dropoff_lat, dropoff_lng, model):
    distance = geodesic((current_lat, current_lng), (dropoff_lat, dropoff_lng)).km
    hour = datetime.datetime.now().hour
    weekday = datetime.datetime.now().weekday()
    features = [[distance, hour, weekday]]
    eta_minutes = model.predict(features)[0]
    return eta_minutes

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict_eta', methods=['POST'])
def predict_eta():
    data = request.json
    current_lat = data['current_lat']
    current_lng = data['current_lng']
    dropoff_lat = data['dropoff_lat']
    dropoff_lng = data['dropoff_lng']

    eta = get_eta(current_lat, current_lng, dropoff_lat, dropoff_lng, model)
    message = "Arriving soon!" if eta < 5 else "On the way."

    return jsonify({"eta_minutes": round(eta, 2), "message": message})
```

- STEP 5: Display live delivery updates and ETA on a basic frontend.



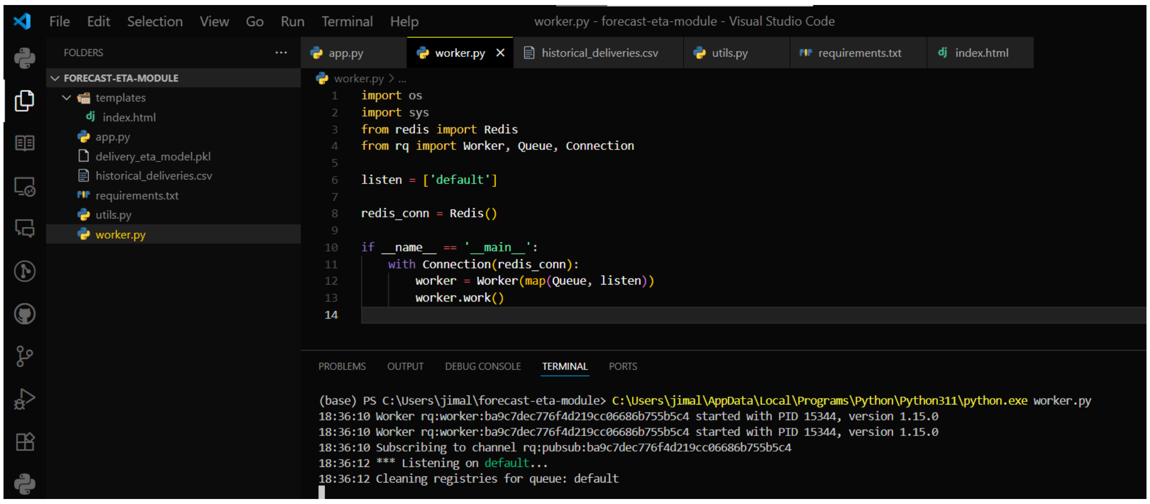
ETA Delivery Tracker

Check ETA

ETA: 34.82 minutes | Status: On the way.

BONUS: Extensions

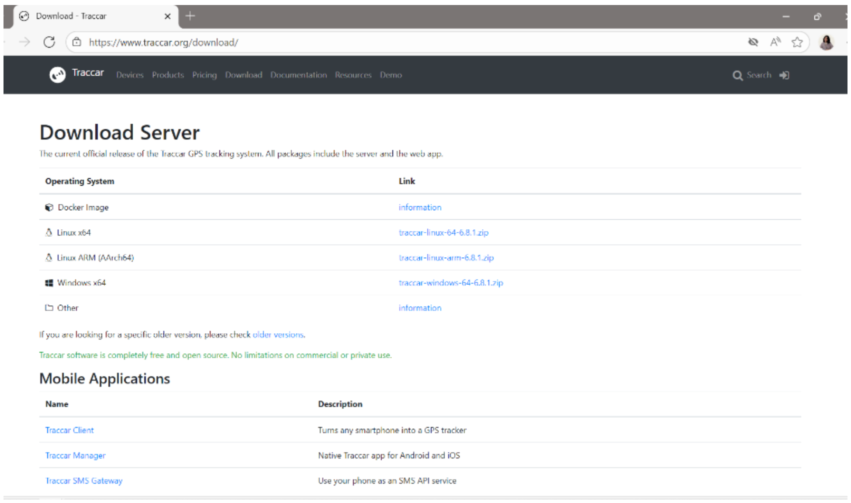
- Redis queue to handle GPS data polling



