**5. Component Design:**

5.1 Phase1: Scrapping and Information Extraction

**5.1.1 Introduction**

Web scraping, web harvesting or crawling are various methods used for extracting information we are interested about from websites. Since the HTML format and design in each news site differs and constantly changes with each redevelopment and renewal, we decided to scrap and extract information using two popular and efficient tools, Puppeteer.js and BeautifulSoup. We will go into deep dives on each of these tools right below.

**5.1.2 Puppeteer.js**

Puppeteer is a Node library which provides a high-level API to control Chrome or Chromium over the DevTools Protocol. Puppeteer runs headless by default, but can be configured to run full (non-headless) Chrome or Chromium. Most things that you can do manually in the browser can be done using Puppeteer!

Here are a few examples of what it can do:

* Generate screenshots and PDFs of pages
* Crawl a SPA (Single-Page Application) and generate pre-rendered content.
* Automate form submission, UI testing, keyboard input, etc.
* Create an up-to-date, automated testing environment. Run your tests directly in the latest version of Chrome using the latest JavaScript and browser features.
* Capture a timeline trace of your site to help diagnose performance issues.
* Test Chrome Extensions.

So we take advantage of puppeteer to scrap some reliable news provider websites such as

* + - Global Voices
    - BBC
    - CNN
    - Sky News

We then divide the process into two steps: -

**5.1.2.1 Design**

In this step, we will design in details the algorithms that we have used in order to extract information we need from the specified providers, and then store the data for further processing.

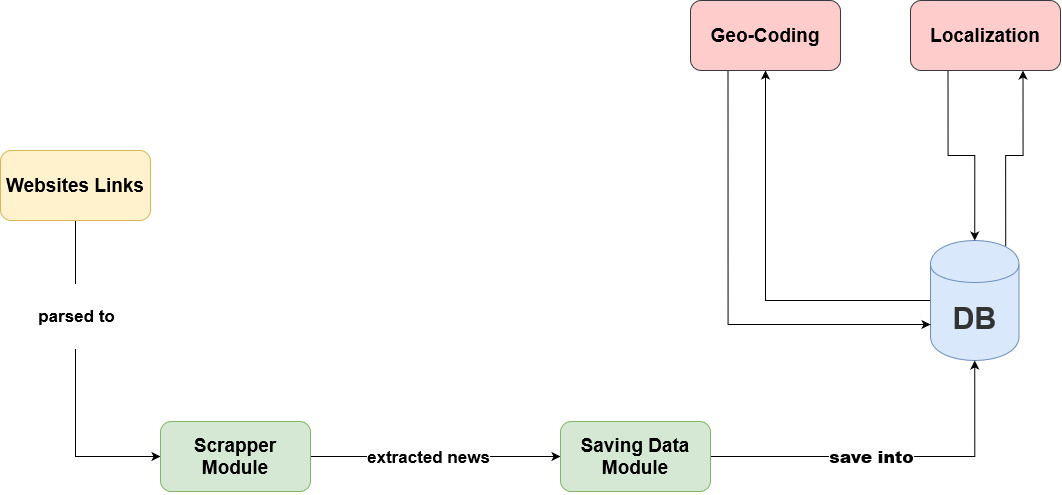
* System Architecture

Firstly, we will talk about the software architecture style we used in the scrapping system.

We used the ***microservices*** architecture, microservices are a style of **service-oriented architecture** (SOA) where the app is structured on a set of interconnected services. With microservices, the application architecture is built with lightweight protocols. The services are finely seeded in the architecture. Microservices disintegrate the app into smaller services and enable improved modularity.

