

# Documentation: Automated Vehicle Tracking System

## 1. Introduction & Project Overview

This document outlines an automated workflow built in n8n for tracking vehicles in near real-time. The system is designed to take a list of vehicle license plates from a spreadsheet, query an external API for each vehicle's current geographical coordinates (longitude and latitude), and then write this location data back to another spreadsheet.

### Why Automate This with n8n?

Manually tracking a fleet of vehicles is a time-consuming, error-prone, and costly process. It involves:

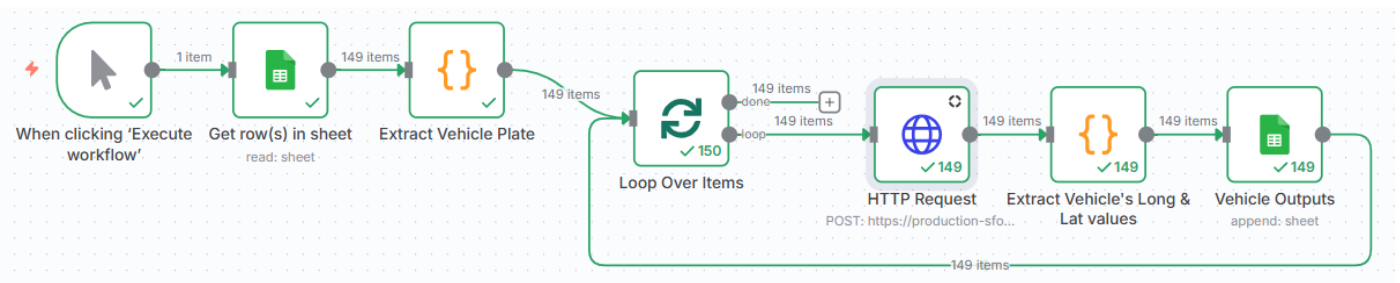
- **Repetitive Tasks:** Copying/pasting license plates, calling APIs one by one, and updating spreadsheets.
- **Human Error:** Misreading plate numbers, entering data into the wrong row, or missing updates.
- **High Labour Cost:** Requiring an employee to dedicate hours to a task that a machine can do in minutes.
- **Lack of Real-Time Data:** Manual processes cannot keep up with the frequency required for true real-time tracking.

n8n provides a solution by:

- **Saving Time:** The entire process for over 100 vehicles is executed automatically with a single click ("Execute workflow"), completing in a fraction of the time it would take a human.
- **Reducing Cost:** It eliminates hours of manual labour, allowing staff to focus on higher-value tasks like analysing the data or managing exceptions.
- **Improving Accuracy:** Automation ensures that every vehicle plate is processed correctly, and the resulting data is appended to the sheet without errors.
- **Enabling Scalability:** This workflow can process 10 or 10,000 vehicles with minimal additional effort, making it future proof as the fleet grows.

## 2. Workflow Implementation: Step-by-Step Explanation

The workflow consists of several nodes connected in a logical sequence. Here is a breakdown of each step, its purpose, configuration, and output.



### Step 1: Manual Trigger - "Execute workflow"

- **Purpose:** This is the starting point of the workflow. It allows a user to manually initiate the entire automation process. In a production environment, this could be replaced with a **Schedule Trigger** node to run automatically at set intervals (e.g., every 15 minutes).
- **Configuration:** "Manual Trigger" node with a descriptive name.
- **Output:** A signal to start the workflow.

### Step 2: Get Row(s) in Sheet - "sheet read"

- **Purpose:** This node connects to a spreadsheet (likely Google Sheets or Airtable) and reads all the data from a specified sheet. This data contains the list of vehicle license plates that need to be tracked.
- **Configuration:**
  - **Credentials:** Connection details for the spreadsheet service.
  - **Sheet ID:** The identifier for the specific spreadsheet.
  - **Range:** The specific cells or sheet name to read from (e.g., Sheet1!A:Z).
- **Output:** An array of JSON objects, where each object represents a row from the sheet. For example:

```
json  
[
```

```
{ "rowNumber": 1, "vehiclePlate": "ABC123", "status": "" },
{ "rowNumber": 2, "vehiclePlate": "XYZ789", "status": "" },
... // 149 items total
]
```

### Step 3: Extract Vehicle Plate

- **Purpose:** This step is crucial for preparing the data for the API call. It extracts the specific license plate value from each row of data received from the previous sheet. This ensures a clean, isolated value is passed to the HTTP request.
- **Configuration:** This is likely a **Set** node or a **Code** node. It takes the input data (e.g., `{{ $json.vehiclePlate }}`) and sets it as a new field in the workflow's data stream.
- **Output:** The data is now formatted with the license plate as a prominent, easily accessible value for the next node.

### Step 4: Loop Over Items

- **Purpose:** This instructs n8n to take the list of 149 vehicles and process them **one by one**. This is necessary because the HTTP request node will be called for each individual vehicle plate, and the results must be handled separately.
- **Configuration:** This is typically the built-in looping functionality of n8n, often activated by toggling "Execute once for each item" in the HTTP Request node itself, making this a conceptual step rather than a separate node.
- **Output:** The subsequent nodes (HTTP Request, Extract Vehicle's Long & Lat values) will run 149 separate times.