```python
import os
import sys
from redis import Redis
from rq import Worker, Queue, Connection

listen = ['default']

redis_conn = Redis()

if __name__ == '__main__':
    with Connection(redis_conn):
        worker = Worker(map(Queue, listen))
        worker.work()
```

```
(base) PS C:\Users\jimal\forecast-eta-module> C:\Users\jimal\AppData\Local\Programs\Python\Python311\python.exe worker.py
18:36:10 Worker rq:worker:ba9c7dec776f4d219cc06686b755b5c4 started with PID 15344, version 1.15.0
18:36:10 Worker rq:worker:ba9c7dec776f4d219cc06686b755b5c4 started with PID 15344, version 1.15.0
18:36:10 Subscribing to channel rq:pubsub:ba9c7dec776f4d219cc06686b755b5c4
18:36:12 *** Listening on default...
18:36:12 Cleaning registries for queue: default
```

- Traccar or Postrack integration to replace manual GPS
  - Install Traccar Server UI



Download Server

The current official release of the Traccar GPS tracking system. All packages include the server and the web app.

Operating System	Link
Docker Image	information
Linux x64	traccar-linux-64-6.8.1.zip
Linux ARM (AArch64)	traccar-linux-arm-6.8.1.zip
Windows x64	traccar-windows-64-6.8.1.zip
Other	information

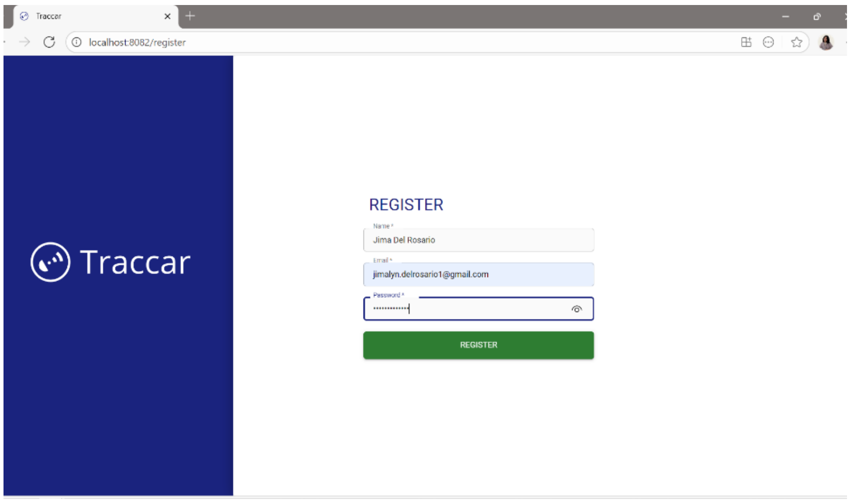
If you are looking for a specific older version, please check older versions.

Traccar software is completely free and open source. No limitations on commercial or private use.

Mobile Applications

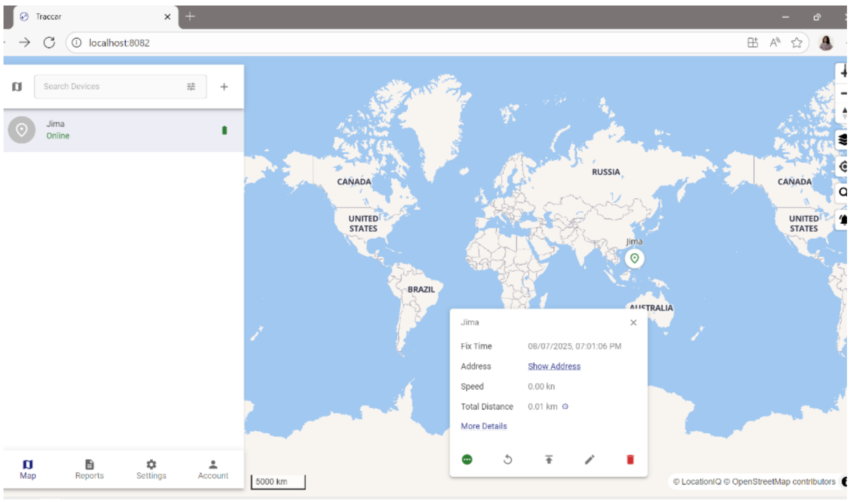
Name	Description
Traccar Client	Turns any smartphone into a GPS tracker
Traccar Manager	Native Traccar app for Android and iOS
Traccar SMS Gateway	Use your phone as an SMS API service

  - Download Traccar Client on phone device



Name *
Jima Del Rosario
Email *
jimalyn.delrosario1@gmail.com
Password *
••••••••••••

  - Set up connection: Device ID and IPv4 Address

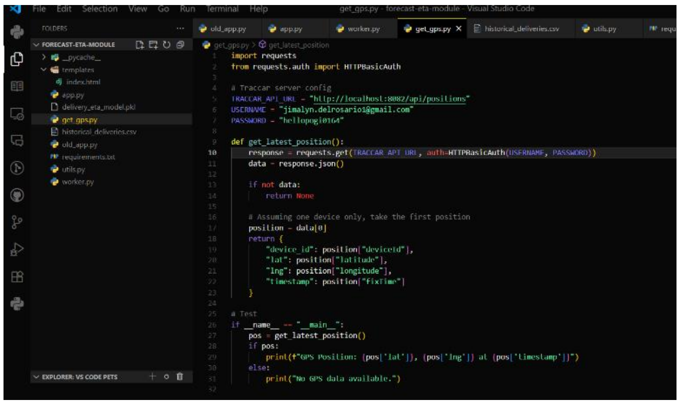


Jima

Fix Time 08/07/2025, 07:01:06 PM
Address Show Address
Speed 0.00 kn
Total Distance 0.01 km
More Details

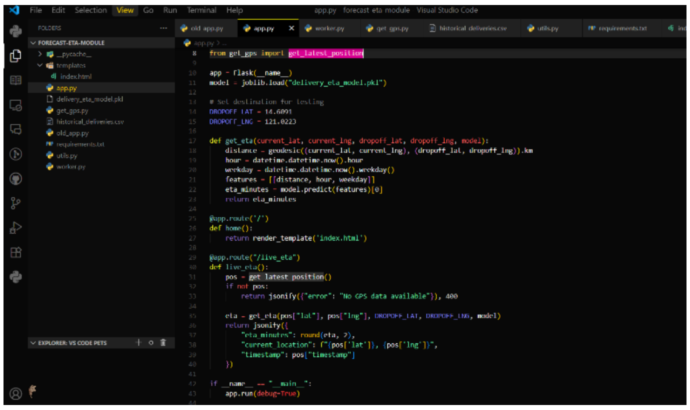
  - Temporarily disable Windows Defender

- Created get_gps.py to get live location of phone device



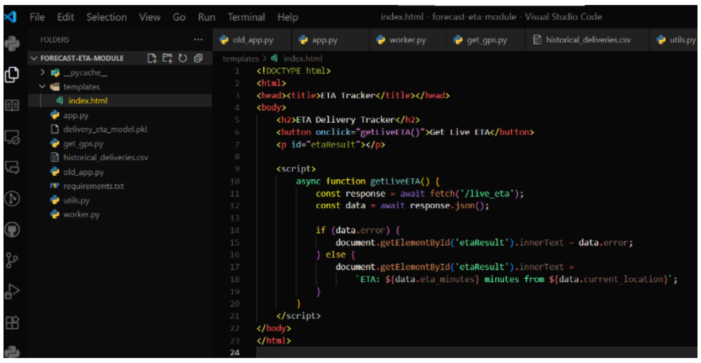
```python
import requests
from requests.auth import HTTPBasicAuth