Compared to its predecessor, the monolithic architecture, microservices are hands down more beneficial. You don’t need to stuff all software components and services in one big container and pack them tightly. With microservices, you can build an app with:

* greater flexibility
* high scalability
* continuous development
* systematic data organization
* time optimization
* reliability

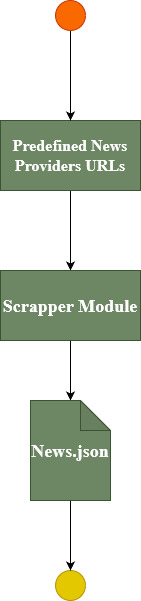
So the scrapping system includes different independent modules that interconnect with each other as will be shown in the diagram below. 

**Figure - A Diagram showing the scrapping process.**

**5.1.2.1 Implementation**

As shown previously in the scrapping process diagram, we are depending in the microservices architecture such as modules where each module is responsible for performing only single task to increase the scalability and reliability of the system.

In this step we will explain in details the scrapping-related modules, and the tools used to achieve their tasks. As we are explaining the puppeteer.js part of the scraping process so we are mainly using node.js which is a JavaScript runtime environment that executes JavaScript code outside the web browser.



**Figure – A State Diagram of the scrapper module implementation.**

Based on the above diagram we can divide the scrapping part into three steps.

* **Create the predefined URLs file:**

This file will contain the links of the web pages we are going to scrap. For example, if we are scrapping the news from https://globalvoices.org website the link will be as following depending on the location or category we are interested to scrap.

Based on the website we are scrapping from, the structure of the link object will vary. That’s because the methodology behind building the website differ from a website to another.

{

pageURL: “https://globalvoices.org/world || topics/ ${location} || ${category}”,

location: “some location”,

category: “some category”

}

The routes design of the application will play a huge part in determining the structure of the link objects structure.

The location will be provided if it’s available in the web page. If the location isn’t available, we will use the localization process that will be stated later in its own phase.

After determining the websites and the links of these websites that will be scrapped, the links are organized in a similar structure and stored in a JSON file.

This JSON file then will be parsed to the scrapper module that will be explained next.

* **The Scrapper Module:**

the basic part in the scrapping process is the scrapper module that take the web page URL as a parameter and output an array of objects, each object contains one article-related information. The information we are focusing on to be extracted are:

* <**Title**> the main title of the article.
* <**Image**> the image source URL.
* <**Summary**> the summary text of the article.
* <**Content**> the main content of the article.
* <**Tag**> the tags of the article if available.
* <**Category**> the category of the article if available.
* <**Location**> the origin location if available.

The final article object pushed to the database contains more data than those listed above, but the previous list are the data being extracted (*if available*).

Next we will discuss the scrapping module that is being written in node.js using puppeteer package. As we already know before that puppeteer initialize a headless browser to do its work, so the underlying job of puppeteer that’s been done behind the scenes can be observed if you configure puppeteer to work headful rather than headless mode.

If you set the headless mode to false in the configuration, you will notice the browser being opened automatically, and the website URL puppeteer is scrapping will be opened in the browser as you are doing all this yourself.

So let’s get into the code part of the scrapping as we will see next in the sample code provided below.



**Figure – A sample code of the scrapping module.**

In the above code sample, we are showing the methodology not the actual code we are using. So you will notice we are working in the header and summary of the article to just show you how the code work nothing more.

The first thing we are doing is requiring the puppeteer package since it’s downloaded using NPM that stands for node package manager. Since node.js follows the commonJS module system and the built-in require function is the easiest way to include modules that exists in separate files.

Then we are creating a scrapper function that act a micro-service that can be exported later and used as needed. This function is prefixed with the *async* keyword so we be able to use the async-await feature of JavaScript so that it can be implemented asynchronously without blocking other operations. The function parameter will be the page URL to be scrapped.

Then inside the scrapper function we initialize an empty array that will contain the news that will be extracted later.

Next we are launching the browser in headless mode which is the default, then we open a new page in the initialized browser to go the parsed page URL that is passed when calling the function.

Since the web page is opened, we can determine the elements we want to extract through defining its selectors. These selectors can be found using the inspect feature of the browser that exist in the browser devtools.

