

BriefMe

A Solution for Curated News Consolidation and Summary



Do Man Uyen Nguyen – 300318626
Fabio Turazzi – 300318010

CSIS 4495 – 090 – Final Report

Contents

1.	Introduction	3
2.	Statement of the Problem	3
3.	Significance of the Project	3
4.	Project Report	4
4.1.	Project Overview and Objectives	4
4.2.	Development Process – Documentation and Evidence	4
4.2.1.	Module: Web scraping with Python	4
4.2.2.	Module: Summarization with Python	5
4.2.3.	Module: Database Storage with MongoDB	8
4.2.4.	Module: Web Application with Node.js/Express	8
4.3.	Software Architecture	11
4.3.1.	BriefMe – Sequence Diagrams	12
4.3.2.	BriefMe – Use Case Diagram	13
4.3.3.	BriefMe – Activity Diagram	14
4.3.4.	BriefMe – Class Diagram	14
4.4.	Design Patterns Applied	15
4.4.1.	Python DB Loader – Design Pattern	15
4.4.2.	BriefMe Web Application – Design Pattern	16
4.5.	Development Process Tasks	17
4.6.	Project Testing	18
4.6.1.	Web Scraping Module Testing	18
4.6.2.	Summarization Module Testing	18
4.6.3.	Web Application Module Testing	19
4.7.	Using the Application – Local Environment Setup and Web Access	20
4.7.1.	Local Environment Setup	20
4.7.2.	Accessing the Deployed Web Application	20
4.8.	Project Delivery: Directory Description	20
5.	Appendixes:	22
6.	References:	22

1. Introduction

This is the final report for CSIS 4495-090 Applied Research Project, developed over the course of the Winter semester of 2021.

This Academic Project consists of a web application, called **BriefMe**, developed to consolidate and summarize news articles from multiple sources with the goal of enhance access to daily news for its users. This project makes use of techniques in Web Scraping and Natural Language Processing to search and summarize articles, respectively, and displays the results in a concise manner inside a Web Application.

2. Statement of the Problem

In the modern digital world, information can be obtained by a multitude of sources, and keeping oneself informed daily can be an extensive and overwhelming process. An article published in Time magazine states that more than half of readers spend only 15 seconds on a page. This behavior can be explained by people's needs to consume more news in a limited time constraint during tight routines. Currently, a growing number of people access news from social media, according to a Forbes' article, bringing issues of misleading information by not credible sources. With this in mind, providing people with accurate news in an efficient manner has potentially huge benefits.

To solve this problem, we intended to develop an application that enhances user experience when accessing news media, by bringing consolidation and summarization of multiple credible sources. Replacing the need to access multiple websites for news exploration, our platform proposes an integrated environment in which the user can visualize titles and links for articles, find summaries to get an overview of the matter, and generate text-to-speech to further ease the process. By dint of condensing sources, the application can reduce the time browsing and increase the time actually reading and getting informed.

3. Significance of the Project

Our primary goal is to improve the experience of news consumption while providing credible and condensed information to the right audience.

In order to accomplish that, we allow users to navigate between topics of interest to explore articles. As the result, users receive credible content curated from several sources, saving time on news sites. News outlets can also gain traffic as their articles are presented on relevant topics.

Additionally, our application differentiates itself from existing solutions in the market by applying Natural Language Processing and Machine Learning techniques to perform text summarization to display summaries of article contents. This is taken into consideration as people might not have enough time to go to articles but still want to know the most important pieces of information from a range of subjects.

We understand this to be a drastic improvement upon current solutions, providing value to customers by providing information in a time-efficient manner.

4. Project Report

4.1. Project Overview and Objectives

Our main goal for this project was to develop a web application that can provide easy and fast access to reliable news. In order to achieve this goal, we have gathered news from 10 benchmark news sources and condensed the news articles to deliver essential information to our readers. We purposefully designed the application to reduce time and energy spent on browsing information.

The main tasks of this project were categorized in different modules, including: web scraping to obtain articles, text summarization, storage of article information on a database, and providing a user-friendly and resource-efficient web application to deliver the information. We will explain our approaches throughout the following sections.

4.2. Development Process – Documentation and Evidence

4.2.1. Module: Web scraping with Python

The first task in our project was obtaining the news articles. We wrote the prototype algorithm to scrape CNN using BeautifulSoup. Additionally, to circumvent webpages that generate content dynamically with Javascript, we added a layer to the scraping code using the requests-html library to pre-render the pages. As the scraping algorithm succeeded in obtaining the information that we needed, we proceeded with replicating the algorithm for other news sources.

Figure 1: Rendering Pages with Requests-html

```
#Loop through urls
for URL in relativeURLs:
    try:
        #Render page
        response = session.get(baseUrl + URL.lower())
        response.html.render(timeout=30)

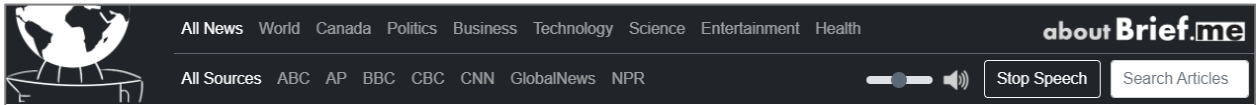
        #Find .zn containers
        containers = response.html.find('.zn')
        links = []

        # Get absolute links from zn containers
        for index, container in enumerate(containers):
            try:
                if index > 0:
                    links.extend(container.absolute_links)
                if index == 1:
                    topStories = len(list(dict.fromkeys(links)))
            except:
                errorlog.write(now.strftime("%m/%d/%Y, %H:%M:%S") + " - Failed to open container." + "\n")
```

Our criteria to choose these news sources included credibility, reputation, and relevancy. After carefully reviewing different sources, we chose to get articles from CNN, BBC, CNBC, CTVNews, AP, NPR, Global News, Business Insider, ABC News, and National Post. We categorized articles between common topics that would be suitably delivered in a concise manner, with a Business-oriented focus. The topics selected were Politics, Business, Technology, Entertainment, and Health. We also targeted Canadian

readers by having a Canadian section that delivers national news from Canadian news sources, in addition to a World section for worldwide news.

Figure 2: BriefMe Navigation - News Sources and Categories



To adapt our initial CNN blueprint for the other sources, we faced a challenge of inspecting website structures and identifying their intricacies to extract the information we needed. This process also was not always fully feasible, with the example of the inability to obtaining images from AP and CNBC, given both websites use SVG to generate the images from their database instead of hosting and referencing them. We also tailored the links from each source to filter relevant articles' links and avoid redundant information.

