**Reverse-Engineering a Poorly Documented NHL API for ML Data Extraction**

**Context**

During the development of a machine learning model intended to analyze hockey game trends and player statistics, I initially relied on the publicly available NHL API. However, partway through the project, the original API endpoints were deprecated and replaced with a newer version — one that lacked formal documentation and examples.

The absence of structured documentation presented a roadblock to continued data ingestion and training. Rather than halting progress or pivoting to a third-party data source, I chose to reverse-engineer the API exposed through the official NHL.com website.

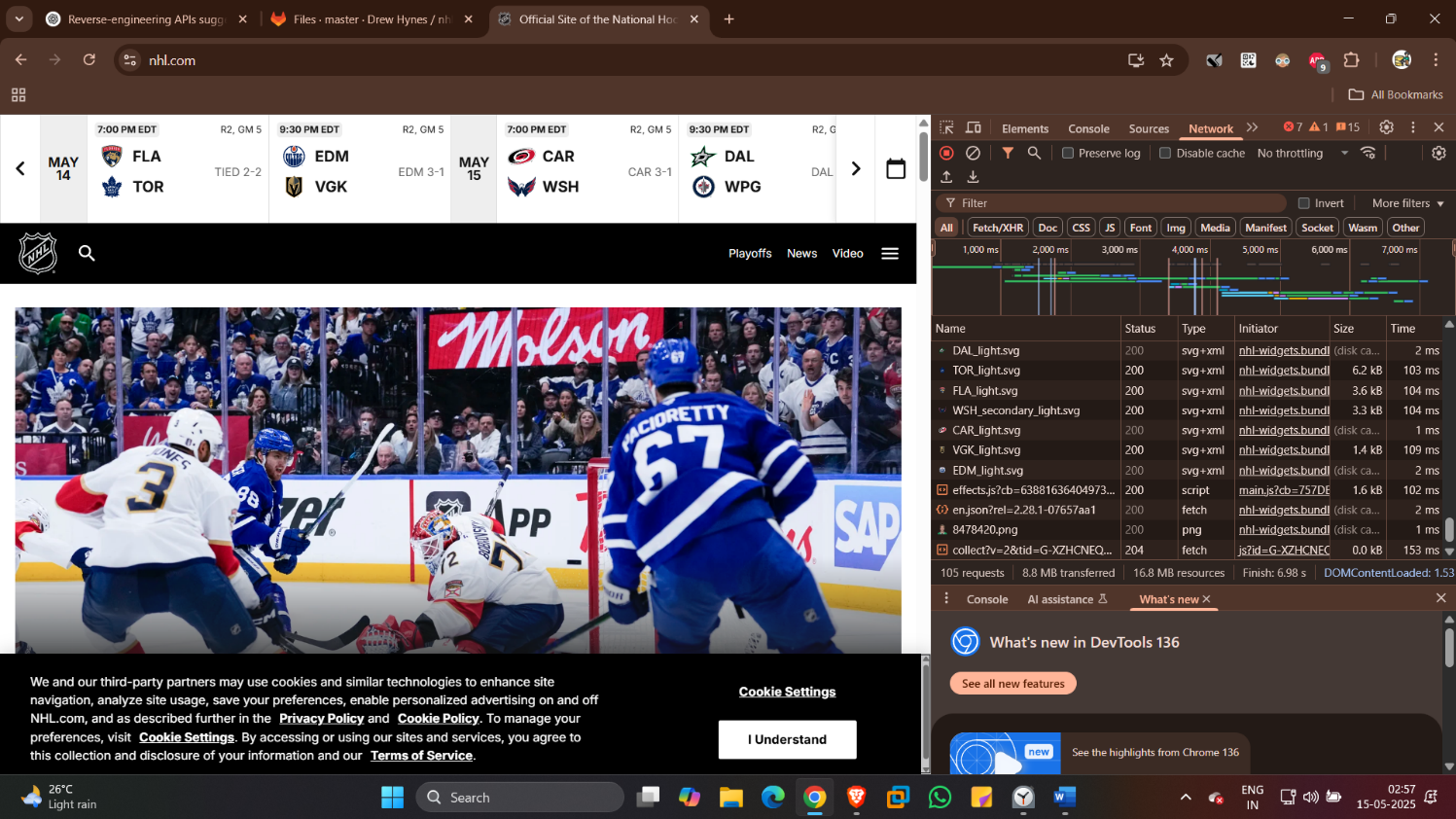
**Objective**

To retrieve structured, real-time NHL data — including schedules, team stats, and player performance — directly from the updated NHL backend, despite the lack of official API documentation.