We use the $$eval () built-in puppeteer function that take the element’s CSS selector as the first parameter, and the second parameter will take a function that its argument will contain the HTML elements have the defined selector. Then we return the text of the found elements using map function that will map over the array of elements that match that selector and the textContent property that return the only text of the specified DOM element.

After getting the required elements’ values, in this case the header and summary values. We push them as an object in the pre-declared empty array. Then we close the running hidden browser and return the resulted news array from the function.

As you can see, the function can be used for any web page but you will have to change the selectors based on those used in this website because it varies from one website to another.

As long as this function is exported as a module, we can import it in any other module and use it to scrap the specified web pages by passing a URL as an argument to the function and receive the extracted news in some variable as an array of objects.

The actual code we are using loop over the predefined links as explained before and visit every link and scrap its contents and push the scrapped news in the array and then go to the next URL and do the same operation and son on unit the specified URLs are used to extracts its contents.

So for example suppose we have the predefined links in a links.json file and the scrapper module in the scrapper.js file, then the combination usage between them will be as follows:

const links = require(‘./links.json’);

const scrapper = require(‘./scrapper.js’);

(async () => {

try {

for (link of links) {

const news = await scrapper (link. pageURL);

}

}

} )

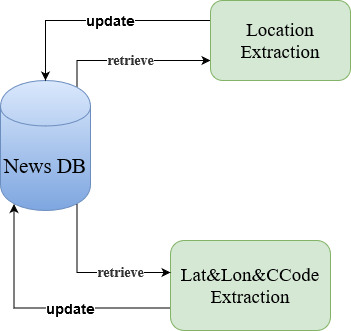
**5.1.2 Beautiful Soup**

5.2 Phase2: Localization

**5.1.1 Introduction**

The localization phase consists of two main dependent parts, the first part is extracting the location of the article through its header and content text using some NLP technology called NLTK that will be explained later, and the next part is extracting the longitude, latitude and country code of the exist location.

**5.1.1 Design**



**Figure – The Localization Process Components Diagram.**

As we can see in the previous diagram, after the news are scrapped from the defined websites, they are stored in a JSON file so that they can be processed easily to extract their location data.

The process above is implemented in a particular order. First the location extraction algorithm will take the news.json file and loop over each article object to extract its location, this extraction process is concerned most with the header and content text of each object. After the locations are extracted, the old news file is updated with the extracted locations.

This part of location extraction uses the NLTK library. The Natural Language Toolkit (**NLTK**) is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

Then it’s the turn of the geo-coding algorithm that output the latitude, longitude and country code of the locations that exist in the news. Those locations can be either the ones that are being scrapped or the ones being extracted.

The geocoding algorithm is implemented using another node.js library called node-geocoder. node-geocoder library take a location text as input and output an object contains a lot of information about that location been parsed. These information includes latitude, longitude, country code, city, zip code and even street name.

So the geo-coding algorithm we wrote take the updated news file that contains the extracted locations and process these news objects and update the file again with the new geocoding information such as latitude, longitude and country code of the exist locations.

**5.1.1 Implementation**

* Location Extraction
* Geo-Coding

For this part, we are using node-geocoder library which in a node.js library used for geocoding and reverse geocoding but here we use it for just geocoding.

Geocoding is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude), which you can use to place markers on a map, or position the map.

We are using HERE maps as a provider for the node-geocoder library. HERE maps are high quality locations APIs that can be used for geocoding services.

const nodeGeocoder = require(‘node-geocoder’);

const options = {

provider: ‘here’,

apikey: ‘SOME API KEY’

}

Const geocoder = nodeGeocoder(options);

Const getLatLon = async location => {

Try{

const data = await geocoder.geocode(location);

const latitude = data[0].latitude;

const longitude = data[0].longitude;

const countryCode = data[0].countryCode;

return {latitude, longitude, countryCode};

} }

**Figure – The Geocoding Process Sample Code.**