Figure 3: Tailoring Links Before Scraping

```
#Loop through those links and add absolute path when it is not contained
for index, link in enumerate(links):
    # Keep top curated links
    if len(curatedLinks) >= topLinks:
        break

    if ".com/videos" in link:
        # skip
        print(link)
    elif "https://" in link or "http://" in link:
        if "cnn.com/" in link:
            if index < topStories:
                curatedLinks.append(dict({"link": link, "top": "yes", "category": URL}))
            else:
                curatedLinks.append(dict({"link": link, "top": "no", "category": URL}))
        else:
            if index < topStories:
                curatedLinks.append(
                    dict({"link": 'https://www.cnn.com' + link, "top": "yes", "category": URL}))
            else:
                curatedLinks.append(
                    dict({"link": 'https://www.cnn.com' + link, "top": "no", "category": URL}))
    ant:
```

4.2.2. Module: Summarization with Python

For this module, our main goal is to deliver a long news article in a short and concise form so that readers can understand the most important pieces of information without having to spend too much time reading, but still obtain more than just what the headlines offer. Even though we collectively select the outlets, the subtle bias in news headlines is still relevant and we believe the summaries themselves can improve the perception of the articles. If readers are interested in getting more details, they can follow the link to the full articles, which can benefit both the readers and outlets.

Final Report

For the purpose of our application, we have determined that an abstractive summary, in which the algorithm re-writes a summarized version of the article, would be better than an extractive summary, in which sentences are ranked and the most important ones are compiled. In order to summarize news articles, we have experimented with some of the most popular text summarization algorithms based on Transformer model, including GPT-2 and BART.

Figure 4: Text Summarization using BART

```
torch_device = 'cuda' if torch.cuda.is_available() else 'cpu'
tokenizer = BartTokenizer.from_pretrained('facebook/bart-large-cnn')
model = BartForConditionalGeneration.from_pretrained('facebook/bart-large-cnn')

def summarize_bart(text):
    article_input_ids = tokenizer.batch_encode_plus([text.replace('\n', ' ')], return_tensors='pt', max_length=1024, truncation=True)['input_ids'].to(torch_device)
    summary_ids = model.generate(article_input_ids,
                                num_beams=4,
                                length_penalty=2.0,
                                max_length=200,
                                min_length=100,
                                no_repeat_ngram_size=3)
    summary_txt = tokenizer.decode(summary_ids.squeeze(), skip_special_tokens=True)
    return summary_txt
```

After extensive experimentation, the final model we used was our tweaked version of the BART algorithm, which generated summaries with an accuracy score of 90% according to our metrics. More details about the experimentation with those algorithms, as well as the methodology used to calculate accuracy, will be described later on this report.

Figure 5: Text Summary Sample taken from Web Application



Link to full text: <https://www.bbc.com/news/world-europe-56705631>

Summarization Challenges

As we developed the summarization algorithm, a few challenges were encountered, being the main ones accomplishing desired summary accuracy and obtaining reasonable execution times. While the first one was addressed with the aforementioned algorithm experimentation, the latter was more challenging to solve.

Since Natural Language Processing algorithms require a large amount of processing, applying them systematically for over 400 articles required long wait periods. We consider this limitation to be solvable on a professional deployment scenario, in which distributed processing systems like HDFS offer a more efficient execution of Machine Learning code. We understand that this is in line with other commercial applications that make use of Big Data analytics or Natural Language Processing features. Nevertheless, with our limited resources, it is not feasible for us to implement our web application using this type of configuration.

For the scope of this project, we decided to solve this problem by reducing the number of news sources from 10 to 6, and limiting the total number of articles per news category from each source to 15 comparing to getting as many available articles as possible previously. It is worth noting that the modules for excluded sources are still included in the code, but aren't called in our main application. Therefore, we can assume that a future professional version of the application would reinclude all sources and be deployed inside a high processing server.

BriefMe - Web Scraping Scope Limitation

```
# Call scraper methods to fetch all articles
articles_cnn = scraper_cnn.scrape_cnn(15)
articles_cbc = scraper_cbc.scrape_cbc(15)
articles_bbc = scraper_bbc.scrape_bbc(15)
articles_npr = scraper_npr.scrape_npr(15)
articles_globalnews = scraper_globalnews.scrape_globalnews(25)
articles_abc = scraper_abcnews.scrape_abc(15)
# articles_nationalpost = scraper_nationalpost.scrape_nationalpost(15)
# articles_ap = scraper_ap.scrape_ap(15)
# articles_businessinsider = scraper_businessinsider.scrape_businessinsider(15)
# articles_cnnbc = scraper_cnnbc.scrape_cnnbc(15)
```

Limitation of Articles per Category

Excluded Sources

As the result of reducing the scope, we were able to decrease the total execution time from 12 hours to 4 hours using basic personal computers.

Identified Future Improvements

For purpose of improving this course, we leave the recommendation for Douglas College to provide support to the integration of Hadoop AWS Module, relieving the processing issues for future Big Data projects. It would also prove useful to demonstrate to students some expertise about working with HDFS architecture, which has been desirable in the job market. However, we also consider this solution would require students to have a strong background in the system and may not be feasible.

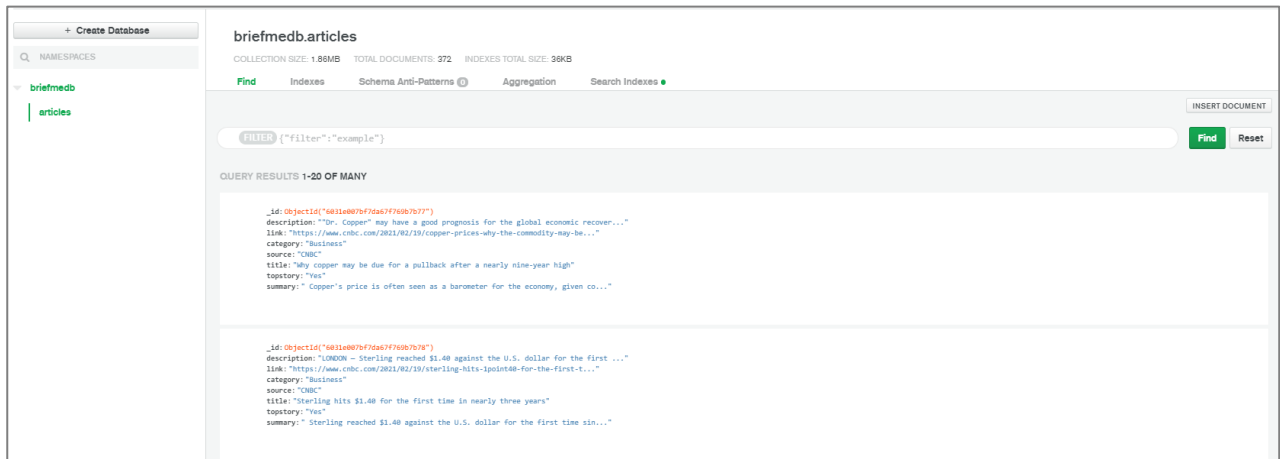
Final Report

4.2.3. Module: Database Storage with MongoDB

After we obtained the summaries of the text, we store all the information in a MongoDB collection for retrieval and display by our web page. Our database is updated with new information every time the scraping and summarizing algorithms finishes running an iteration.