**Methodology**

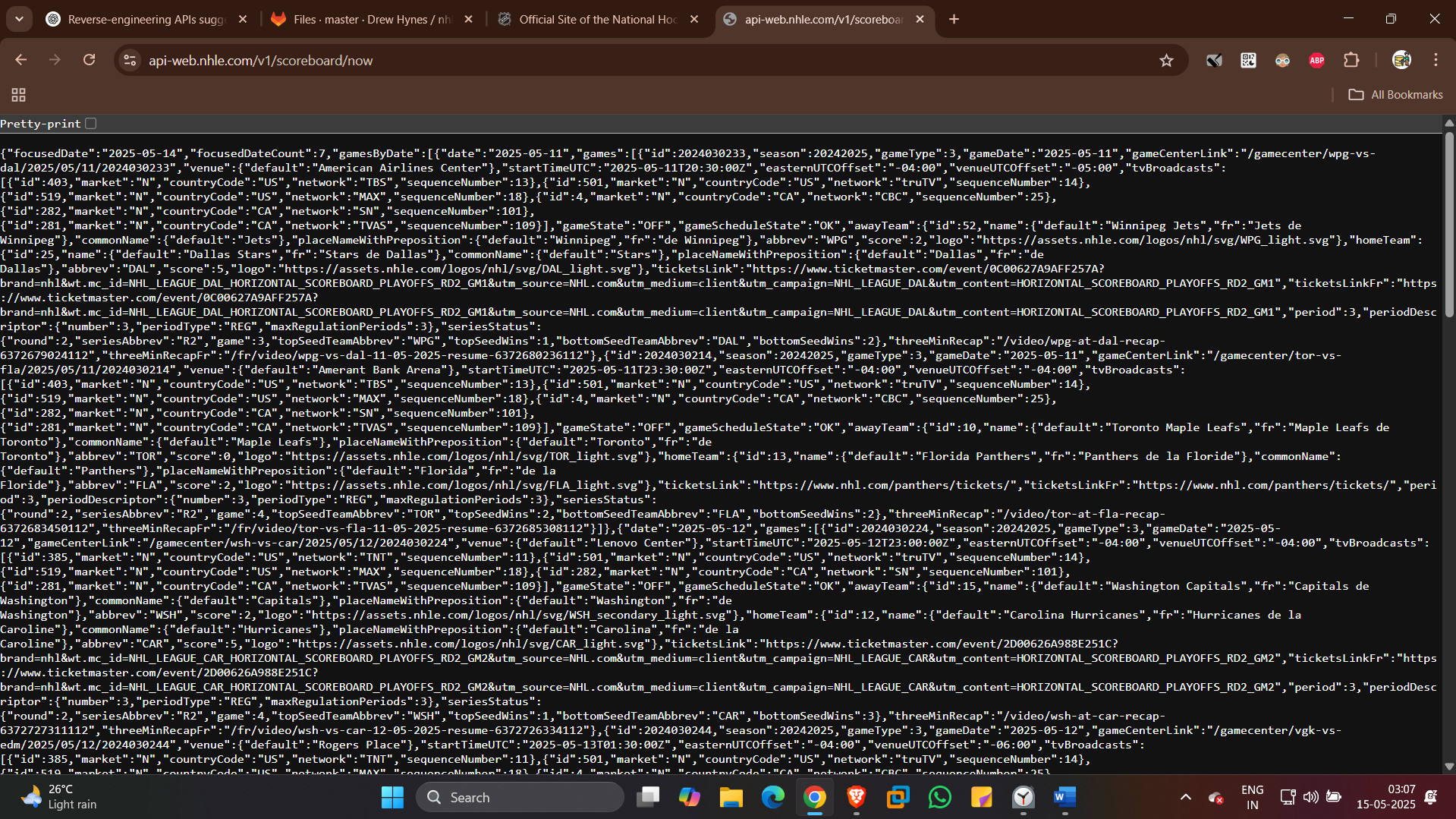
Analyzing Network Activity via Browser Dev Tools

I began by navigating to nhl.com, particularly the sections displaying live schedules and player statistics. Using Chrome DevTools, I opened the Network tab to monitor all HTTP requests being made by the site as data was dynamically loaded.



