UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

News search by region and visualisation on a map

Author:
Mykyta SAMOVAROV

Supervisor:
Dmytro PRYIMAK

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

I, Mykyta SAMOVAROV, declare that this thesis titled, "News search by region and visualisation on a map" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

News search by region and visualisation on a map

by Mykyta SAMOVAROV

Abstract

Nowadays, people are flooded with news - many news providers constantly create articles. For a person who wants to read about a specific region in a country, it will be tough to segment the news by region manually. This project aims to segment Ukrainian news by region and project it into a website with an interactive map. For usability, users would also have the ability to select a specific period to get even more segmented data. In order to achieve this, it would require writing the whole workflow: from writing connectors to each news provider, creating a dataset, searching for keywords, storing article data, aggregating it and finally projecting data on a map.

Here is a link to the GitHub repository: link.
Here is a link to the website: link.

Acknowledgements

I would like to thank my supervisor Dmytro PRYIMAK for his help in this project and our collaboration in sync-up driven development.

I also want to thank all Ukrainian guardians for their service in defending territory from the enemy.

Contents

List of Figures

List of Tables

List of Abbreviations

API	**A**pplication **P**rogramming **I**nterface
JS	**J**ava**S**cript
JSON	**J**ava**S**cript **O**bject **N**otation
HTML	**H**yper**T**ext **M**arkup **L**anguage
REST	**RE**presentational **S**tate **T**ransfer
URL	**U**niform **R**esource **L**ocator
UI	**U**ser **I**nterface

Dedicated to someone who makes me wake up every day and enjoy my life. Someone who never betrayed me and always leads out of uncertainty to understanding.

Chapter 1

Introduction

1.1 Problem

The usual practice of getting information about what is happening in the world today is reading a list of article headlines, sometimes with an image and tapping on them if interested.

However, there is a problem with searching for something specific for people who do not have time to read through every headline. When reading an article, the first question is 'Where does it happen?'. The reader is interested in how far the event happened, or maybe he or she wants to search for the news in a specific region.

The problem with local news of the desired region is, in most cases, their speed and correctness. Events are published with a delay and sometimes contain spelling mistakes. Article tags in global news are subjective if labelled by an editor (or even forgotten). Another problem is that a person who may not know the name of a small village where the event happens or by duplication of names could think of a place in another part of a country.

1.2 Goal

The main goal of this project is to create a website that would allow its users to see the news more conveniently. It would be a visual map that divides the country by regions and contain news for each separate region. Also, a time field will filter news by time.

This website will require a robust API that processes articles and stores them in a database.

1.3 Constraints

In a place where this project interacts with the user - the UI part should be minimal and intuitive. So, there should be only a map and a period selector. When the user clicks on region, he or she should see a list of hyperlinked article headlines.

The following constraint is the number of news providers. It is limited because it requires writing a script to extract each web page type from different news providers.

Another constraint is refresh time. To not be banned as a bot and not get the same pages a dozen times, the script should wait some time for a new article to appear.

Lastly, and most importantly, news to region mapping correctness. It will be a custom algorithm and an amateur database of keywords. It will not guarantee 100% correctness. For controversial cases, there should be a special flag.

Chapter 2

Related Works

2.1 Live Universal Awareness Map

““Live Universal Awareness Map"(“Liveuamap”) is a leading independent global news and information site dedicated to factual reporting of a variety of important topics, including conflicts, human rights issues, protests, terrorism, weapons deployment, health matters, natural disasters, and weather-related stories, among others, from a vast array of sources.” [4]

This service was started in 2014 by two Ukrainians. Currently, the whole project suggests maps for Ukraine, the USA, Russia and Syria and other countries. Their website is well-developed: there is an option for a custom legend, language, and time.

Liveuamap has its API, a website that uses it and a team of managers that manually approve news. Website scrapes Ukrainian data from Twitter and Telegram sources only.

The NewsMap solution loses in suggested maps to users and in the number of features that the map provides.

However, NewsMap may win in two cases. The first win, the news will appear faster on the map than in Liveuamap because first does not require human interaction. The second win is news variety: script scrapes news sources from websites but not Telegram channels.

Chapter 3

Backend

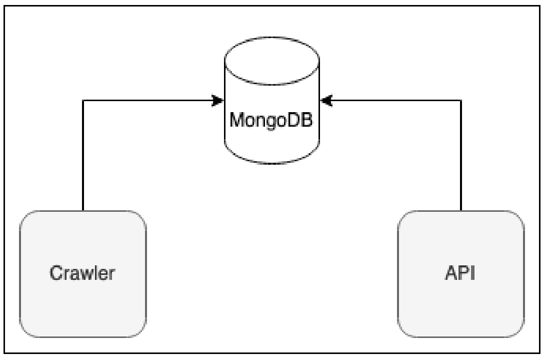

FIGURE 3.1: Backend structure: Crawler, MongoDB, API

As a backend, there are three parts:

- **Crawler** - An automated script that goes every N minutes over news provider websites and sends processed articles to the database.
- **MongoDB** - Non-relational database containing processed articles and last ids of news per news provider.
- **API** - FastApi tool that aggregates articles from MongoDB and sends them to Web UI.