We stored our data using MongoDB, on cluster from MongoDB Atlas. This cluster is connected to both our Python scripts and our Node.js web application, transferring the information generated by the Python program to the user-oriented application.

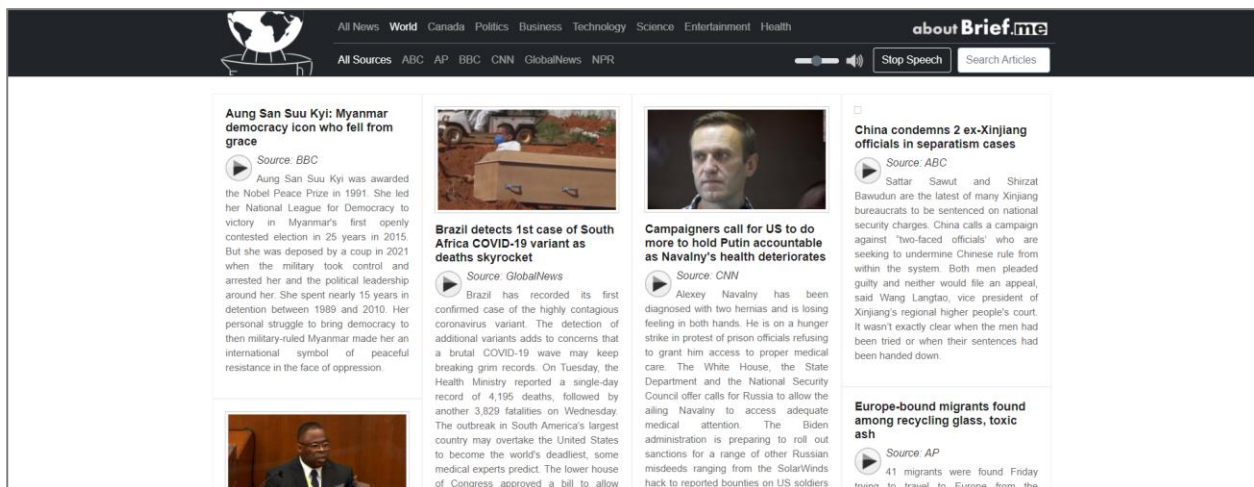
Figure 6: BriefMe Mongo Atlas Cluster



4.2.4. Module: Web Application with Node.js/Express

To deliver the information to users, the built web application was designed with a focus on simplicity and conciseness, in accordance with our goals for this solution. The application presents the collected information on small cascaded frames and offers navigation and filter options to improve browsing.

Figure 7: BriefMe – Main Page



Article Navigation

BriefMe offers three main ways for users to find articles of interest, consisting in browsing by category, source, and search results. Users can choose the topics from the navigation bar or perform keyword search, which will query this attribute from our MongoDB database. All of these filters were designed so that these three layers can work simultaneously, in order to help our readers to explore contents combining their preferences in a convenient manner.

Figure 8: BriefMe – News Articles by Source (BBC)

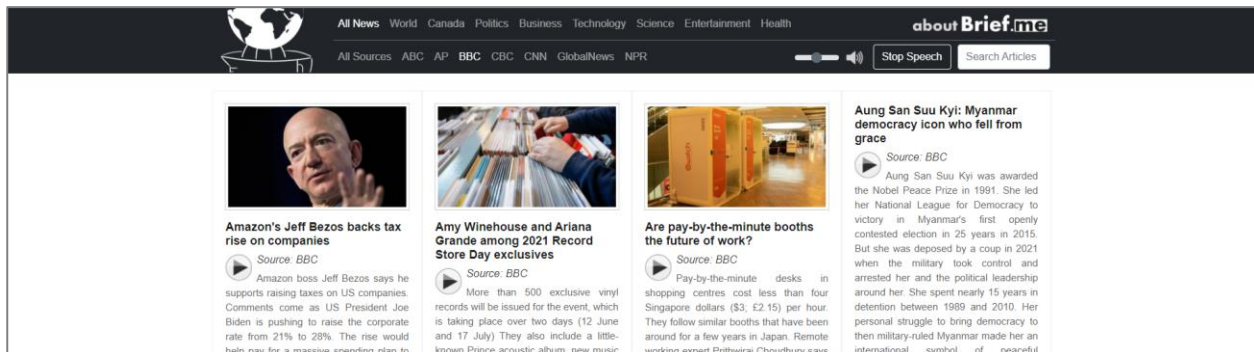


Figure 9: BriefMe – News Articles by Category (Canada)

