**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.

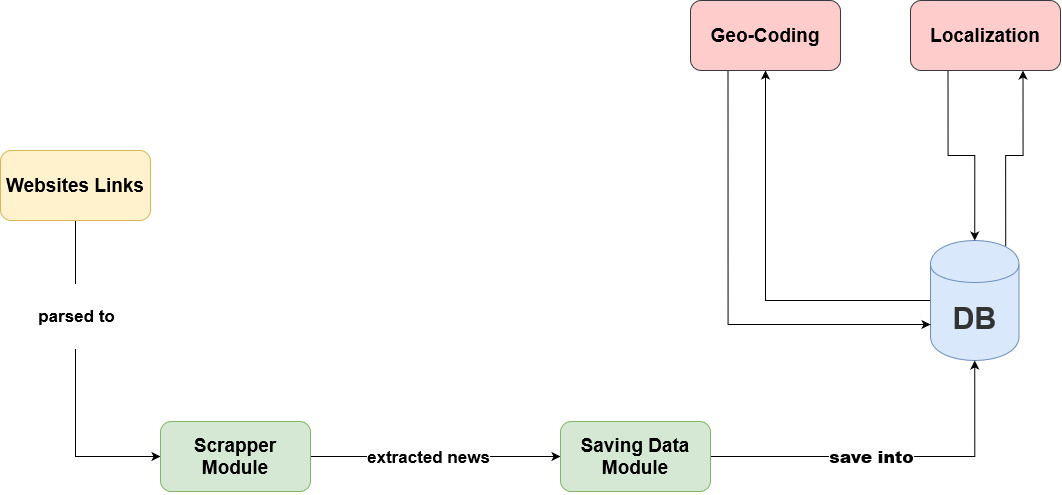
* System Architecture

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

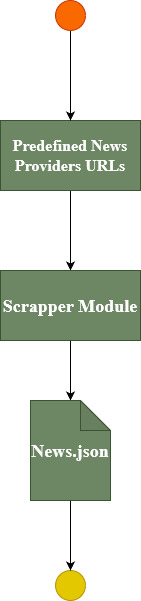
So the scrapping system includes different independent modules that interconnect with each other as will be shown in the diagram below. 

**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);

}

}

} )

**5.1.3 Beautiful Soup**

* Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.
* Also, to load the source code of the web page we use Requests.

**Main elements for BeautifulSoup scrapping**

1. Creating a Response ‘r’ to save source code



1. Making the Soup.



* Among different types of parsers, we have chosen to work with “lxml” parser as it’s Very fast and Lenient.
* Beautiful Soup transforms a complex HTML document into a complex tree of Python objects. But you’ll only ever have to deal with about four *kinds* of objects: Tag, Navigable String, BeautifulSoup, and Comment.

1. We will work with a Tag object that corresponds to an XML or HTML tag in the original document:



**Required Tags**

* Here we specify the most common tags used for retrieving data.

1. <div>
2. <p>
3. <h> (1 to 6)
4. <span>
5. <img>
6. <a>

**5.1.3.1 Design**

In this step, we will design in details the producer and algorithm that we have used in order to extract information we need from Webpage and store them for further processing.

Algorithm

1. Webpage are parsed in order to retrieve the contents of the item’s tags, we only emphasize on Title, imgUrl, Category and Content. We have chosen a number of URLs to get the corresponding page content “Response”. In this project, the number and the scope of the URLs would be predefined as follows:

However, the system can function for any Webpage and is not restricted to the predefined URLs.

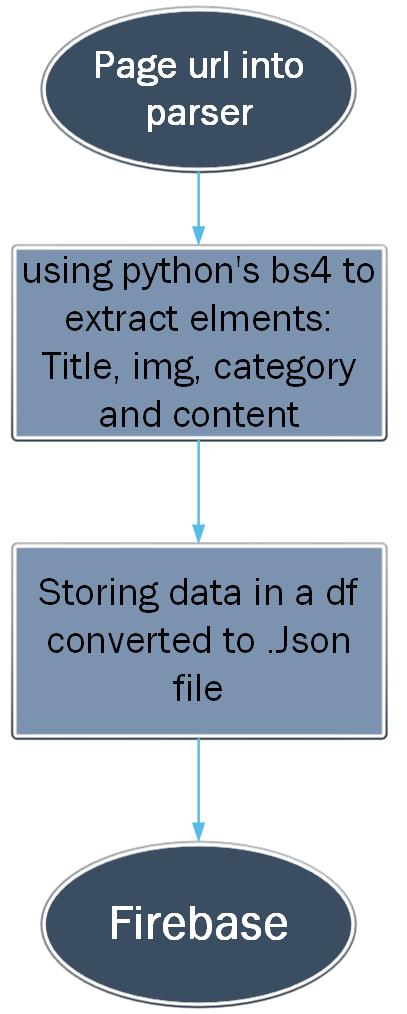
1. After parsing the data, we need to store them in order to use them in next phases, so we have created a data frame using Pandas.

Then converting into a .json file for further processing.

**5.1.3.2 Implementation**

Below is a state diagram of the steps that will guide us to the intended outcome. The tools we will be using are explained within each procedure.

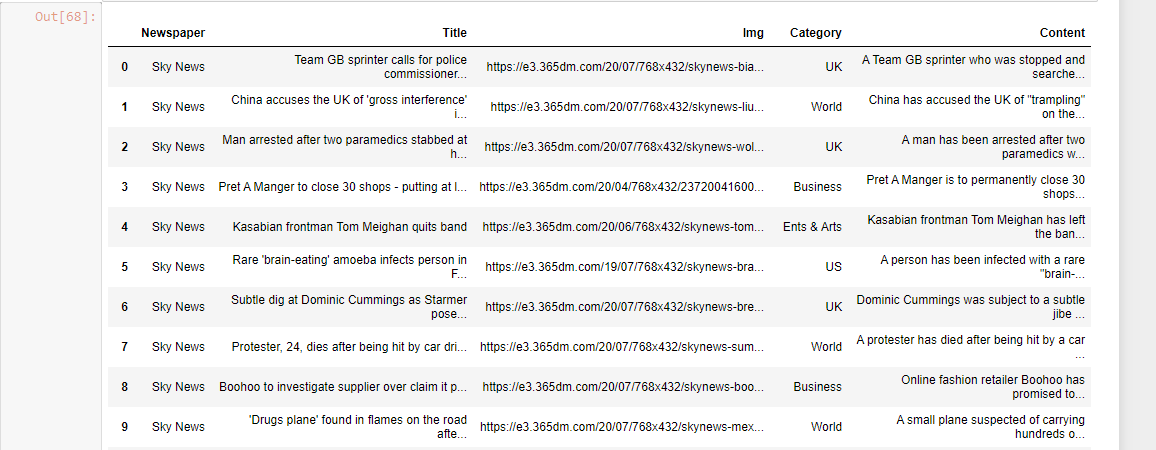
The project is implemented using Python2.8 using iPython Editor “Jupyter notebook”. Throughout this step, we will be explaining the basic of functions of each procedure in the diagram.