Why Python

This project, apart from the web part, is written in Python. This programming language has the advantage of many libraries for different purposes. Here used libraries are: requests, BeautifulSoup, pymongo, and fastapi.

3.1 Crawler

Crawler is an automated script that iterates over news providers, checks if new articles were already added and if not, then collects them into a database.

3.1.1 General algorithm

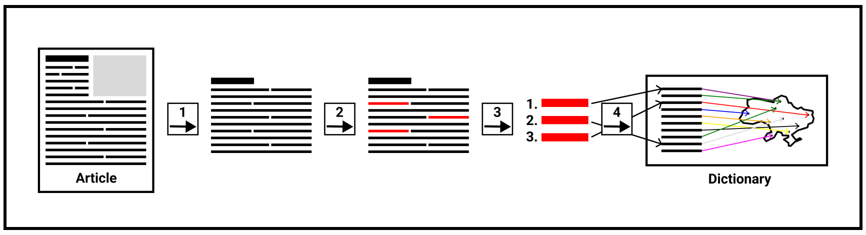

FIGURE 3.2: Crawler algorithm: from raw page to region list

Crawler's primary goal is to extract regions from an article. There is a raw page in the input. Then script pulls from it only text. After that, the program searches for the keywords in the geographical location dictionary, including only cities or villages, and deletes other text. Finally, script maps found words with the dictionary of word to region relation.

3.1.2 Classes

All class methods use an approach of class methods, which allows not to initialise variables but instead logically separate them.

Database

Database class contains a number of useful methods:

Database class	
Name of method	Explanation
get_client	A method that connects to Mongo client and returns it.
get_collection	A method that returns a specific collection from Mongo client.
load_documents	A method that loads a number of documents into a specific collection.
get_news_providers_data	A method that gets data from NewsProviders collection.
update_news_providers_ids	A method that sets new ids to the specific news provider.

TABLE 3.1: List of methods in Database class

Since there is only one instance of a database, this class is not inherited.

News provider

A news provider is an abstraction of a website that posts news articles and has a web page with the most recent news articles.

It is worth mentioning what the **id** and the **last processed id** are.

The id is the unique string of every article.

The last processed id is the id of the last processed article. **NewsProviders** Mongo collection contains a list of last processed ids.

A news provider may publish links to more than one article website. For example website "Pravda" streams articles from itself, "LifePravda", "EPravda", and others. Because of this, there is a mapping of a link to an article.

The Python NewsProvider class contains the following methods:

NewsProvider class	
Name of method	Explanation
fetch_new_links	This method takes as an argument list of the last processed ids. After that, it requests a web page of the most recent news articles and returns links only to those articles that were not processed yet.
identify_article	A method takes a link to a web article as an argument and returns its Python class representation.
process	A general method for full news provider pipeline: Request to get new article links, iteration over them and processing one by one.
form_last_article_ids	A method that pushes new article ids to old ones and cuts them by limit.
extract_new_ids	A method that returns only those ids that were not processed yet.
get_links_to_download	A method that returns links from ids.
links_to_ids	A method that converts a link to id.
get_all_links	A method that requests news provider web pages of most recent articles and returns links. This is the only method that is custom to every news provider.

TABLE 3.2: List of methods in NewsProvider class

This class also have class variables:

NewsProvider class	
Name of class variable	Explanation
LINK_TO_ALL_ARTICLES	A link to a web page that contains a list of most recent news articles.
LINK_TO_CLASS_MAPPING	A dictionary that has a mapping from article URL to article class.
BASE_ARTICLE_CLASS	A general article object of news provider. It generates an id from the link.

TABLE 3.3: List of class variables in NewsProvider class

Every new news provider class will use NewsProvider class methods and class variables.

Article

An article abstraction is an interface for how to process an article from the same website.

In order to parse raw web pages easily, the script uses a library called BeautifulSoup. It parses HTML page into its abstractions.

Script also uses the Datetime module and its format to keep articles publish dates in the same type and for sorting purposes in future. When sorting dates, it is much faster to convert them to UNIX timestamp - the number of seconds since 1970. The basic article class contains the following methods:

Article class	
Name of method	Explanation
get_beautiful_page	Scrape the raw page and transform it into a BeautifulSoup object.
extract_date_published	A method that extracts in-article date representation.
convert_date	A method that converts in-article date representation into commonly used Datetime format.
extract_title	A method that extracts article title from BS representation.
extract_text_body	A method that extracts article body from BS representation.
extract_tags	A method that extracts article tags if they exist from BS representation.
get_page_data	A method extracts the date, title, and body from the article.
decompose_page_by_kw	This method decomposes page contents into “places” keywords via kwsearcher and analyser.
link_to_id	A method that extracts page id from page link.
to_json	A method that converts page content into dictionary type to put this data into a database.