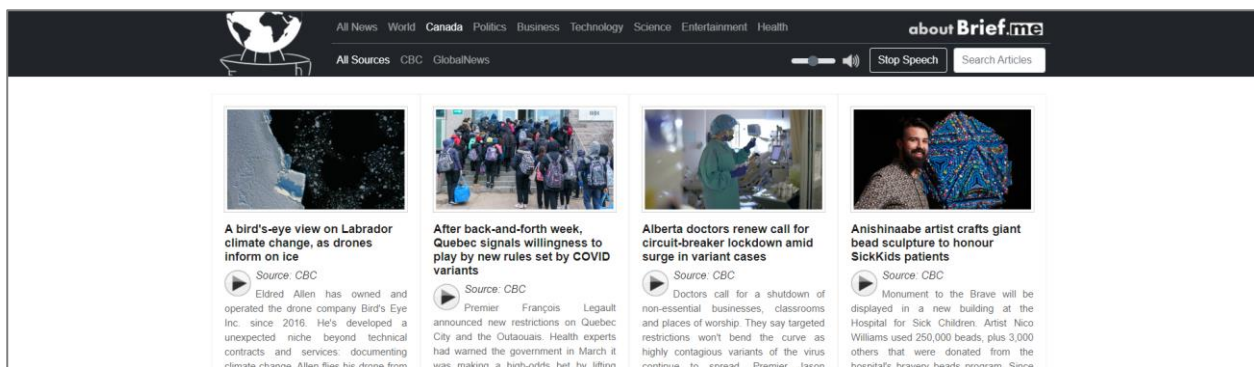
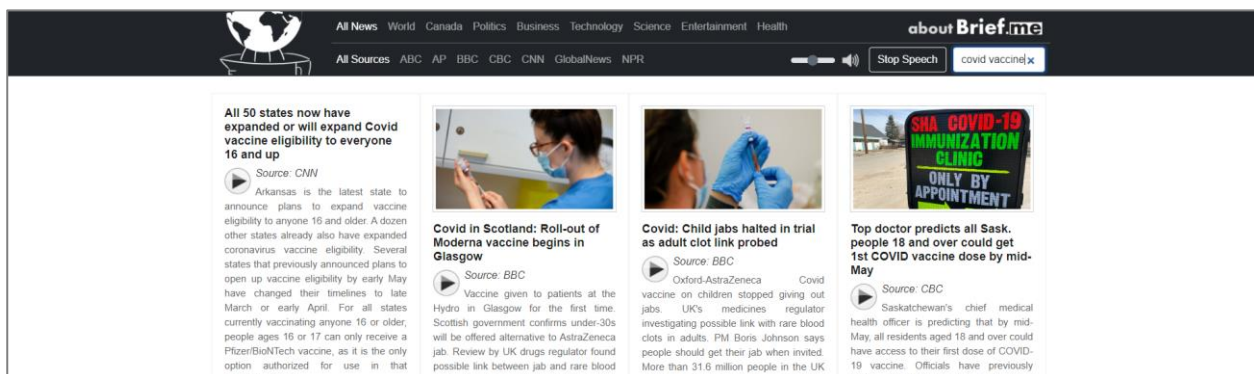


Figure 10: BriefMe – News Articles Search (Covid Vaccines)



Article Interface

For each article, we present an interface that includes a headline, a summary, a link to the source, and a picture. By displaying all the summaries on one page, we reduce the time to go back and forth between articles like traditional news browsing. Additionally, the hyperlink to the full article on the respective news outlet's official website supports readers who want to read particular articles in more in detail. We believe that news outlets can also benefit from this application, by having new traffic coming to the pages for their presented articles.

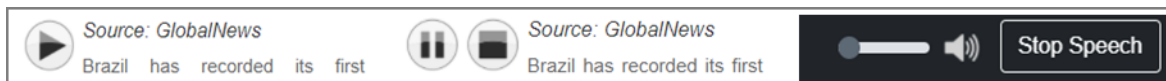
Figure 11: Article Interface



Speech Synthesis

To further aid users in accessing news quickly, we added a Speech Synthesis functionality to the application, using browsers' native speechSynthesis functionalities to generate text-to-speech output. Articles can be played individually, and controls to pause/stop playing articles or change the volume were added to support user preferences.

Figure 12: Speech Synthesis Controls

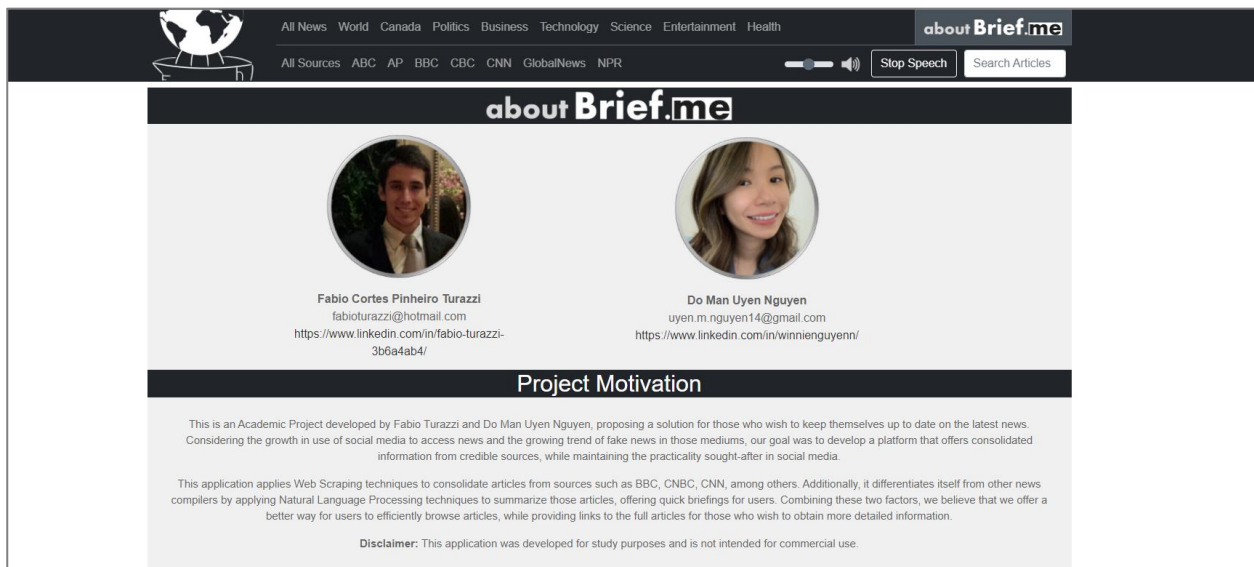


About Page

Lastly, considering the academic nature of this project and our intention to include it as a demonstration in our portfolio, we developed an about page to inform users about our development process and challenges faced. The page provides a brief summary of the following topics:

- **Personal Information:** Information about the developers;
- **Project Motivation:** Description of problem statement and the potential value of the solution;
- **Technologies Used:** List of technologies and libraries used for development;
- **Development Information:** Explanation of some important aspects of our development process for outside users.

