Explanation of Code:

Imports:

Various libraries like selenium, datetime, re, json, and logging are imported for web scraping, date manipulation, logging, and saving data to a JSON file.

Logging Setup:

The logging configuration is set up to output logs in a clear format (%(asctime)s - %(levelname)s - %(message)s) and with a level of INFO (which will display informational logs).

WebDriver Setup:

The webdriver.Chrome() initializes the Chrome browser for Selenium-based scraping.

extract_and_save_notice Function:

This function extracts data from each tender HTML element. It uses try-except blocks to handle exceptions if an element is not found, logging errors when they occur.

It extracts the title, date_time, location, and link, then appends this data to the global data list.

Main Scraping Loop:

The URLs of pages to scrape are stored in urls. For each URL:

The script waits for the page elements to load and scroll down and clicks "Show More" to load additional content.

It waits for the rows containing event links and loops through each one, calling the extract_and_save_notice function.

Exception Handling:

Errors are caught and logged with detailed information, ensuring that the script doesn’t break unexpectedly.

Final Data Saving:

After scraping is done, the collected data is saved into a JSON file (scraped_data.json), using json.dump() to format the data with an indentation level for readability.

Key Notes:

WebDriver Waits: WebDriverWait is used to wait until elements are present or clickable, ensuring that the script interacts with elements only after they've fully loaded.

Regular Expressions: The regex pattern (r'\w+ \d{1,2} • \d{1,2}:\d{2} [APM]+') is used to extract the date and time from the text.

Logging: Logs help track the flow of the script and capture exceptions for debugging.