# Traccar server config
TRACCAR_API_URL = "http://localhost:8082/api/positions"
USERNAME = "jimalyn.delrosario1@gmail.com"
PASSWORD = "hellopogi0164"

def get_latest_position():
    response = requests.get(TRACCAR_API_URL, auth=HTTPBasicAuth(USERNAME, PASSWORD))
    data = response.json()

    if not data:
        return None

    # Assuming one device only, take the first position
    position = data[0]
    return {
        "device_id": position["deviceId"],
        "lat": position["latitude"],
        "lng": position["longitude"],
        "timestamp": position["fixTime"]
    }

# Test
if __name__ == "__main__":
    pos = get_latest_position()
    if pos:
        print(f"GPS Position: {pos['lat']}, {pos['lng']} at {pos['timestamp']}")
    else:
        print("No GPS data available.")
```

- Created new app.py with /live_eta routing that imports get_latest_position function



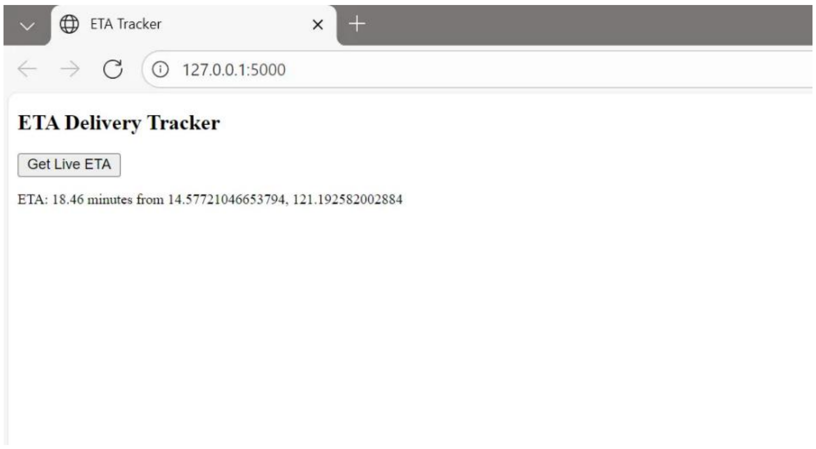
```python
from get_gps import get_latest_position

app = Flask(__name__)
model = joblib.load("delivery_eta_model.pkl")

# Set destination for testing
DROPOFF_LAT = 14.6091
DROPOFF_LNG = 121.0223

def get_eta(current_lat, current_lng, dropoff_lat, dropoff_lng, model):
    distance = geodesic((current_lat, current_lng), (dropoff_lat, dropoff_lng)).km
    hour = datetime.datetime.now().hour
    weekday = datetime.datetime.now().weekday()
    features = [[distance, hour, weekday]]
    eta_minutes = model.predict(features)[0]
    return eta_minutes

@app.route('/')
def home():
    return render_template('index.html')

@app.route("/live_eta")
def live_eta():
    pos = get_latest_position()
    if not pos:
        return jsonify({"error": "No GPS data available"}), 400

    eta = get_eta(pos["lat"], pos["lng"], DROPOFF_LAT, DROPOFF_LNG, model)
    return jsonify({
        "eta_minutes": round(eta, 2),
        "current_location": f"{pos['lat']}, {pos['lng']}",
        "timestamp": pos["timestamp"]
    })

if __name__ == "__main__":
    app.run(debug=True)
```

- Created new index.html to fetch from the /live_eta route



```html
<!DOCTYPE html>
<html>
<head><title>ETA Tracker</title></head>
<body>
    <h2>ETA Delivery Tracker</h2>
    <button onclick="getLiveETA()">Get Live ETA</button>
    <p id="etaResult"></p>

    <script>
        async function getLiveETA() {
            const response = await fetch('/live_eta');
            const data = await response.json();