Figure 13: About Page

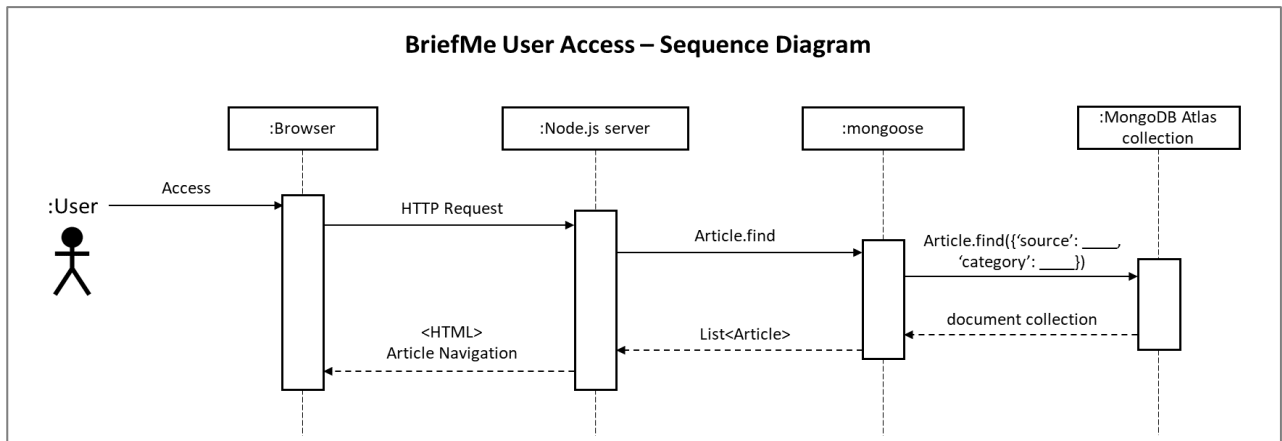


4.3. Software Architecture

The architecture of our system has some particularities when considering traditional development methodologies since it relies on high-processing algorithms to perform web scraping and text summarization. The database loading logic done by those algorithms is completely separated from the main program, contained in a Python program that runs systematically in the background. The main Web Application is deployed on an Express server, using Node.js, and has a simpler implementation focused solely on browsing the summarized articles. This flow is illustrated on the following UML Sequence Diagrams:

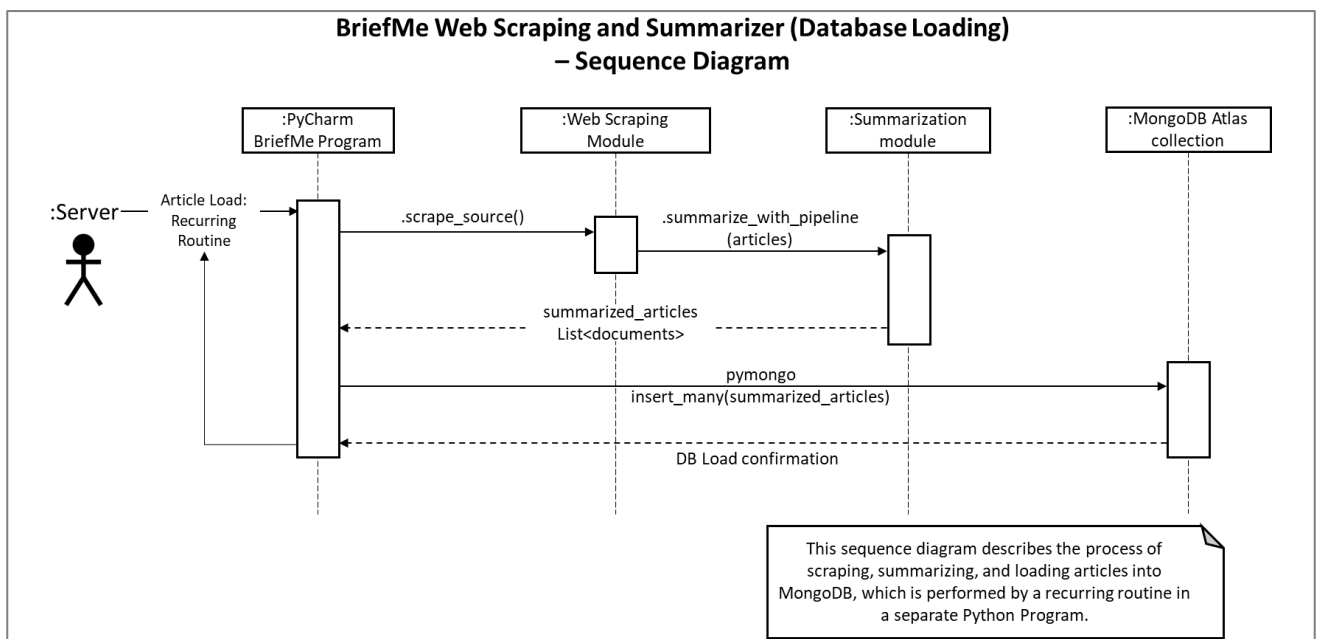
4.3.1. BriefMe – Sequence Diagrams

Figure 14: User Access Sequence Diagram



- The User Access diagram above describes the logic for accessing the Web Application, with the browser sending a signal to the Node.js server, which in turn uses mongoose to query the MongoDB collection of articles. Lastly, this information is loaded and displayed for the user.

Figure 15: Database Loading Sequence Diagram

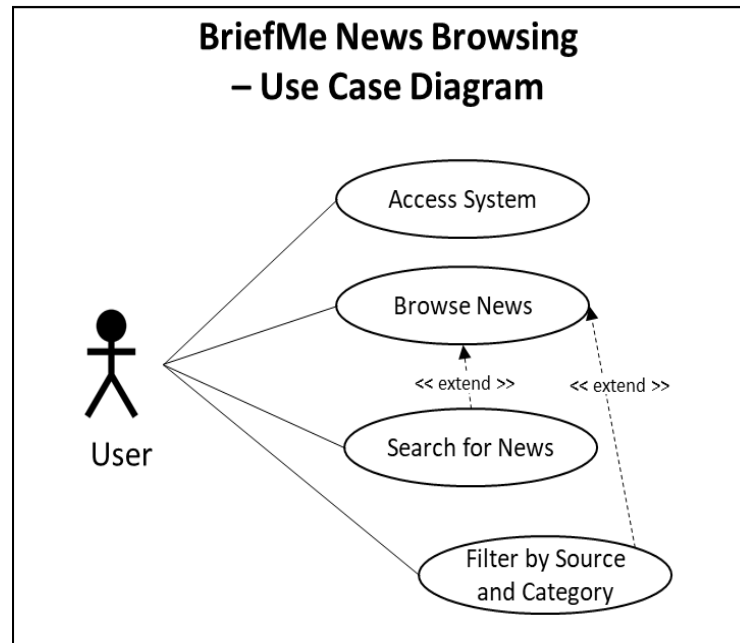


- The Database Loading diagram describes how this information is loaded on the MongoDB collection, with the PyCharm program running its Web Scraping and then Summarization modules, inserting the resulting document collection in MongoDB.

4.3.2. BriefMe – Use Case Diagram

The following diagram describes the Use Case for news browsing performed by users inside the system. It includes alternative paths for filtering or searching for specific news, categories, and sources.