**Getting page content**

Our main aim in this step is to get the Page content of different news sources after we’ve inserted their corresponding URL links then parsing the content. To parse the extracted content, we use Python’s bs4 and lxml Parser.

**Storing in Data Frame**



**Convert to .json file**

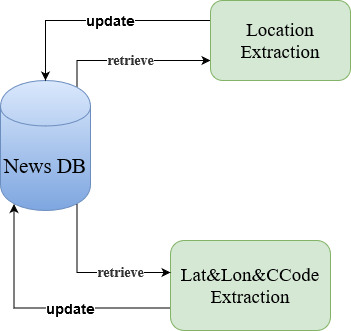


5.2 Phase2: Localization

**5.2.1 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.

**5.2.2 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.

**5.2.3 Implementation**

* Location Extraction

**Intro**

* Location extraction is a method of Entity Extraction; a semantic technology that tries to find a meaning in unstructured text. An Entity Extractor promotes words in text to concepts; this is typically realized in the form of entity tagging, where an ontology is associated with a word or phrase, such as, PERSON, PLACE, and ORGANIZATION etc. In our project, we will associate every object with PLACE.
* Once entities have been ‘tagged’, the next step is to ‘resolve’ them to a global concept or entity. This step is known as Entity Resolution. Geo-tagging is considered a more sophisticated form of entity resolution with techniques that include associating geographic coordinates. For instance, we not only want to know that New York City is a LOCATION, but also, that it's center latitude and longitude is 40.7142° N, 74.0064° W, the location is in the "New York State" administrative district, and in the country of the United States of America.

**So, how do we extract location?**

Python introduces a library that can handle Human Language “Natural Language” called NLTK:

* Natural Language Toolkit or simply “nltk” is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to [over 50 corpora and lexical resources](http://nltk.org/nltk_data/) such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.
* NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”
* This process is based on Natural Language Processing.

**What is Natural Language Processing?**

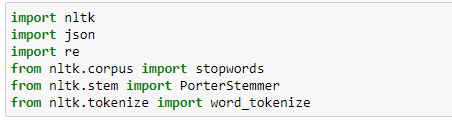
Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language.

The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable.

**Alongside nltk we will use json files that contain both news and countries code.**

**Main elements for extraction**

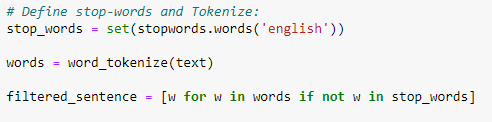
1. Importing required tools



1. Reading news data.



1. Getting data ready for processing:



* Here we go through different steps:

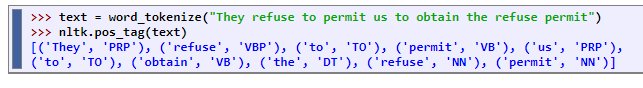
1. Stop words refer to unnecessary words that won’t affect the process of extraction but will increase the speed of the process as these words [The, And, to …] help humans understand the sentence.
2. When tokenizing we have two options word or sentence tokenize, here we choose word as we looking for countries’ names.

**Design**

In this step, we will design in details the producer and algorithm that we have used in order to extract countries names we needed from text and store them for further processing.

**Algorithm**

1. A part-of-speech tagger



* processes a sequence of words, and attaches a part of speech tag to each word.
* Among the list of POS tags available we are interested in a particular set of them which represents the most popular cases we will find a country name on it:

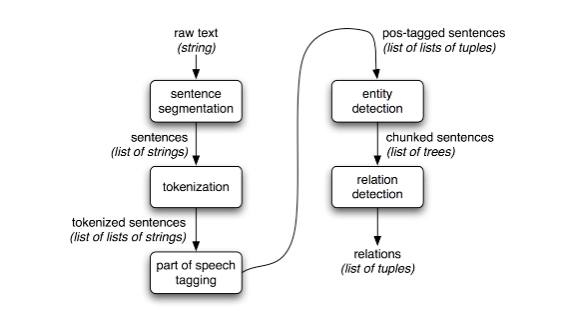
['NNP’, 'JJ', 'NNPS', 'VBP', 'NN']

A problem here is that countries like “South-Africa” won’t be identified as it divided into two words “South” and “Africa”.

**Solution: -**

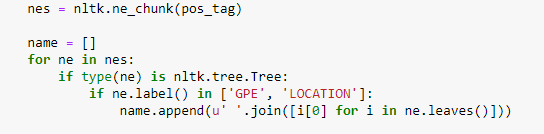
1. Identify named entities

Information comes in many shapes and sizes. One important form is structured data, where there is a regular and predictable organization of entities and relationships. For example, we might be interested in the relation between companies and locations. Given a particular company, we would like to be able to identify the locations where it does business; conversely, given a location, we would like to discover which companies do business in that location. If our data is in tabular form, such as the example in [1.1](https://www.nltk.org/book/ch07.html" \l "tab-db-locations), then answering these queries is straightforward.



Information Extraction Architecture

Example:

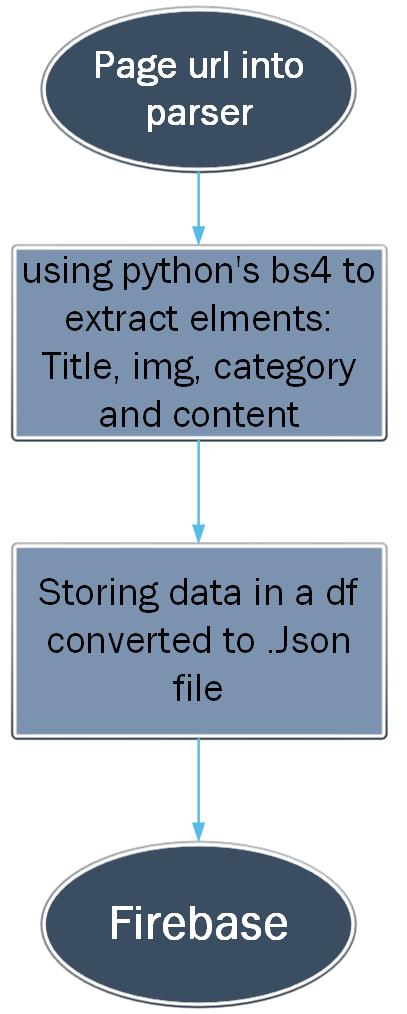


“GPE” and “LOCATION” as they refer to the name of country.