            if (data.error) {
                document.getElementById('etaResult').innerText = data.error;
            } else {
                document.getElementById('etaResult').innerText =
                    `ETA: ${data.eta_minutes} minutes from ${data.current_location}`;
            }
        }
    </script>
</body>
</html>
```

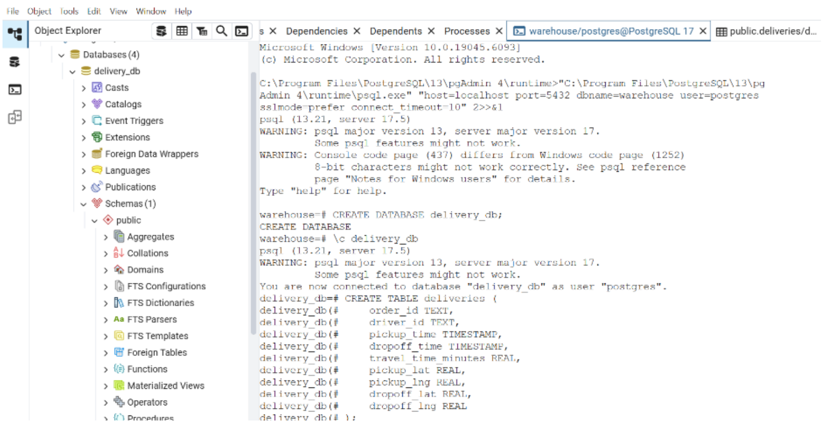
- Run app.py to predict based on live data



ETA Delivery Tracker

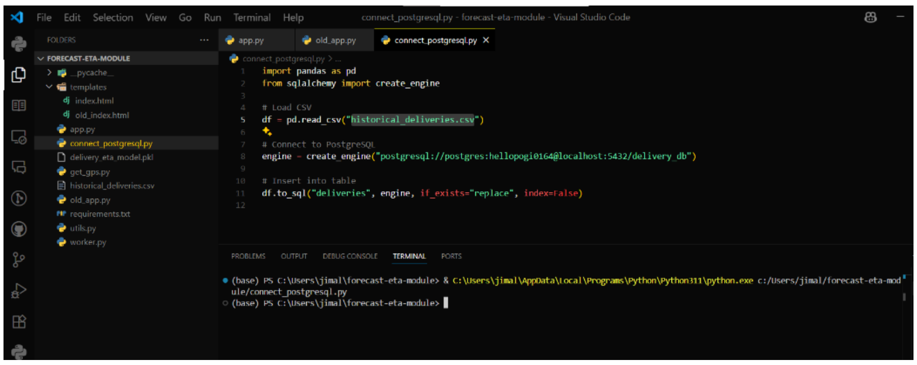
ETA: 18.46 minutes from 14.57721046653794, 121.192582002884

- Metadata dashboard for average delivery time per driver or location
  - Create delivery_db in pgAdmin



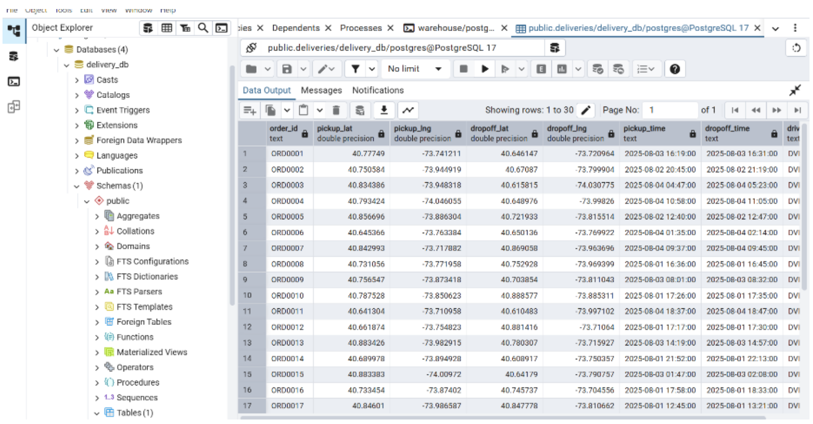
```
Microsoft Windows [Version 10.0.19045.6093]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\PostgreSQL\13\pgAdmin 4\runtime>"C:\Program Files\PostgreSQL\13\pg
Admin 4\runtime\psql.exe" "host=localhost port=5432 dbname=warehouse user=postgres
sslmode=prefer connect_timeout=10" 2>>&1
psql (13.21, server 17.5)
WARNING: psql major version 13, server major version 17.
         Some psql features might not work.
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

warehouse=# CREATE DATABASE delivery_db;
CREATE DATABASE
warehouse=# \c delivery_db
psql (13.21, server 17.5)
WARNING: psql major version 13, server major version 17.
         Some psql features might not work.
You are now connected to database "delivery_db" as user "postgres".
delivery_db=# CREATE TABLE deliveries (
delivery_db(#     order_id TEXT,
delivery_db(#     driver_id TEXT,
delivery_db(#     pickup_time TIMESTAMP,
delivery_db(#     dropoff_time TIMESTAMP,
delivery_db(#     travel_time_minutes REAL,
delivery_db(#     pickup_lat REAL,
delivery_db(#     pickup_lng REAL,
delivery_db(#     dropoff_lat REAL,
delivery_db(#     dropoff_lng REAL
delivery_db(# );
```

  - Import historical_deliveries.csv