**Capturing Dynamic API URLs**

When loading the "Scores" and "Stats" sections of the site, I observed REST-like API calls to endpoints such as:

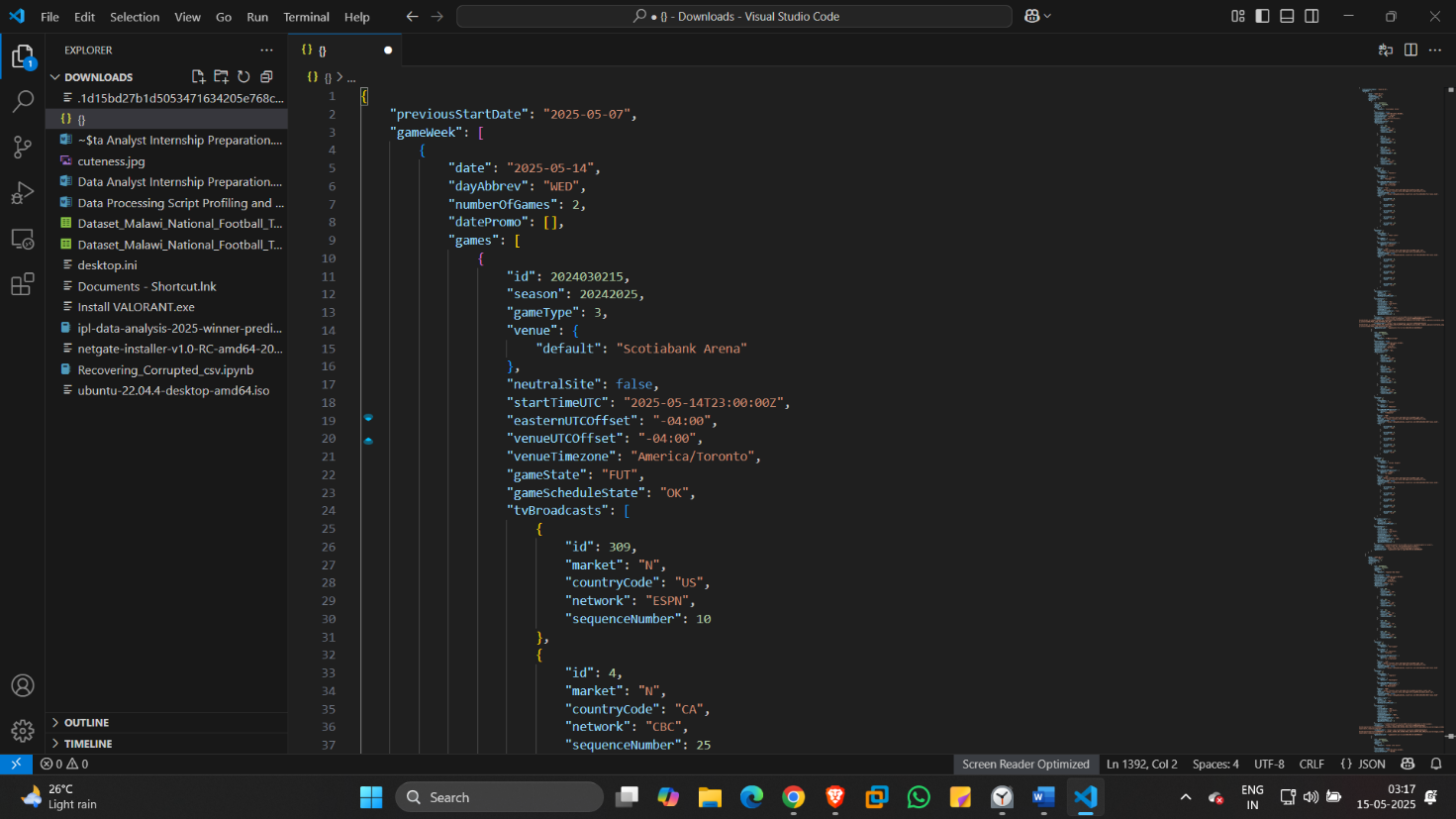


These responses returned well-structured JSON data — identical to what the frontend uses to populate UI elements. I tested these URLs directly in the browser to validate access and confirmed they were publicly reachable without authentication.

**Saving and Structuring the JSON Data**

Once validated, I saved the JSON responses locally into my development environment using VS Code. This involved making requests either through:

* Opening the endpoint URLs in the browser and saving the response as .json
* Or using a lightweight Python script (via requests or urllib) to automate downloads for multiple endpoints.



The stored JSON files were then cleaned, transformed, and used for feature extraction in the training pipeline for my ML model.

**Conclusion**

Through direct observation and reverse-engineering, I was able to re-establish data flow into my machine learning pipeline despite official API access being deprecated and the new system lacking public documentation.

This method proved efficient, reliable, and replicable for similar cases where internal APIs are exposed indirectly through web client behavior. It also underscores the value of understanding browser-level communication when traditional API docs are unavailable or incomplete.