Event Web Scraping System

Technical Report

Submitted to:
Louder Technical Team

Prepared by:
Krish Srivastava

May 11, 2025

1 Introduction

This report provides a technical overview of the web scraping solution developed to extract event information from insider.in. The system allows users to fetch events by city name through an API endpoint, delivering structured event data including titles, dates, and links. The implementation utilizes headless browser automation to interact with dynamic web content.

2 Approach

The web scraping system follows a straightforward pipeline approach:

2.1 System Architecture

The solution is built as a Node.js API endpoint that accepts city names and returns event data in JSON format. The core functionality utilizes Puppeteer, a Node.js library that provides a high-level API to control headless Chrome or Chromium browsers.

2.2 Data Collection Process

1. User provides a city name through the API endpoint
2. System validates the input parameters
3. Puppeteer launches a headless browser instance
4. The browser navigates to the insider.in events page for the specified city
5. Automated scrolling ensures all dynamic content is loaded
6. DOM elements are parsed to extract event information
7. Structured data is returned to the user as JSON

2.3 Implementation Details

The implementation handles several key aspects of web scraping:

- **Dynamic Content Loading:** Implements an automated scrolling function to ensure all lazy-loaded content is rendered before extraction
- **DOM Element Selection:** Uses CSS selectors to target specific event information (titles, dates, links)
- **Error Handling:** Includes comprehensive error management for browser issues and invalid inputs
- **Data Normalization:** Ensures consistent data format with proper URL handling

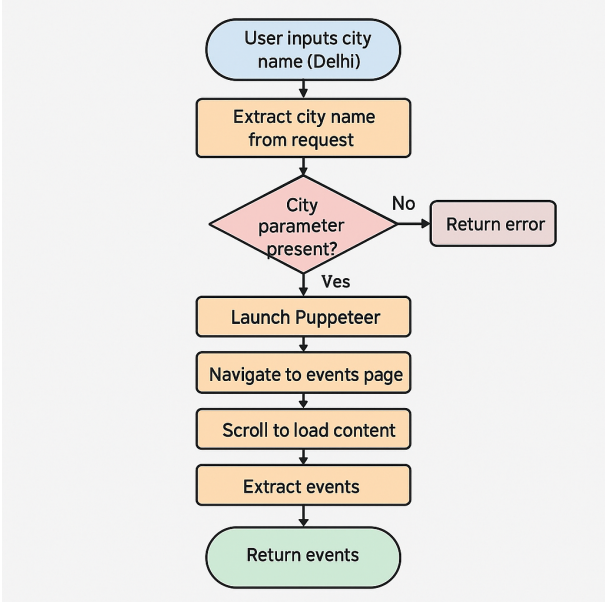

Figure 1: Event Web Scraping Process Flow

3 Challenges Faced

The development process encountered several technical challenges inherent to web scraping:

3.1 Client-Side Rendered Content

Modern websites like insider.in heavily utilize JavaScript frameworks that render content dynamically on the client side. This poses a significant challenge for traditional HTTP request-based scrapers, as the initial HTML response contains minimal content before JavaScript execution.

Solution: The implementation uses Puppeteer's headless browser capabilities to fully render the page, including all JavaScript-generated content, before attempting data extraction.

3.2 Dynamic Content Loading

The target website implements infinite scrolling and lazy loading techniques that only render content as the user scrolls down the page. This means not all events are immediately available in the DOM when the page initially loads.

Solution: A custom automated scrolling function was developed that:

- Simulates user scrolling behavior incrementally
- Monitors document height changes to detect new content
- Adds appropriate delays to ensure content has time to render
- Includes a final timeout to ensure all network requests complete

4 Improvements

While the current implementation is functional, several improvements could enhance performance and reliability:

4.1 Caching and Database Integration

Current Limitation: The system is relatively slow due to the overhead of browser automation and the time required for scrolling and content loading. Each request requires a fresh scrape, causing unnecessary load on both the scraper and target website.

Proposed Solution: Implement a scheduled CRON job that:

- Periodically scrapes events for popular cities (e.g., every 5 minutes)
- Stores structured data in a database (MongoDB or similar)
- Updates the API endpoint to query the database instead of scraping on demand

Trade-offs: While this approach would significantly improve response times and reduce load, it would increase infrastructure costs due to database requirements and scheduled execution.

5 Conclusion

The implemented web scraping solution successfully extracts event information from insider.in using Puppeteer for headless browser automation. While the current implementation is cost-effective and functional, performance improvements through database caching would significantly enhance user experience at the cost of additional infrastructure investment.