**Implementation**

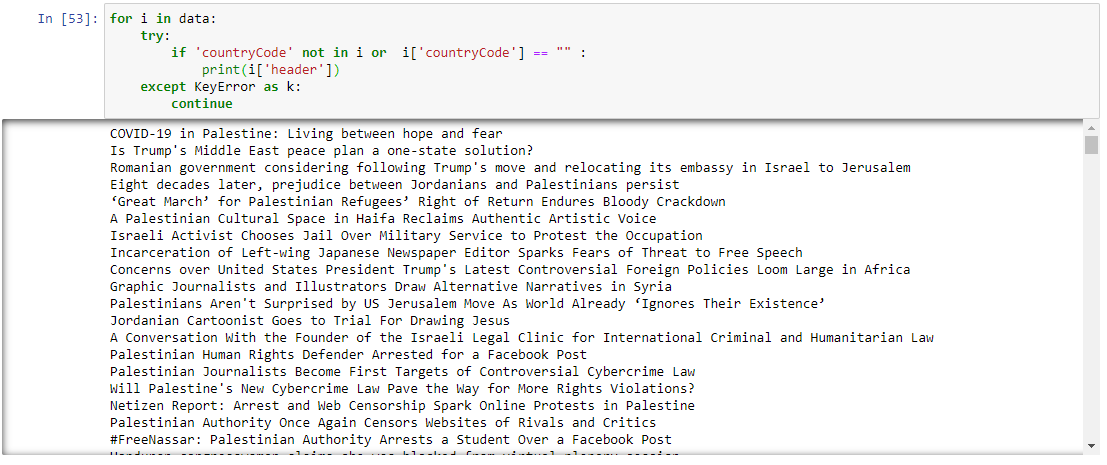
Below is a state diagram of the steps that will guide us to the intended outcome. The tools we will be using are explained within each procedure.

The project is implemented using Python3.8 using iPython Editor “Jupyter notebook”. Throughout this step, we will be explaining the basic of functions of each procedure in the diagram.

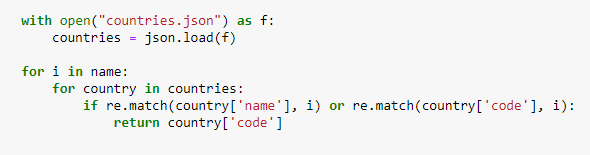


**Getting title of news**

Our main aim in this step is to get the Header of news that will be passed to extract location from it.



**Retrieve country code**



* 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};

} }

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

5.3 Phase3: Saving Data

**5.3.1 Introduction**

After the data is ready and the localization and categorization process are successfully implemented, the next step is to save these data to some database. Here we are using firebase as our DB service.

Firebase provide a suite of awesome services such as

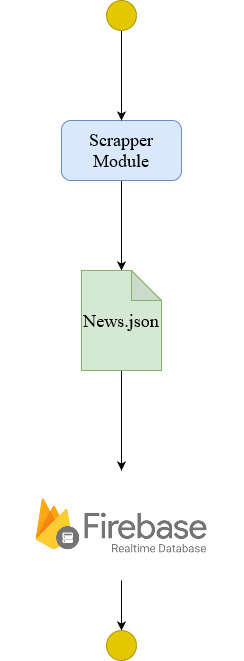
* < Cloud Firestore > is a cloud-hosted, NoSQL database that gives you synchronization between users and devices.
* < Firebase ML >Brings powerful machine learning features to your mobile app.
* < Cloud Functions > Extend your app with custom backend code without needing to manage and scale your own servers.
* < Authentication > Manage your users in a simple and secure way.
* < Hosting >Simplify your web hosting with tools made specifically for modern web apps.
* < Cloud Storage > Store and share user-generated content like images, audio, and video.
* < Realtime DB >is Firebase's original database. It's an efficient, low-latency solution.

But for our purposes, realtime DB service was enough for out database part.

The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in realtime.

Realtime syncing makes it easy for your users to access their data from any device: web or mobile, and it helps your users collaborate with one another.

**5.3.2 Design**



**Figure – State Diagram for Saving Data into Firebase DB.**

As shown in the above diagram, the news data after being scrapped and stored in a local JSON file to be processed for localization and categorization. The data need to be stored in a persistent database.

Realtime DB is optimized for offline use as When your users go offline, the Realtime Database SDKs use local cache on the device to serve and store changes. When the device comes online, the local data is automatically synchronized.

**5.3.3 Implementation**

As we said before that we use the microservices architecture for the scrapping system, so we created an independent service for saving data into database. The service is presented as a module in node.js application.

For integrating the firebase services we need in node.js backend applications, we use the firebase-admin NPM package to be able to use the realtime DB.

Firebase-admin node.js SDK enables access to Firebase services from privileged environments (such as servers or cloud) in Node.js.

The saving module is exported in the end so that it can be imported and used in other services.

As you will see in the following code sample, we need some firebase configurations to be required so it could be parsed to the firebase-admin SDK and be able to use the firebase realtime DB service.

After initializing the database reference we could use it to make queries to the current node reference.

Then we create a saving function that take the news array as input and determining that it will be saved under the news node as a main node.



**Figure – Sample Code of Saving News into Firebase DB.**

As you seen in the above code figure, the saving function is defined as asynchronous so that it can’t block any other operations during running.

During pushing data into the firebase database, firebase automatically create some unique ID for each new article being pushed in the specified node, this unique ID is based on the timestamp so that it could be ordered chronologically later when retrieving.



**Figure – Random Screenshot from the firebase DB.**

The above screenshot shows a list of nodes identified by some unique ID. Each list node contains an article data, so you can see the available news information in the collapsed node.

After the scrapped news data are finally stored in the firebase, the whole scrapping system role can be considered as done and it’s the turn of the web and mobile application to retrieve these data and display them in different ways as you will see in the next parts.

One of the best features of the firebase realtime DB that it provides a RESTful API endpoint to the database it has. So we can use it in our application to retrieve data.

For example, if we postfix the database URL with “. json” and hit this endpoint in the browser or some API testing software such as postman. It will return a json object containing the data under the specified node. In addition to the ability to filter these data in just the URL by passing the correct query parameters as you will see right now.

* The following command will return the whole objects under the news node in the database.
* CURL –X GET ‘https://glopper-f830f.firebaseio.com/news.json’
* The next command will filter the retrieved news based on the query parameters which will be in this case:
* Ordering by the views property
* Limiting to the first three news objects
* CURL –X GET ‘https://glopper-f830f.firebaseio.com/news.json?orderBy=%22views%22&limitToFirst=3’

**Snippet – Example Commands of the Firebase DB RESTful APIs.**

5.4 Phase2: Graphical User Interface - GUI

**5.4.1 Mobile Application**

**5.4.2 Web Application**

As another platform besides the mobile platform. We provide a web interface that serve the same database and offers similar features as the mobile application does.

Next we will discuss the web-specific tools and services each tool provides. Firstly, we need to know that the infrastructure of every web application is based on three basic technologies.