Figure 16: News Browsing Use Case



Use Case Description

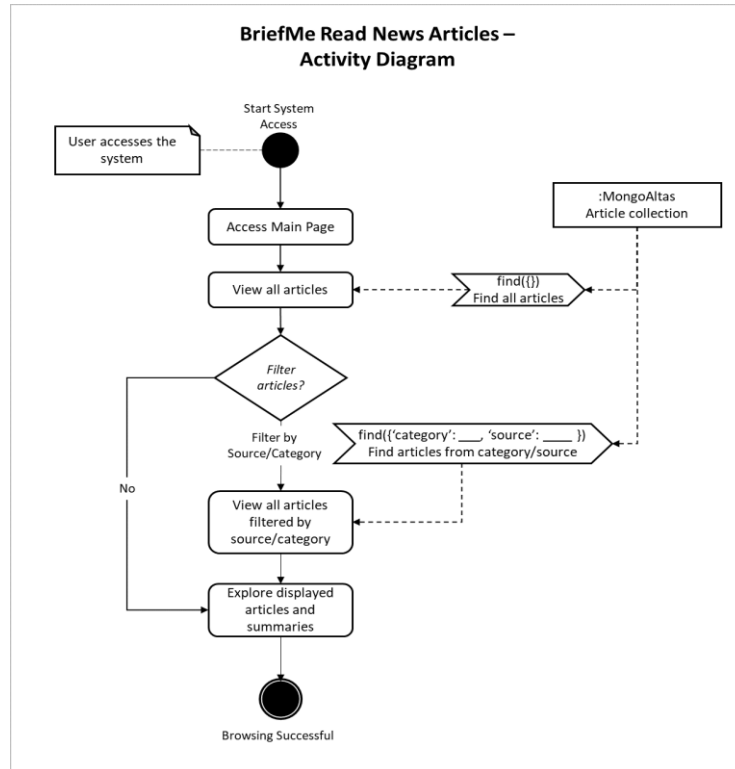
BriefMe News Browsing

- **Primary Actors:** User.
- **Preconditions:** The system has been loaded with a summarized article by the recurring Python Script.
- **Postcondition:** User browsed articles as desired.
- **Primary Path:** User accesses the system, reads articles and summaries, and navigates to any news websites he desires by clicking each summary.
- **Alternate Path:** User applies filters for news sources and categories, searches for specific news summaries, and browses the results.

4.3.3. BriefMe – Activity Diagram

The activity diagram describes the sequential steps performed by a user when accessing the system and browsing news:

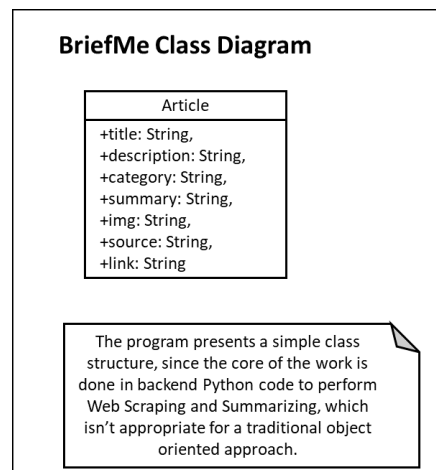
Figure 17: Read News Articles Activity Diagram



4.3.4. BriefMe – Class Diagram

Since most of the heavy work of the application is done inside our Python program, it uses Python objects with a more data-oriented approach rather than traditional object-oriented models. For the web application, we then use a simple Article model to access our data, as seen in the class diagram below:

Figure 17: BriefMe Class Diagram



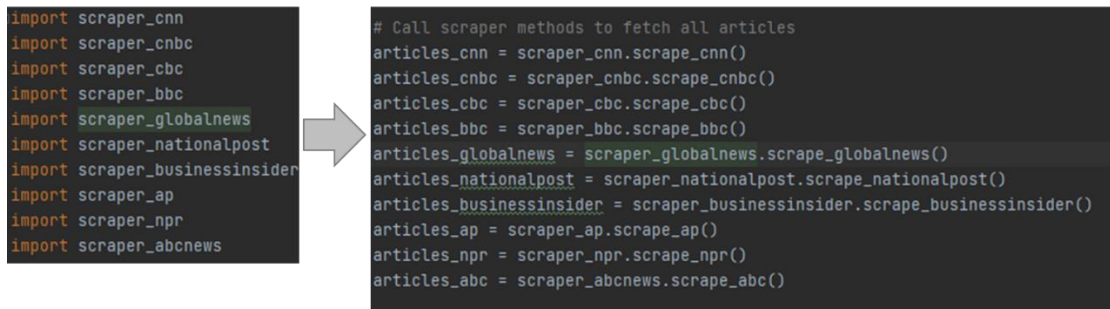
4.4. Design Patterns Applied

When describing the design pattern in our application, we must consider the separate logic for the Web Application and the Python Database Loader. Those two different programs use different logic between them to handle objects.

4.4.1. Python DB Loader – Design Pattern

In the database loader program, we use Python objects to move data around, specifically using arrays of dictionaries containing all properties from articles that we intend to load into MongoDB. To modularize this implementation, we use similar logic to Strategy Design Pattern, by separating the codes for web scrapping each source, and for summarizing the complete code, inside separate Python files. This allowed us to keep a large amount of code used for scraping and summarizing in separate files, resulting in better code management and maintenance.

Figure 11: Importing Web Scrapping Modules to compile articles



```
import scraper_cnn
import scraper_cnbc
import scraper_cbc
import scraper_bbc
import scraper_globalnews
import scraper_nationalpost
import scraper_businessinsider
import scraper_ap
import scraper_npr
import scraper_abnews
```

```
# Call scraper methods to fetch all articles
articles_cnn = scraper_cnn.scrape_cnn()
articles_cnbc = scraper_cnbc.scrape_cnbc()
articles_cbc = scraper_cbc.scrape_cbc()
articles_bbc = scraper_bbc.scrape_bbc()
articles_globalnews = scraper_globalnews.scrape_globalnews()
articles_nationalpost = scraper_nationalpost.scrape_nationalpost()
articles_businessinsider = scraper_businessinsider.scrape_businessinsider()
articles_ap = scraper_ap.scrape_ap()
articles_npr = scraper_npr.scrape_npr()
articles_abnews = scraper_abnews.scrape_abnews()
```

The main.py execution file calls each of the modules using their scrape_source() methods, and compiles the results inside arrays of documents.