### Step 5: HTTP Request - POST: [https://production-sfo.browserless.io/content?token=API\\_Key](https://production-sfo.browserless.io/content?token=API_Key)

- **Purpose:** This is the core of the workflow. For each vehicle plate, it sends a secure (HTTPS) POST request to the vehicle tracking service's production API endpoint. This API is the service that has the real-time GPS data for each vehicle.
- **Configuration:**
  - **URL:** The endpoint provided by the tracking service (<https://production-sfo...>).
  - **Method:** **POST**
  - **Authentication:** Usually added in the "Headers" or "Authentication" tab (e.g., Bearer Token or API Key).
  - **Body:** Contains the mapped data from the previous step (e.g., the license plate number).
- **Output:** For each successful request, the node receives a response from the API. This response is a JSON object containing the vehicle's location data. It might look like this:

```
json
{
  "vehicle": "ABC123",
  "status": "moving",
  "location": {
    "latitude": 40.7128,
    "longitude": -74.0060
  },
  "timestamp": "2023-10-27T10:30:00Z"
}
```

### Step 6: Extract Vehicle's Long & Lat Values

- **Purpose:** This node parses the JSON response from the API to extract the precise longitude and latitude values. It isolates this crucial data so it can be cleanly written back to the spreadsheet.
- **Configuration:** This is usually a **Function** node or multiple **Set** nodes. It uses JavaScript code or expressions to access the nested values:
  - `{{ $json.location.latitude }}`
  - `{{ $json.location.longitude }}`
- **Output:** The workflow data now contains the original sheet data *plus* the new `latitude` and `longitude` fields.

### Step 7: Vehicle Outputs Append - "sheet"

- **Purpose:** This final node takes the processed data for each vehicle—including the newly fetched coordinates—and appends it back to the new spreadsheet. It updates the system of record with the latest tracking information.

- **Configuration:**
  - **Operation:** Append (adds a new row) or Update (if a unique key like the plate number is used to find the correct row).
  - **Columns:** Mapped to specify which field from the workflow data goes into which column in the sheet (e.g., Vehicle Plate -> A, Latitude -> B, Longitude -> C).
- **Output:** The spreadsheet is updated with 149 new rows of geolocation data. The "149 items" note indicate that all the 149 requests returned valid location data.

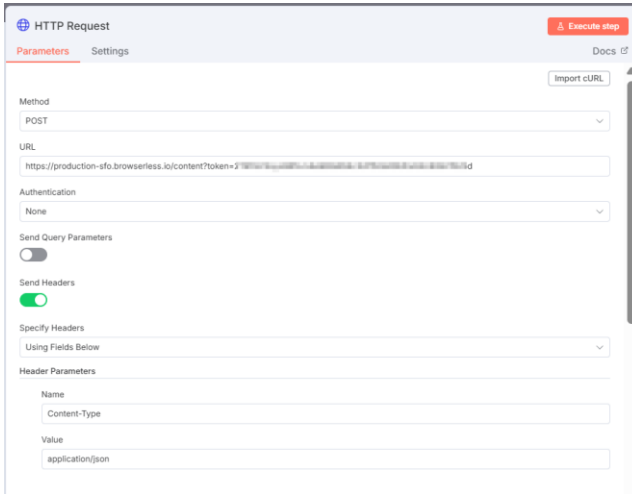
### 3. Summary of Output

Upon clicking "Execute workflow", the system:

1. Reads **149** vehicle license plates from a spreadsheet.
2. Queries a production tracking API for each vehicle's location.
3. Successfully retrieves and processes location data for **149** vehicles.
4. Appends the longitude and latitude for these **149** vehicles to a specified output sheet, creating a clear and actionable log of vehicle positions.
5. This automated flow transforms a previously manual, multi-hour task into a reliable, one-click operation that takes minutes, providing immediate visibility into fleet operations.

### Details

We are inputting the vehicle number in the textbox in this website <https://www.cocreate.asia/rpa-test> and press the Submit button. Because there is no API endpoint for the above website, we will use a Headless Browser Service from Browserless.io which we need an API Key.



URL: [https://production-sfo.browserless.io/content?token=API\\_Key](https://production-sfo.browserless.io/content?token=API_Key)

Body Content Type: JSON

Specify Body: Using JSON

```

{
  "url": "https://www.cocreate.asia/rpa-test",
  "gotoOptions": {
    "waitUntil": "networkidle2"
  },
  "addScriptTag": [
    {
      "content": "document.getElementsByName('name')[0].value='{{ $json.plate }}';document.forms.myform.querySelector('input[type=submit']").click();"
    }
  ],
  "waitForTimeout": 5000
}

```

Inside the Body Content Type JSON, we have:

```

{
  "url": "https://www.cocreate.asia/rpa-test",
  "gotoOptions": {
    "waitUntil": "networkidle2"
  },
  "addScriptTag": [
    {
      "content": "document.getElementsByName('name')[0].value='{{ $json.plate }}';document.forms.myform.querySelector('input[type=submit']").click();"
    }
  ],
  "waitForTimeout": 5000
}

```