* <HTML> stands for Hypertext Markup Language that’s used to organize and describe the structure of the web page.
* <CSS> stands for Cascading Style Sheet and it’s a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents.
* <JavaScript> is a client scripting language which is used for creating web pages. It is a standalone language developed in Netscape. It is used when a webpage is to be made dynamic and add special effects on pages like rollover, roll out and many types of graphics.

Before getting into the tools part, we want to discuss the architecture we used for the web application, as we used the microservices architecture for the scrapping system, we use MVC architecture for the web application.

* Model-View-Controller <MVC>

It’s an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application.

The MVC model or "pattern" is commonly used for developing modern user interfaces. It provides the fundamental pieces for designing a programs for desktop or mobile, as well as web applications.

It works well with object-oriented programming, since the different models, views, and controllers can be treated as objects and reused within an application.

* Model

A model is data used by a program. This may be a database, file, or a simple object, such as an icon or a character in a video game. In our case it’s the firebase database.

* View

A view is the means of displaying objects within an application. Examples include displaying a window or buttons or text within a window. It includes anything that the user can see.

* Controller

A controller updates both models and views. It accepts input and performs the corresponding update. For example, a controller can update a model by changing the attributes of a character in a video game. It may modify the view by displaying the updated character in the game.

The MVC architecture is popularly used with node.js backend applications and Express.js particularly. We will go through these tools next in the discussion part.

Next we will deeply dive into the main tools used in the web application and discuss what each tool does in details, either on the Front-End part (client-side) or on the Back-End part (server-side).

* **EJS – Embedded JavaScript**

Is a simple templating language that lets you generate HTML markup with plain JavaScript. No religiousness about how to organize things. No reinvention of iteration and control-flow. It's just plain JavaScript.

Main EJS Features:

* Sub-template includes
* Simple template tags: <% %>
* Custom delimiters (e.g., use [? ?] instead of <%
* Both server JS and browser support
* Static caching of intermediate JavaScript
* Compiles with the Express view system

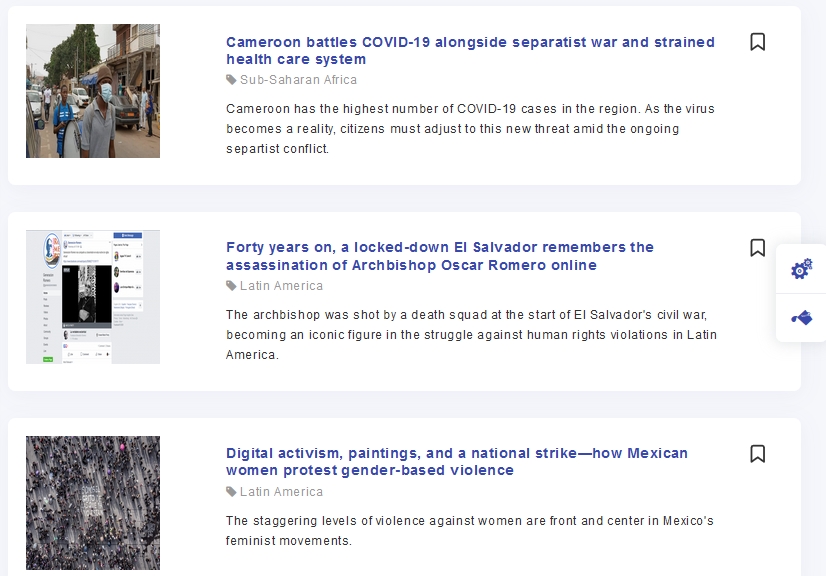
We are making use of the layouts advantage of EJS engine, meaning that we are specifying some default layouts to be inherited in any view we want it to use this layout.

For example, if we have the same header, footer, navbar and sidebar in all views, since it will be inefficient to rewrite the same code again in each view.

And here where the layouts shine, it can be considered as components that can be injected in any custom view. This concept of repeatable components is one of the best practices in software development in general since it saves as time and cost.

The following two screens show a sample code snippet of EJS and its result. The result screen is in the recent news view that retrieve and display the most recent news from the DB.





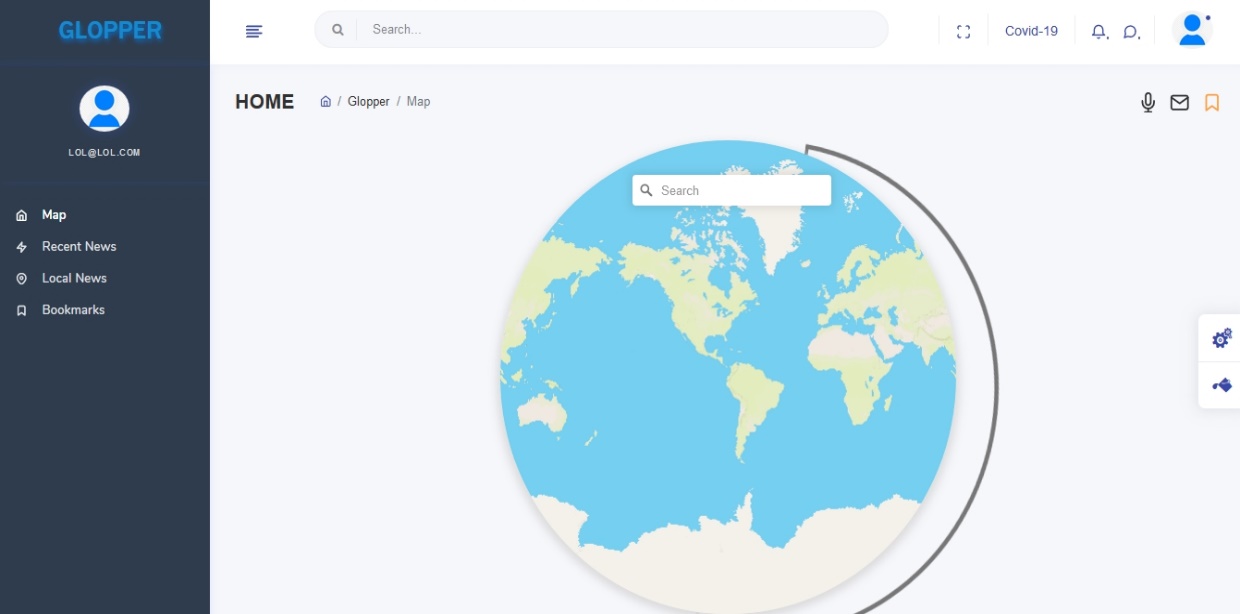
* Side Bar

The application has a fixed sidebar that ease the navigation of the main services of the app such as map, recent news, local news and bookmarks.

* Map – Home Page

The map link in the sidebar redirects to the home page that contains the circular map created using mapboxGL.

This map part will be explained in details within the mapbox GL part.