Figure 12: Passing Articles into Summarization Module and Loading to MongoDB

```
import summarize_text

#Loop through the array of arrays, containing sets of articles from each source
for articles in article_all_sources:

    print("Summarizing: ", articles[0]['source'])
    #Assign array of dictionaries to a dataframe
    df = pd.DataFrame.from_dict(articles)
    #Remove duplicates and NaN values
    df = df.drop_duplicates()
    df = df.dropna(subset=['description'])
    df = df.dropna(subset=['title'])

    #Dropping articles with less than 200
    df = df[df['description'].map(len) > 200]

    #Apply summarizer to the article
    df['summary'] = df['description'].apply(lambda x: summarize_text.summarize_with_pipeline(x))

    #Delete previous articles from that source and add new articles
    clear = mycol.delete_many({"source": articles[0]['source']})
    repopulate = mycol.insert_many(df.to_dict('records'))
```

The arrays of documents are then passed through the summarization module, gaining a new property related to the actual summary, and lastly, the information is loaded into MongoDB.

4.4.2. BriefMe Web Application – Design Pattern

As we discussed previously, the Web Application itself has a simpler implementation due to the heavy work being performed by the Python scripts in the back-end. The application is based on a Factory Method Design Pattern, with our object creation being centralized inside the routes.js file. We implemented this logic since a large volume of article objects had to be created systematically, and their information always came directly loaded from the MongoDB collection of articles that the Python program creates.

This was done by first implementing a Mongoose model schema for articles, included in the models/article.js file:

Figure 13: Mongoose Model Schema

```
const mongoose = require('mongoose');

// Define schema
const Schema = mongoose.Schema;

const ArticleSchema = new Schema({
  title: String,
  // description: String,
  summary: String,
  category: String,
  img: String,
  source: String,
  link: String
});

const Article = mongoose.model('Article', ArticleSchema);

module.exports = Article;
```

The schema is then used whenever we query MongoDB using mongoose, automatically creating a list of article objects which we can freely display in our Web Application. This list can be seen in the variable “articles”, which is created in the callback function after querying MongoDB inside the routes/router.js file:

Figure 14: Creating Article Objects using MongoDB and Mongoose Schema

```
//Router for home page of the application
router.route("/").get((req, res, next) => {
  // find all articles for a given category
  // Article.find({})
  Article.find({}).select({ "title": 1, "category": 1, "summary": 1, "img": 1, "source": 1, "link": 1, "_id": 0 })
    .then((articles) => {
      //Render home page with data from mongodb
      res.render('index', {articleData: articles, appData: dataJson});
      next();
    })
    .catch((err) => res.status(400).json("Error: " + err));
});
```

The code above also displays the use of the Design Pattern of Immediately Invoked Function Expressions (IIFE), in which we use arrow functions to declare and invoke our callback functions at the same time. This is common in Node.js implementations and was included to allow our callback functions to execute the server codes within their logical order, which would otherwise be fully asynchronously.

4.5. Development Process Tasks

The tasks from this development process were divided between the two developers, enforcing equal participation between group members. The following table displays activities with their corresponding estimated hours, responsible developer, and completion date:

Activity	Description	Est. Hours	Resp.	Completion Date
Develop initial web scraping algorithm	Develop scraping algorithm using scrapy or beautifulsoup library	10	Fabio	2021-01-18
Develop text summarizing algorithm	Experiment with different text summarization algorithms for initial implementation	20	Winnie	2021-01-22
Implement scraping with search filters	Apply improvements in web scraping algorithm to allow search for specific filters	10	Winnie	2021-01-25
Implement back-end with node.js and Python integration	Implement express server back-end, which will receive filters and run python code	15	Fabio	2021-01-25
Integrate front-end and back-end of application	Integrate with server to send requests for search filters and receive/display the filtered and summarized articles	10	Fabio	2021-01-25
Develop front-end page with news and search filters	Page with news articles and search filters	10	Fabio	2021-02-03
Adapt scraping algorithm for 10 different websites	Adapt scraping algorithm to contemplate different news sources	20	Fabio Winnie	2021-02-20
Integrate scraping code with recurring logic	Add a recurring logic to execute the algorithms periodically	10	Fabio	2021-02-22
Prepare midterm progress paper and video	Write presentation paper describing work progress	6	Fabio Winnie	2021-02-25
Implement improvements in summarization algorithm	Experiment and implement new summarization model	20	Winnie	2021-03-07
Integrate Python scraping and summarizing scripts and test results	Integrate infrastructure to the scraping and summarizing algorithms developed and test the complete process	5	Fabio Winnie	2021-03-10
Test algorithms and identify improvements or bugs	Testing stage to identify improvements and bugs	10	Fabio Winnie	2021-03-30
Implement code improvements and polish the final application	Apply necessary improvements. If no improvements are identified, support to more websites will be added	15	Fabio Winnie	2021-04-01
Implement speech synthesis functionality	Add text-to-speech features and generate controls for users	5	Fabio	2021-04-06
Implement "About" page of the application	Page summarizing relevant information about the project	5	Winnie	2021-04-07
Prepare final presentation paper and video	Write presentation paper describing complete work	6	Fabio Winnie	2021-04-10

4.6. Project Testing

To ensure our application is working correctly and satisfies the goals stated for this project, a set of tests was performed for each module of the application. The tests and their results will be described in this section of the report.

4.6.1. Web Scraping Module Testing

The focus of our tests on the Web Scraping module was to ensure the completeness of the scraped articles and the correct access of article features, such as title, image, text, and source. After this module was developed, we started monitoring an indicator to measure if these aspects were adequately collected for our application.

- **Measured Indicator: Percentage of Relevant News Identified** – This indicator measures if news articles are being successfully found. The measurement of this indicator involves selecting a sample of portals and filters and calculating the percentage of relevant articles correctly identified by our program.

Since the first measurement (see Appendix B: Progress Report 1), 100% of articles in our sample were correctly identified and loaded by our algorithm, a result maintained for our two subsequent measures (Reports 2 and 3). Considering the consistency in formatting of the news sources websites, quality assurance for this module was easy to satisfy.

4.6.2. Summarization Module Testing

This was the section of the application that received most of our efforts during testing. To ensure the quality of the summaries, we experimented with several different Natural Language Processing algorithms and assessed the accuracy of the summaries generated by each of them. During each round of tests, we assessed the quality of the summaries using the following metric:

- **Quality of Summaries** –The measurement involved one member selecting a sample of articles, summarizing them with the algorithm. The member then evaluated the summaries based on 3 criteria: accuracy (how consistent the information in comparison with the original article), coherence (how easy and consistent the summaries for human to comprehend), and coverage (how many important pieces of information were delivered). The process was repeated by the second member, and the average scores were recorded for further assessment.