```python
import pandas as pd
from sqlalchemy import create_engine

# Load CSV
df = pd.read_csv("historical_deliveries.csv")

# Connect to PostgreSQL
engine = create_engine("postgresql://postgres:hellopogi0164@localhost:5432/delivery_db")

# Insert into table
df.to_sql("deliveries", engine, if_exists="replace", index=False)
```

```
(base) PS C:\Users\jimal\forecast-eta-module> & C:\Users\jimal\AppData\Local\Programs\Python\Python311\python.exe c:/Users/jimal/forecast-eta-mod
ule/connect_postgresql.py
(base) PS C:\Users\jimal\forecast-eta-module>
```

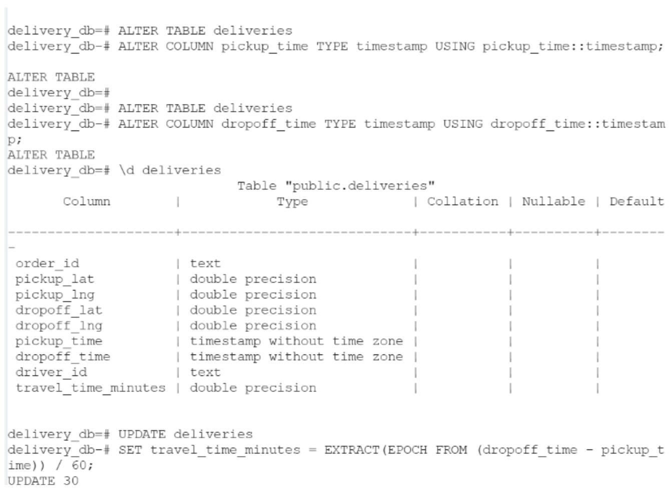


public.deliveries/delivery_db/postgres@PostgreSQL 17

Showing rows: 1 to 30 Page No: 1 of 1

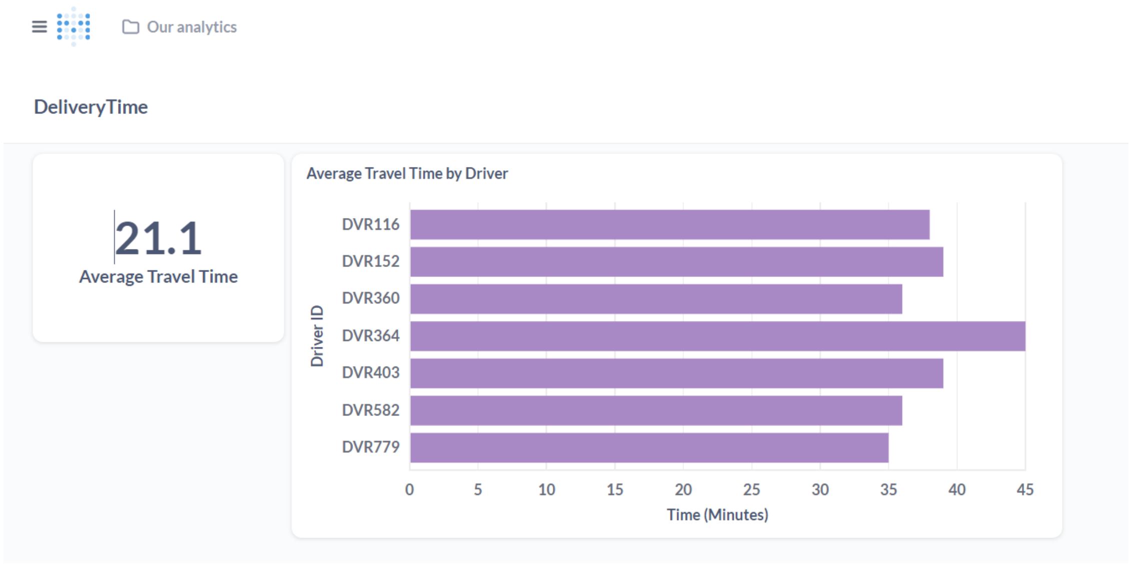
	order_id text	pickup_lat double precision	pickup_lng double precision	dropoff_lat double precision	dropoff_lng double precision	pickup_time text	dropoff_time text	driv text
1	ORD0001	40.77749	-73.741211	40.646147	-73.720964	2025-08-03 16:19:00	2025-08-03 16:31:00	DVI
2	ORD0002	40.750584	-73.944919	40.67087	-73.799904	2025-08-02 20:45:00	2025-08-02 21:19:00	DVI
3	ORD0003	40.834386	-73.948318	40.615815	-74.030775	2025-08-04 04:47:00	2025-08-04 05:23:00	DVI
4	ORD0004	40.793424	-74.046055	40.648976	-73.99826	2025-08-04 10:58:00	2025-08-04 11:05:00	DVI
5	ORD0005	40.856696	-73.886304	40.721933	-73.815514	2025-08-02 12:40:00	2025-08-02 12:47:00	DVI
6	ORD0006	40.645366	-73.763384	40.650136	-73.769922	2025-08-04 01:35:00	2025-08-04 02:14:00	DVI
7	ORD0007	40.842993	-73.717882	40.869058	-73.963696	2025-08-04 09:37:00	2025-08-04 09:45:00	DVI
8	ORD0008	40.731056	-73.771958	40.752928	-73.969399	2025-08-01 16:36:00	2025-08-01 16:45:00	DVI
9	ORD0009	40.756547	-73.873418	40.703854	-73.811043	2025-08-03 08:01:00	2025-08-03 08:32:00	DVI
10	ORD0010	40.787528	-73.850623	40.888577	-73.885311	2025-08-01 17:26:00	2025-08-01 17:35:00	DVI
11	ORD0011	40.641304	-73.710958	40.610483	-73.997102	2025-08-04 18:37:00	2025-08-04 18:47:00	DVI
12	ORD0012	40.661874	-73.754823	40.881416	-73.71064	2025-08-01 17:17:00	2025-08-01 17:30:00	DVI
13	ORD0013	40.883426	-73.982915	40.780307	-73.715927	2025-08-03 14:19:00	2025-08-03 14:57:00	DVI
14	ORD0014	40.689978	-73.894928	40.608917	-73.750357	2025-08-01 21:52:00	2025-08-01 22:13:00	DVI
15	ORD0015	40.883383	-74.00972	40.64179	-73.790757	2025-08-03 01:47:00	2025-08-03 02:08:00	DVI
16	ORD0016	40.733454	-73.87402	40.745737	-73.704556	2025-08-01 17:58:00	2025-08-01 18:33:00	DVI
17	ORD0017	40.84601	-73.986587	40.847778	-73.810662	2025-08-01 12:45:00	2025-08-01 13:21:00	DVI

  - Created new column travel_time_minutes
  - Converted time columns in time date format and updated content of travel_time_minutes column