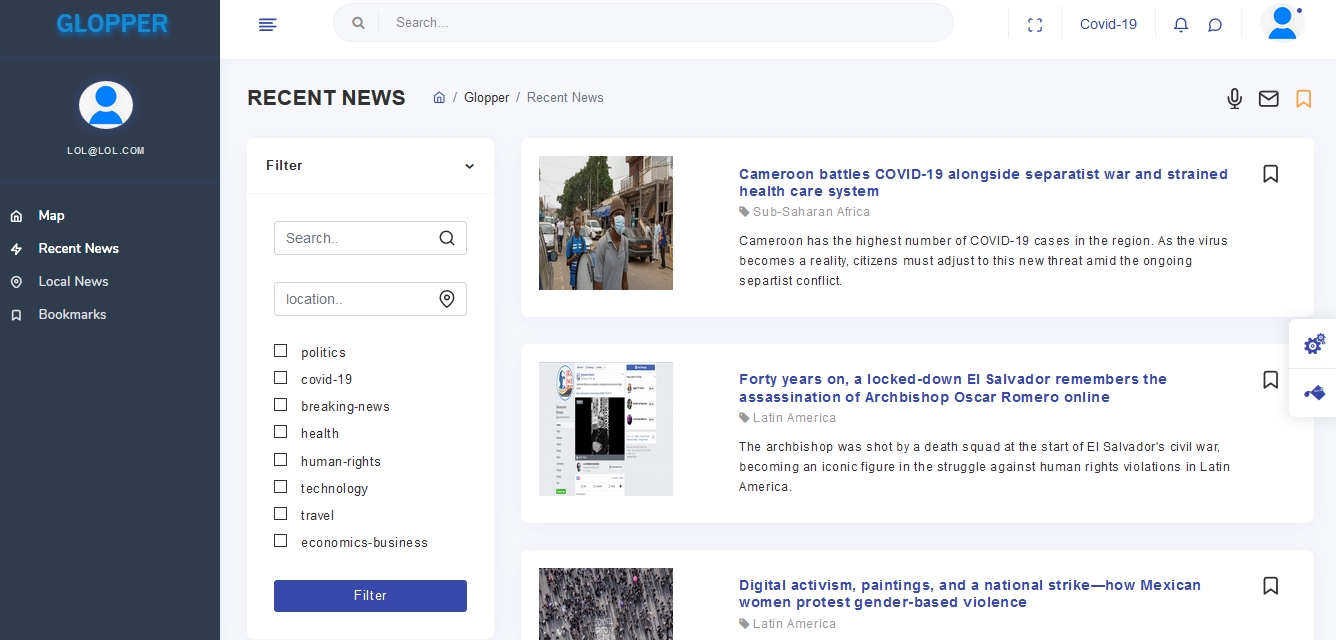


**Figure – Home Page (Map)**

* Recent News

The recent news part provides the user the ability to check the most recent news which exist in the database.

The recent news can be retrieved be implementing a query to order the news based on their unique id and get The last news that are ordered chronologically. There is a service function that its purpose is to get the recent news. We will go through it later in express.js part.



**Figure – Recent News page.**

* Local News

The local news link redirect to a web page with the same previous layout but with different data. As the name implies that it displays the news based on the user location. We get the location of the user using some backend service that will be shown in the following snippet.

async function goToLocalNews() {

let res = await fetch('http://ip-api.com/json');

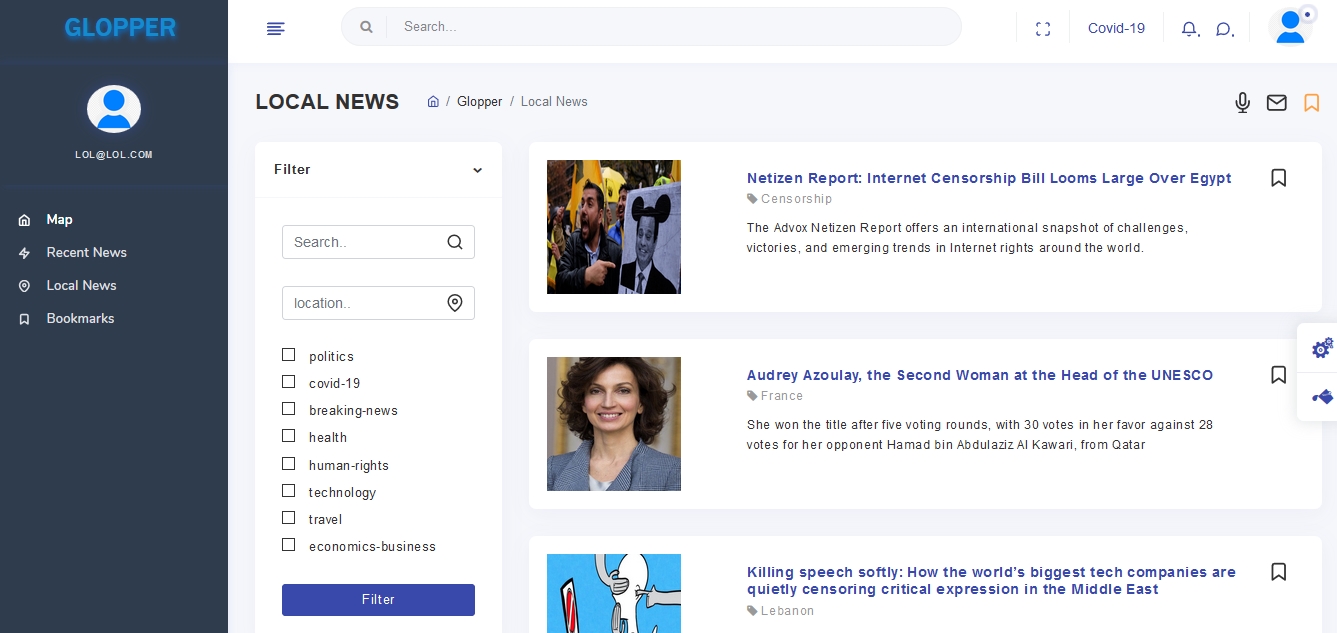
let data = await res.json();

let country = data.country;

window.location.href = `/news/local?country=${country}`

}

**Snippet – Getting The User Location.**



**Figure – local News page.**

* Bookmarks – Saved News

You will notice that every article in any view has a bookmark icon aligned with the header of the article.

If you click this icon the corresponding article will be saved and can be displayed in the bookmarks page.

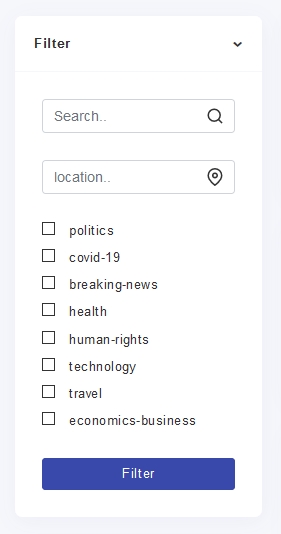
But this feature is only available for the logged in users, if not logged user clicked on the bookmark link, the user will be redirected to page says that he/she should log in so he can use this feature.

