Fuel Price Tracker - Phase 2: Live Data Integration (Beginner-Pro Guide)

1. Introduction

Welcome to Phase 2 of your Fuel Price Tracker project! In this phase, you?ll learn how to gather real-world fuel price data from Israeli fuel company websites and store it in your application?s database. This guide is designed for beginners ? no prior scraping or backend experience needed. Each step is broken down in detail, and by the end, your app will reflect **real, live fuel prices** from around the country.

2. Overview of What You'll Build

You will build:

- A script that visits fuel websites and collects fuel prices (web scraper)
- A data cleaner that ensures consistency (e.g., rounding prices, formatting names)
- A tool to enrich the data with GPS coordinates (so you can plot it on a map)
- A function to save that data into your database (MongoDB)
- A scheduler that updates prices every few hours automatically

3. Tools You'll Need (Install First)

Make sure you have Node.js installed. Then run:

`npm install puppeteer mongoose dotenv axios node-cron`

- **puppeteer**: controls a browser to scrape data from real websites
- **mongoose**: lets you interact with your MongoDB database easily
- **dotenv**: securely stores database connection strings
- **axios**: lets you send requests to geolocation APIs
- **node-cron**: runs code on a schedule (like an alarm clock)

4. Step-by-Step: Creating the Scraper

1. Create a file called `scrapeYellow.js`
2. Paste this structure:

```js
const puppeteer = require('puppeteer');
(async () => {
```

```
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://yellow.co.il/station-prices');
  const data = await page.evaluate(() => {
    const rows = Array.from(document.querySelectorAll('.station-row'));
    return rows.map(row => ({
      name: row.querySelector('.station-name')?.innerText.trim(),
      city: row.querySelector('.station-city')?.innerText.trim(),
      price: parseFloat(row.querySelector('.station-price')?.innerText.replace('?', ''))
    }));
  });
  console.log(data);
  await browser.close();
})();
```

5. Step-by-Step: Normalizing and Enriching the Data

- Add `station_id` by combining name+city, and replacing spaces with underscores
- Use Axios to get coordinates from the Nominatim API:

```js
const axios = require('axios');
async function geocode(city) {
  const res = await axios.get(`https://nominatim.openstreetmap.org/search`, {
    params: { q: city + ', Israel', format: 'json' }
  });
  const loc = res.data[0];
  return { lat: parseFloat(loc.lat), lng: parseFloat(loc.lon) };
}
```

Fuel Price Tracker - Phase 2: Live Data Integration (Beginner-Pro Guide)

6. Step-by-Step: Connecting to MongoDB and Saving

- Create a file `.env` and put your Mongo URI like:

`MONGO_URI=mongodb+srv://username:password@cluster.mongodb.net/dbname`

- Create `models/Station.js`:

```js
const mongoose = require('mongoose');
const stationSchema = new mongoose.Schema({
  station_id: String,
  name: String,
  company: String,
  city: String,
  price_per_liter: Number,
  coordinates: { lat: Number, lng: Number },
  last_updated: Date
});
module.exports = mongoose.model('Station', stationSchema);
```

7. Saving Your Data

In your scraper file, after you enrich the data with geolocation:

```js
require('dotenv').config();
const mongoose = require('mongoose');
const Station = require('./models/Station');
mongoose.connect(process.env.MONGO_URI)
.then(() => console.log('MongoDB Connected'));
await Station.deleteMany({ company: 'Yellow' }); // optional cleanup
await Station.insertMany(enrichedData);
```

Fuel Price Tracker - Phase 2: Live Data Integration (Beginner-Pro Guide)

8. Automating with node-cron

- Install node-cron: `npm install node-cron`
- Create a scheduler.js file:

```js
const cron = require('node-cron');
const runScraper = require('./scrapeYellow');
cron.schedule('0 */6 * * *', () => { runScraper(); });
```

9. Extra Tips for Beginners

- Use `console.log()` to debug each part: scraping, enriching, saving
- Run each part separately before combining
- Don?t panic if a site structure changes ? that?s part of scraping
- Build it step by step ? not all at once. Test constantly.

10. Final Words

You?ve just built a real-world, real-time fuel data engine. Once it works, integrate it with your frontend. When users open the site, they'll see **live, automatically refreshed** fuel prices across Israel ? just like magic.