```
delivery_db=# ALTER TABLE deliveries
delivery_db-# ALTER COLUMN pickup_time TYPE timestamp USING pickup_time::timestamp;

ALTER TABLE
delivery_db=#
delivery_db=# ALTER TABLE deliveries
delivery_db-# ALTER COLUMN dropoff_time TYPE timestamp USING dropoff_time::timestam
p;
ALTER TABLE
delivery_db=# \d deliveries
                              Table "public.deliveries"
       Column        |            Type             | Collation | Nullable | Default

---------------------+-----------------------------+-----------+----------+--------
-
 order_id            | text                        |           |          |
 pickup_lat          | double precision            |           |          |
 pickup_lng          | double precision            |           |          |
 dropoff_lat         | double precision            |           |          |
 dropoff_lng         | double precision            |           |          |
 pickup_time         | timestamp without time zone |           |          |
 dropoff_time        | timestamp without time zone |           |          |
 driver_id           | text                        |           |          |
 travel_time_minutes | double precision            |           |          |


delivery_db=# UPDATE deliveries
delivery_db-# SET travel_time_minutes = EXTRACT(EPOCH FROM (dropoff_time - pickup_t
ime)) / 60;
UPDATE 30
```

- Connected to Metabase and created dashboard


Our analytics
DeliveryTime
21.1
Average Travel Time
Average Travel Time by Driver
DVR116
DVR152
DVR360
DVR364
DVR403
DVR582
DVR779
Driver ID
0
5
10
15
20
25
30
35
40
45
Time (Minutes)