In simple words, the bookmarks feature is only available to the authenticated users, so we can save their data with their own unique saved news.

* Filter bar

You will notice the there is a filter bar in every view, this filter feature is mainly used to filter the current news based on the selected categories.

Say you want to get the news that have the category “politics” or you can check more than one category in the same time to get the related news. Or you can write a location in the text box to get news in the inserted location with the checked categories.



**Figure – Filter By Category Bar.**

* **Mapbox GL JS**

Mapbox is the location data platform for mobile and web applications. It provides building blocks to add location features like maps, search, and navigation into any experience you create. It’s changing the way people move around cities and explore our world.

Mapbox GL JS is a JavaScript library that uses WebGL to render interactive maps from vector tiles and Mapbox styles. It is part of the Mapbox GL ecosystem, which includes Mapbox Mobile, a compatible renderer written in C++ with bindings for desktop and mobile platforms.

* <script src='https://api.mapbox.com/mapbox-gl-js/v1.11.1/mapbox-gl.js'></script>
* <link href='https://api.mapbox.com/mapbox-gl-js/v1.11.1/mapbox-gl.css' rel='stylesheet' />

<div id='map' style='width: 400px; height:300px;'></div>

<script>

mapboxgl.accessToken = 'pk.eyJ1IjoiYXBkb2Vsc2FlZCIsImEiOiJja2IxZWE3NHEwMTlqMnFzNXI5ZGFkc3RvIn0.Ltca2MhPqyWI0xIJaE4Imw';

let map = new mapboxgl.Map({

container: '<your HTML element id>',

style: 'mapbox://styles/mapbox/streets-v11',

center: [-74.5, 40], // starting position [lng,lat]

zoom: 9 // starting zoom

});

</script>

**Snippet – Sample Code of using Mapbox GL library for creating Maps.**



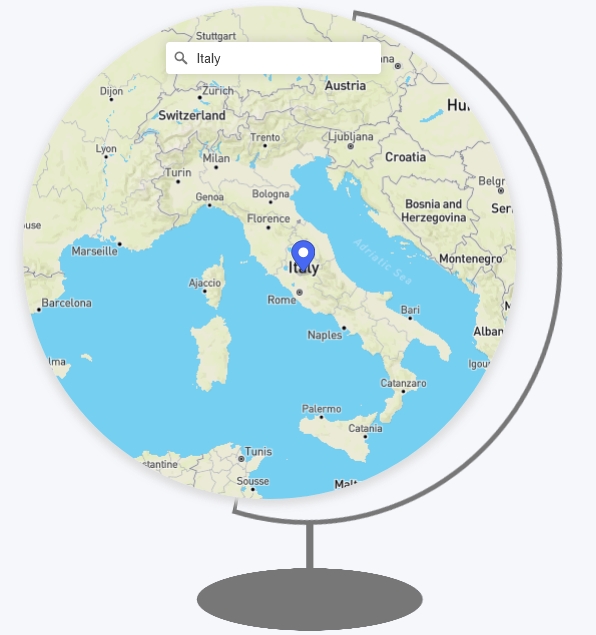
**Figure – The Main Map Created Using Mapbox GL**

The circular map is created using Mapbox GL library, but the additional drawing is done using the Canvas API technology that enables us to draw over the DOM of the web page.

The main objective of the map is to ease the process of getting news based on the location the user will click on in the map.

Say for example, there is a user wants to know the news of Italy, the user will search by the location he is interested in within the search textbox in the top middle of the map. The map will smoothly zoom in to the selected location entered in the textbox.

After the map zoom in to the inserted location, if the user clicks on this location, the web app will redirect to a web page containing a list of news related to the selected location.



In simple words, you want to know the current news about some location, search for it in the map and click on it. Just as same as the real map ball that people use to inspect the world locations. Our map is better since you can zoom in and out and drag the map right, left, top and bottom so you can see different parts of it.

Also, you can enter the latitude and longitude of some location, and the map will figure out the location of this coordinated and zoom in to it so you can click on it and get the news you want.

The mechanism of how the map redirects to web page based on the clicked location, is implemented using Express.js which is a node.js backend framework that lets you handle multiple different http requests and is mainly used for building RESTful APIs, and we will discuss it later in more details.

For the map part, we are listening to a click event on the map, and when any click event occur we get its coordinated and use the here maps provider to get the location country by parsing the coordinates as parameters and after getting the location name, the application redirects to a route with the country name on it, this route is handled using express in the backend, where it implements a query on the DB to retrieve news that have this location. Then the news list view is rendered with the retrieved data from the DB.

**map.on('click',(evt) => {**

**findAndGoTOThisLocation(evt.lngLat);**

**})**

**Async function findAndGoTOThisLocation(latlon){**

**Let lat = latlon.lat;**

**Let lng = latlon.lng;**

**Let apikey = ‘<HERE MAPS API KEY>’;**

**Let res = await fetch(‘https://reverse.geocoder.ls.hereapi.com/reversegeocode.json?prox=${lat}%2C${lng}** **%2C250& =${apikey}’);**

**Let countryName= await res.json().result[0].location.value;**

**Window.location.href=”/news/mapSearch?country=${countryName}”;}**

**}**

**Snippet – Handling The Click Event On The Map.**

* **Express.js**

Express is a Fast, unopinionated and modular web framework for Node.js that lets you handle multiple different http requests and is used mainly to build RESTful APIs for web and mobile applications.

We use express mainly for providing server-side rendering with the co-operation of EJS framework and its node.js packages that ease the rendering such as express-ejs-layouts package.

The basic building block of the express backend is the routing technology. Routing refers to how an application’s endpoints (URIs) respond to client requests.

You define routing using methods of the Express app object that correspond to HTTP methods; for example, app.get() to handle GET requests and app.post to handle POST requests.

These routing methods specify a callback function (sometimes called “handler functions”) called when the application receives a request to the specified route (endpoint) and HTTP method. In other words, the application “listens” for requests that match the specified route(s) and method(s), and when it detects a match, it calls the specified callback function.

**router.get('/news/recent', async (req, res) => {**

**try{**

**const recentNews = await getRecentnews();**

**return res.render('news/list',**

**{ title: 'Recent News', blogs: recentNews });**

**}catch(err){**

**console.log(err)**

**}**

**});**

**Snippet – Handling The Recent News Route.**

* Helper Functions

We depend mainly on the custom helper function we are creating to help us develop fast and efficiently.

Helper functions are those function that will be repeated more than once, so they are created as function in some related module and exported, so they can be imported in another module and used.