Initial Exploration

In an initial testing round, we assessed the algorithms Text-To-Text Transfer Transformer(T5), Transformer pipeline API, and GPT-2. In this first test, we limited the minimum and maximum words to 100 and 200 respectively. During this first assessment, all initial versions of our models scored below our accuracy targets, with performing model having an approximate 50% score (see Appendix B: Progress Report 1). This motivated us to overhaul the summarization module of the application, evaluating new models and tweaking the existing ones.

Final Model Evaluation

For our second assessment, we analyzed the performance the models BART, retrained BART (alternative version trained on a different subset of articles), Pipeline API (adding text truncation), and GPT-2 (with some adjustments to training). We used a sample of 5 articles of different lengths to perform the evaluation, which rendered the results recorded on the table below:

Table 1: Results of Summary Algorithms

	Accuracy (0.3)	Coherence (0.3)	Coverage (0.3)	Runtime* (0.1)	Weighted average
Pipeline API	5/5	4.5/5	3.5/5	~ 17 sec ea (2/5)	0.82
BART	5/5	4.75/5	4.5/5	~8 sec ea (3.5/5)	0.92
GPT-2	2.5/5	3.5/5	3/5	~ 17 sec ea (2/5)	0.58
BART Retrained	5/5	3.5/5	3/5	~(1200/no. articles + 1) sec (5/5)	0.79

*Runtime varies based on the GPUs used.

Considering the results, we selected the BART model as our best performing candidate. This rendered a score of approximately 90% (see Appendix C: Progress Report 2), satisfying our target for this project.

4.6.3. Web Application Module Testing

Since the bulk of the application logic is included in the Scraping and Summarization modules, our Web Application assessment was performed with simple Black-Box Testing. For this task, the two developers performed the following set of tests:

- **Browsing News & Search Feature:**
 - Access articles by source, category, and search form.
 - Verify that the correct articles were displayed after each filter was applied.
- **Navigation:**
 - Source and category navigation assessed in the previous step.
 - Verify the correct navigation to home and about buttons of the website.
- **Speech Synthesis:**
 - Access text-to-speech for multiple articles in different pages.
 - Verify that the correct text is being played.
 - Verify the correct behavior of play, pause, stop, and volume controls.
 - Repeat steps for different browsers (Google Chrome, Mozilla Firefox, Internet Explorer).

Assessing User Interface (UI)

While the Black-Box tests assessed website functionality, we also implemented a test to verify that the UI is user-friendly. To perform this test, we asked for 5 parties not involved in the development to access the deployed website and freely navigate between the sections of the application. This was done without providing any context beforehand, other than the main abstract concept of the application.

The results of this assessment were positive, rendering good feedback and no questions about the structure from the students that attempted it. This result was expected, given the simplicity of the Web Application itself.

4.7. Using the Application – Local Environment Setup and Web Access

This section will describe how to set up the environment and run the application on a new computer. Additionally, we will provide details on how to access our application via the web, considering it is currently deployed using Heroku.

4.7.1. Local Environment Setup

The environment setup from our system involves running its two programs: the PyCharm (Python) program CSIS4495_BriefMe_DbLoader and the Node.js program CSIS4495_BriefMe. The step-by-step instructions are detailed below:

CSIS4495_BriefMe_DbLoader (Python)

1. Download and install PyCharm and Python 3;
2. Extract the program CSIS4495_BriefMe_DbLoader to a directory of your choice and open it in PyCharm (File > Open > select directory).
3. Inside PyCharm's Terminal tab, install Python dependencies by running the following code:
 - a. pip install torch
 - b. pip install transformers
 - c. pip install dnspython
 - d. pip install pymongo
 - e. pip install pandas
 - f. pip install bs4
 - g. pip install requests_html
4. Run the program's main file "main.py".
5. This program will run recurringly, updating the database every time an execution cycle completes.

CSIS4495_BriefMe (Node.js)

1. Extract the program CSIS4495_BriefMe to a directory of your choice and open it in Visual Studio Code (File > Open Folder > select directory).
2. Right-click the folder inside VS Code and select "Open in integrated terminal".
3. Run the command:
 - a. node server.js
4. Access the WebApp by navigating to the address localhost:3000 in your browser.

After completing these two sets of tasks, the web application server will be set up on the machine's localhost (port 3000) and can be accessed at any time while the server is still running. Additionally, the PyCharm program will run indefinitely to maintain the application data updated. In deployment, these two tasks would be running on the server concomitantly.

4.7.2. Accessing the Deployed Web Application

In order to provide easier access for demonstration, we have deployed the application to Heroku. To access the latest deployed version, navigate to the following link on any internet browser: <https://briefmenews.herokuapp.com/>

4.8. Project Delivery: Directory Description

The complete code for this application can be accessed in the following folders of this submission:

Final Report

- CSIS4495_BriefMe_DbLoader
 - Python program that runs main.py recurringly to extract and summarize.
 - GitHub Page: <https://github.com/fabioturazzidouglas/BriefMe>
- CSIS4495_BriefMe
 - Node.js server that renders the Web Application.
 - GitHub Page: <https://github.com/fabioturazzidouglas/BriefMeDbLoader>

Since the database is generated by the Python program, no additional database creation scripts are necessary to load information before using the application.

5. Appendixes:

Additional documentation from the development process is included in the following documents on this submission folder:

- Appendix A: Project Proposal
- Appendix B: Project Progress Report 1
- Appendix C: Project Progress Report 2
- Appendix D: Project Progress Report 3

6. References:

Haile, Tony. "Chartbeat CEO Tony Haile: What You Get Wrong about the Internet." *Time*, Time, 9 Mar. 2014, time.com/12933/what-you-think-you-know-about-the-web-is-wrong/.

Suciu, Peter. "More Americans Are Getting Their News From Social Media." *Forbes*, Forbes Magazine, 11 Oct. 2019, www.forbes.com/sites/petersuciu/2019/10/11/more-americans-are-getting-their-news-from-social-media/?sh=468c7e913e17.

Transformers Library Documentation: <https://huggingface.co/transformers/>

Requests-html Python Documentation: <https://docs.python-requests.org/projects/requests-html/en/latest/>

BeautifulSoup Library Documentation: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

SpeechSynthesis Documentation: <https://developer.mozilla.org/en-US/docs/Web/API/SpeechSynthesis>

Pandas Library Documentation: <https://pandas.pydata.org/docs/>

Express Documentation: <https://devdocs.io/express/>, <http://expressjs.com/en/api.html>

Node.js Documentation: <https://nodejs.org/en/docs/>

JQuery Documentation: <https://jquery.com/>