The **Gas Tracker** fetches data from a **public Ethereum API provider** like [Etherscan](https://etherscan.io/) or [Etherchain](https://www.etherchain.org/). These services run their own Ethereum nodes and provide **real-time gas price estimates** based on pending transactions in the mempool (the queue of unconfirmed transactions).

**import React, { useEffect, useState } from 'react';**

➡️ This line brings in React, and two special hooks:

useState to hold data (like gas prices),

useEffect to run code when the component loads.

**import axios from 'axios';**

➡️ We're using axios (a promise-based HTTP client) to make API requests to Etherscan and get gas prices.

**import './style.css';**

➡️ This loads your custom CSS file so the gas price display looks good on the screen.

**const GasTracker = () => {**

➡️ We’re defining a React component called GasTracker that will show Ethereum gas prices.

**const [gasData, setGasData] = useState(null);**

**const [error, setError] = useState(null);**

➡️ These are React states:

gasData will hold the fetched gas prices.

error will hold any error messages if the API call fails.

try {

**const response = await axios.get(`https://api.etherscan.io/v2/api?chainId=1&module=gastracker&action=gasoracle&apikey=EKYF8VJ6V6RZF4SF3RT7MJ3US2V699AB8U`);**

➡️ We're sending a GET request to Etherscan’s API to get current gas prices.

await waits until the data is received.

If successful, it stores the result in response.

**const gasInfo = response.data.result;**

➡️ We extract the actual gas prices from the API response (they are stored inside result).

**setGasData({**

**safeLow: gasInfo.SafeGasPrice,**

**standard: gasInfo.ProposeGasPrice,**

**fast: gasInfo.FastGasPrice,**

**});**

➡️ We update our state with the gas price values we care about:

Safe Low = Cheapest (but slow)

Standard = Normal speed/cost

Fast = Higher cost but faster confirmation

**catch (err) {**

**console.error('Failed to fetch gas data:', err);**

**setError('Error fetching gas data');**

**}**

➡️ If something goes wrong with the API call (e.g., network issues), we catch the error and store an error message.

**useEffect(() => {**

**fetchGasPrice();**

**const interval = setInterval(fetchGasPrice, 60000); // Refresh every 60 sec**

**return () => clearInterval(interval);**

**}, []);**

➡️ This runs the fetchGasPrice() function:

Once when the component loads Every 60 seconds after that, using setInterval()

When the component is removed, the interval is cleared to prevent memory leaks.

**<li><strong>Safe Low:</strong> {gasData.safeLow ?? 'N/A'} Gwei</li>**

**<li><strong>Standard:</strong> {gasData.standard ?? 'N/A'} Gwei</li>**

**<li><strong>Fast:</strong> {gasData.fast ?? 'N/A'} Gwei</li>**

➡️ If gas data was fetched successfully:

Show gas prices under each label.

?? 'N/A' means: if value is missing, show “N/A”.

✅ Summary

This component:

Talks to the Etherscan API.

Fetches and displays current Ethereum gas prices.

Updates automatically every 60 seconds.

Displays error messages when something goes wrong.

**Addition of dynamic chain support**

**1. 🌐 Supports a Multi-Chain World**

* Today’s Web3 users don’t just use Ethereum — they also transact on **Polygon, BNB Chain, Arbitrum, Avalanche**, etc.
* Your tracker becomes **useful to a wider audience**, not just ETH users.

**2. 🧠 Educational for Beginners**

* Many users don’t know gas works differently across chains.
* Your app helps them **learn and compare** real-time fees across blockchains.

**3. 💼 More Practical for Developers**

* DApp developers often deploy on multiple chains.
* Having a unified gas tracker helps them **monitor costs** across networks.

**4. 🎨 Better UI/UX Possibilities**

* Dropdown with **logos or themes per chain**
* Dynamic color schemes (e.g., Ethereum = blue, BNB = yellow, Polygon = purple)
* Charts or comparisons across chains

**5. 🚀 Makes Your Project Stand Out**

* Most beginner gas trackers only support Ethereum.
* Yours would look more polished, **ready for real-world use**, and **impressive on GitHub or your portfolio**.

### Objective & Purpose of the Project

The **Multichain Gas Price Tracker** is a lightweight and easy-to-use web application designed to help users monitor real-time gas prices across multiple EVM-compatible blockchains, such as Ethereum, Polygon, and Binance Smart Chain.

#### 🧩 Why this project?

Gas fees (also called transaction fees) play a crucial role in blockchain networks. Depending on how busy the network is, these fees can go up or down significantly. For developers, traders, and everyday users, it's important to know:

* When is the best time to make a transaction?
* What is the estimated cost of a transaction on different blockchains?
* How does one chain’s fee compare to another?

This tracker answers those questions **instantly and visually**, helping users make smarter and more cost-effective blockchain decisions.

#### 🔍 Key Goals:

* ✅ Fetch and display **live gas prices** from multiple networks.
* ✅ Let users **switch between chains** with a simple dropdown.
* ✅ Show different speed options: **Safe**, **Standard**, and **Fast**, with clear labels.
* ✅ Provide a clean, responsive UI that’s beginner-friendly.

#### 💡 Who is it for?

This project is perfect for:

* Blockchain learners who want to understand how gas fees work.
* Developers testing their smart contracts and dApps.
* Crypto users who want to time their transactions to save money.
* Educators teaching gas estimation and network usage.

### In Summary:

A **Multichain Gas Tracker** empowers users and developers to:

* **Compare real-time gas prices**
* **Choose the most cost-effective chain**
* **Decide whether to bridge assets**
* **Avoid high transaction fees on congested networks**