Most of the helper functions we created, are those who deal with the database, such as retrieving news based on some location, or retrieving news based on some category and son on. The following snippet will show you the helper function we wrote to get the local news from the firebase database, so each time we want to get the recent news, we import this function and implement it and it will return the recent news in an array.

**const getRecentnews = () => {**

**let recentNews = [];**

**return new Promise(function(resolve, reject){**

**const recentNewsRef = dbRef.ref('news').orderByKey().limitToLast(10);**

**recentNewsRef.on('value', async (snapchot) => {**

**snapchot.forEach(function(child){**

**const childKey = child.key;**

**const childData = child.val();**

**childData['id'] = childKey;**

**recentNews.push(childData);**

**})**

**if(recentNews.length>0){**

**resolve(recentNews)**

**} else {**

**reject(‘some error happens’)**

**}**

**})}**

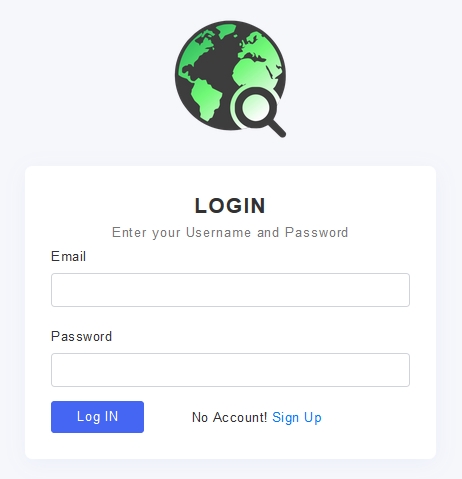
**)}**

**Snippet – Helper Function to get the Recent News.**

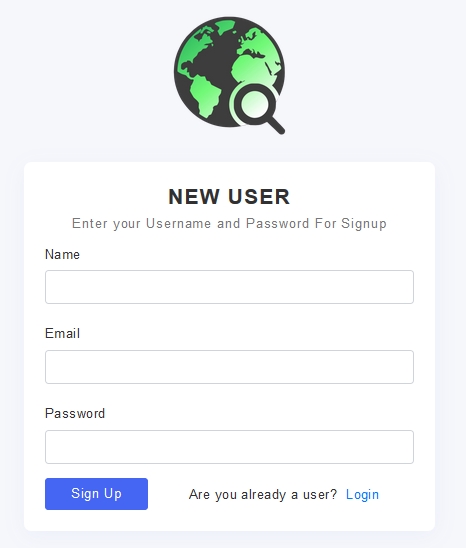
* **Firebase Authentication**

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers and social media platforms and more.



**Figure – Log-In Screen processed using firebase.**

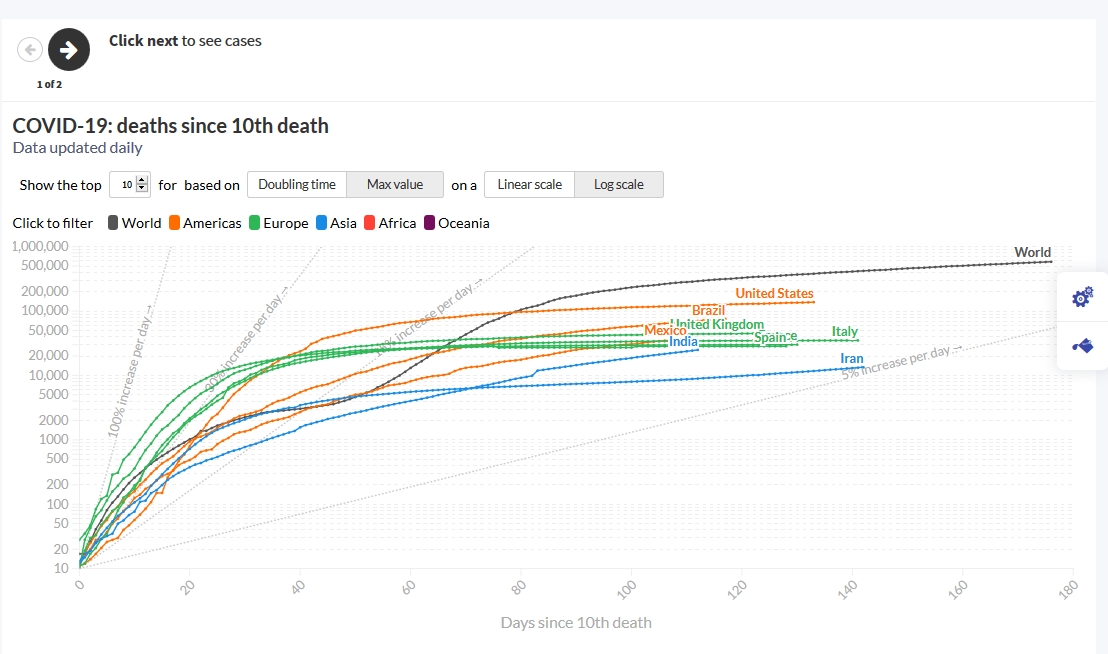


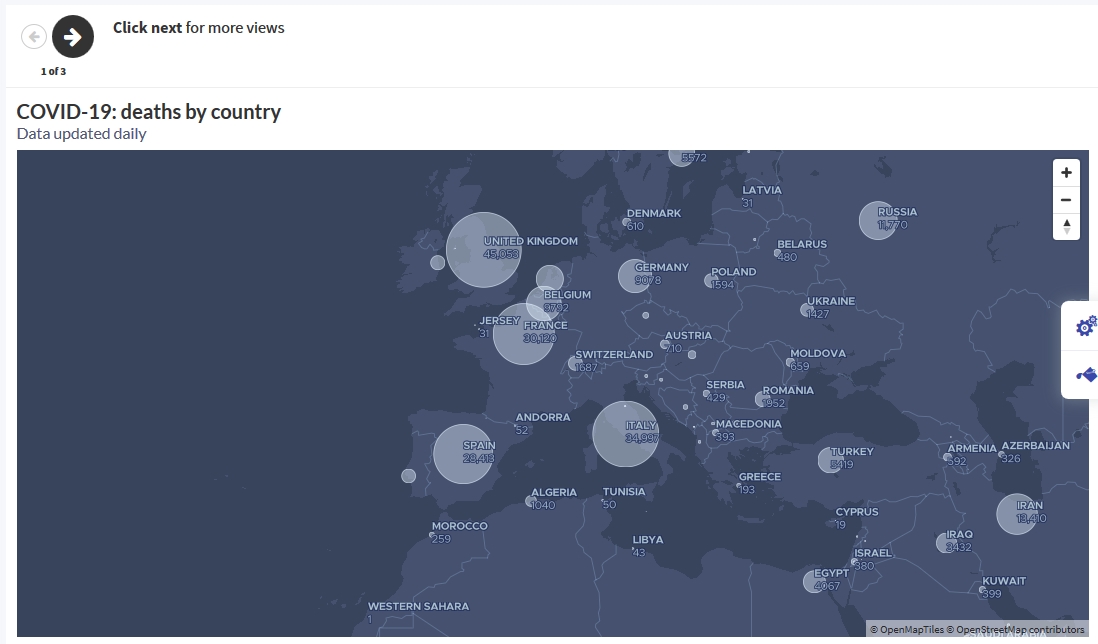
**Figure – Sign-Up Screen processed using firebase.**

