**CHAPTER 3**

# **API Endpoints**

## **3.1. Dynamic API Endpoint:**

**Description**: The dynamic API endpoint (https://deal-checker.vercel.app/api/item/\*) dynamically retrieves detailed product information based on user queries.

### **3.1.1. Why Dynamic API Endpoint:**

Flexibility: Accommodates a wide range of product queries for adaptability.

Real-Time Data: Provides up-to-date information for an enhanced user experience.

### **3.1.2. Feature Explanation:**

This documentation provides a comprehensive explanation of the code used to scrape data from a website using Axios and Cheerio in a Next.js application.

### **3.1.3. Importing Required Modules:**

To begin with, the code ​imports the necessary modules: NextResponse from the "next/server" package, axios, and cheerio. These modules are required for making HTTP requests and parsing HTML respectively.

### **3.1.4. Setting Up URLs:**

The code defines two URL variables: main\_url and base\_url. These variables store the main URL and the base URL for the search query on Google Shopping.

### **3.1.5. Scraper Function:**

The scraper function is an asynchronous function that takes a URL as an input parameter. This function is responsible for making an HTTP GET request to the specified URL and scraping data from the response.

Within the scraper function, the following steps are performed:

1. Make an HTTP GET request to the specified URL using Axios and retrieve the response.
2. Load the response data into a Cheerio instance to parse and manipulate the HTML.
3. Iterate over each element with the class "sh-dgr\_\_content" and   
   extract the desired data using Cheerio selectors.
4. Construct an object with the extracted data and push it to the data array.
5. Return the data array.

### **3.1.6. GET Request Handler:**

The code ​exports an async function named GET that handles the GET requests to the server. This function takes two parameters: request and { params }.

Within the GET function, the following steps are performed:

1. Extract the value of the item parameter from the params object.
2. Construct the search URL by appending the item value to the base\_url.
3. Call the scraper function with the constructed URL and await the scraped data.
4. Return the scraped data as a JSON response using NextResponse.json().

Example Usage:

This code can be used to fetch data from a website using Axios and Cheerio in a Next.js application. The scraped data is returned as a JSON response, which can be further processed or displayed as needed.

## **3.2. Comparison API Endpoint:**

**Description**: The comparison API endpoint (https://deal-checker.vercel.app/api/comparison/\*) offers a comprehensive view of product pricing and delivery details across various online retailers.

### **3.2.1. Why Comparison API Endpoint:**

Informed Decision-Making: Enables users to make decisions based on pricing, delivery, and brand reputation.

User Empowerment: Provides the ability to choose the best option based on individual preferences.

### **3.2.2. Usage**

To use the web scraper, you need to ​import the required modules and call the scraper function with the URL of the product page on Google Shopping as the argument. The scraper function returns an array of objects containing the scraped data.

Here's an example of how to use the web scraper:

In the example above, the GET function is an API endpoint that receives a request and extracts the comp parameter from the request URL. It then constructs the URL for the product page on Google Shopping and calls the scraper function to retrieve the data. The scraped data is returned as a JSON response using the NextResponse.json method.

### **3.2.3. Scraper Function**

The scraper function is the main function responsible for scraping the data from the provided URL. It uses Axios to make HTTP requests and Cheerio to parse the HTML response.

The function follows the following steps:

1. It initializes the url2 variable and sets the baseUrl to "https://www.google.com".
2. It makes an HTTP GET request to the specified URL using Axios and sets the User-Agent header to mimic a web browser.
3. It loads the HTML response into the Cheerio library.
4. It extracts the URL of the first search result from the loaded HTML using the CSS selector ".EJbZzc".
5. If the extracted URL is "https://www.google.comundefined", it sets url2 to the original URL.
6. It initializes the mainData and content arrays to store the scraped data.
7. It makes another HTTP GET request to the extracted URL using Axios.
8. It loads the HTML response into Cheerio.
9. It extracts the image link, title, review, and review out of the product from the loaded HTML using CSS selectors.
10. It iterates over each offer row in the loaded HTML using the CSS selector ".sh-osd\_\_offer-row".
11. For each offer row, it extracts the company brand, company link, delivery details, item price, and total price using CSS selectors.
12. It creates an object with the extracted data and adds it to the content array.
13. It creates an object with the main data, including the title, review, review out of, image link, and content array, and adds it to the mainData array.
14. It returns the mainData array containing all the scraped data.

### **3.2.4. Error Handling**

The code includes error handling to catch any exceptions that may occur during the scraping process. If an error occurs, it will be logged to the console along with the message "Scrapper Error".

Please note that web scraping may be subject to legal restrictions and terms of service of the websites being scraped. Make sure to comply with all applicable laws and obtain proper authorization before scraping any website.

That's it! You now have a basic understanding of the web scraper code and how to use it to extract data from Google Shopping product pages. Happy scraping!