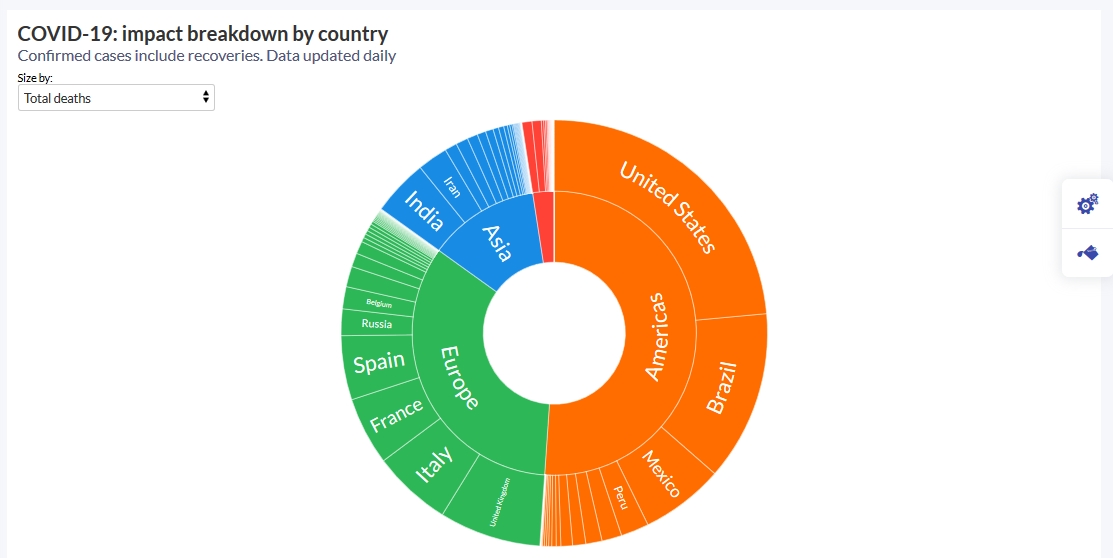
* **Covid-19**

We provide some statistical based maps and visualizations that are based on some coronavirus daily updated database.

Some of them are built using d3.js which is a JavaScript library for producing dynamic, interactive data visualizations in web browsers. It makes use of Scalable Vector Graphics, HTML5, and Cascading Style Sheets standards.









5.5 Phase5: Hosting & Update

**5.5.1 Hosting**

We will host our database in firebase realtime DB as previously explained. And the web application will be hosted on the heroku platform. Heroku is a container-based cloud Platform as a Service (PaaS). We can use Heroku to deploy, manage, and scale modern apps.

Heroku Features:

* Heroku Runtime - Apps run inside smart containers in a fully managed runtime environment, it handles everything critical for production — configuration, orchestration, load balancing, failovers, logging, security, and more.
* Scaling - Heroku scales in an instant, both vertically and horizontally. You can elegantly run everything from tiny hobby projects to enterprise-grade e-commerce handling Black Friday surges.
* Code and Data rollback - Work fearlessly — Heroku’s build system and Postgres service let you roll back your code or your database to a previous state in an instant.
* Heroku Redis - The most popular in-memory, key-value datastore — delivered as a service. Heroku Redis provides powerful data types, great throughput, and built-in support for top languages.
* Continuous Delivery - Heroku Flow uses Heroku Pipelines, Review Apps and GitHub Integration to make building, iterating, staging, and shipping apps easy, visual, and efficient.
* Data-Clips - Data Clips make it easy to keep everyone in the loop with up-to-the-second data insights.
* Deploying to Heroku

We are using heroku CLI that will shorten the deploying process. Deploying to Heroku works via Git. In essence you can deploy to Heroku by pushing to a remote repo, just like pushing to GitHub.

**git init**

**git add .**

**git commit –m “initial commit”**

**Snippet – initializing a git repository.**

**Snippet - create a Heroku app that'll correspond to our Node app.**

heroku create

Creating app...!

▸ Invalid credentials provided.

heroku: Press any key to open up the browser to login or q to exit:

Opening browser to https://cli-auth.heroku.com/auth/browser/abcd1234-b6a7-4df4-bb42-0eaf987d0637

Logging in... done

Logged in as example@gmail.com

Creating app... done, ⬢ glopper-12818

https:// glopper-12818.herokuapp.com/ |

https://git.heroku.com/ glopper-12818.git

$ git push heroku master

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 4 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 326 bytes | 326.00 KiB/s, done.

Total 3 (delta 2), reused 0 (delta 0)

...

...

...

remote: Verifying deploy... done.

To https://git.heroku.com/ glopper-12818.git

5cb9118.dd0bacd master -> master

**Snippet – deploying to heroku using the created master branch.**

**5.5.2 Updating**

In this last phase, there are some points to complete our project and make it more efficient and applicable to use, those points are:

* Automation of the scrapping system through heroku scheduler.
* Check for new feeds.

The above two tasks are interrelated in their implementation case. Since when the scrapper module run, before pushing the data into the database, it first checks if the new scrapped articles exist in the database, if it exists the articles will be ignored else it will be pushed to the database.

So it’s all about the automation task, once it’s done the other task is implemented as well.

We are implementing the automation task by running node.js script using the heroku scheduler every day at midnight.

Here is an example of how we do it.

* Deploy the scrapping system to heroku as we did with the web application above in the hosting step.
* We put the scrapper module in the node + heroku dev environment and every time this script runs it will do all the work. Remember that we use microservices architecture to module out scrapping system, so we will need to just run the main script and the other modules will be imported and implemented automatically.
* Testing the script locally. Heroku runs scheduled jobs as One-Off Dynos. This is the same thing as using your terminal to run a given script.
* Deploy to heroku and test, do our Git adds and commits and push the script up to Heroku. Now we're ready to test. When we're in the Heroku environment we have to append "heroku run" to our terminal commands to make them One-Off Dynos.
* Since the terminal command we used to run our script was "node scrapper.js" we can run a One-Off Dyno using this command:
* The last step is getting your script to trigger automagically at regular temporal intervals. First we install the scheduler from Heroku's addons page. If you haven't already, Heroku will also require that you enter your billing information. This would be used if your script exceeded Heroku's usage limits.

heroku run node scrapper.js