TABLE 3.4: List of methods in Article class

Article class also contains next class variables:

Article class	
Name of class variable	Explanation
NEWS_PROVIDER_NAME	The name of news provider to which the article belongs.
ARTICLE_TYPE	The name of website that is part of news provider.
LANGUAGE	The language of the article.

TABLE 3.5: List of class variable in Article class

Every kind of article in the news provider inherits and overrides methods and class variables from the base Article class.

3.1.3 Word dictionary

In the Ukrainian language, nouns have different endings depending on the case of the word. Due to this, word dictionaries have become much larger.

Dictionary types

A dictionary is a set of community names. There are four different types of them. One for kwsearcherv1 and three for kwsearcherv2.

word_list_v1.json

Single dictionary used for *kwsearcherv1*. Schema is:

```
{
    "OBLAST_NAME": {           // Name of oblast in Ukraine.
        "misto": List[str],    // Cities of this oblast.
        "rayon": List[str],    // Districts of this oblast.
        "selo": List[str],      // Villages of this oblast.
        "general": List[str], // Other names that refer
                               // to this oblast.
        }
}
```

all_words.json

Dictionary used for *kwsearcherv2*. Schema is:

```
{
    "words": List[str]   // All community names
}                        // in lower case in all cases.
```

word_info.json

Dictionary used for *kwsearcherv2*. Schema is:

```
{
    "COMMUNITY_NAME": List[object]  // Name of community
}                                  // capitalised in general case.
```

Object structure is:

```
{
    "oblast": str,   // An integer in string form.
                     // It shows to what region this name belongs.
    "type": str      // Type of community name:
}                    // "selo", "rayon", "misto", "general"
```

The reason for using a list of objects is because of duplicating names across regions.

words_cases_to_main.json

Dictionary used for *kwsearcherv2*. Schema is:

```
{
    str: List[str]
}
```

Here the key is community name in any form. Value is the name in the general case. It is in a list because different community names may be equal in some cases.

Dictionary sources

General cases of community names were taken from UkraineCitiesAndVillages open repository in GitHub from user Adushar. Every object in the source JSON file had irrelevant data, like *level_** and *object_code*.

Word cases were taken from dict_uk open repository in GitHub from user brown-uk. Unfortunately, this repository is more about all Ukrainian word cases, not community names specific. That is why it contains much unneeded data and not all community names.

Another part of word cases collected from lcorp.ulif.org.ua website. Under the hood, this website uses data from the “Ukraine Dictionaries online” CD version. Unfortunately, the CD version is unavailable on the web. The website also is quite old and does not use requests to get source data.

The only way to extract data from the lcorp website was to search every word manually. Fortunately, a tool called Selenium exists. This service’s primary purpose is browser automation. It also has a Python library that automates some simple browser operations.

With Selenium second part of cities was found and a big part of villages. There are approximately 270 cities and 30000 villages in the Adushar JSON file. At the moment, all cities' names cases exist in the dictionary. Village names repeat, and out of 17000 unique names, more than 15000 exist now. Other 2000 village names are searched only in the general case.

3.1.4 Keyword searcher

Keyword searcher (kwsearcher) is functionality that searches for community names inside raw text. There are kwsearcherv1 and kwsearcherv2.

Keyword searcher v1

This script uses *word_list_v1.json* dictionary. It is inefficient, finds names only in the general case, and skips all villages' names.

The algorithm is the following:

1. Join title and body via new line.
2. Filter only those words that start from a capital letter and clear them from text symbols like comma, dot, and quotation symbols.
3. Join the capital words into one sentence by space.
4. Go over each word inside the word dictionary and see if it appears in a filtered text.

Keyword searcher v2

This version uses *all_words.json*, *word_info.json*, and *words_cases_to_main.json* dictionaries by loading them once and using as a variable.

For better efficiency additional script called an analyser exists.

The algorithm of kwsearcherv2 is the following:

1. Join title and body via new line.
2. Lower the first letter after the new line, dot or quotation symbol. This will skip those words that are not names but are just written at the beginning of a sentence and therefore start from capital letters.
3. Filter only those words that start from a capital letter and clear them from text symbols like comma, dot, and quotation symbols.
4. Lowercase found words.
5. For every word that exists, add it to the filtered word list with the next word joined with a space. This will allow finding community names that consist of two words.
6. Convert the resulting list to a set and find an intersection with *all_words.json*. The result will contain words that are confirmed community names.
7. Next, for every found word, search a word in the general case for it via *word_cases_to_main.json*.
8. Finally, for every found word in the general case, add information about this word via *word_info.json* to returning list.

This script converts raw text into a list of found community names and information of them.

After this, data from the article title and body, along with article tags, if they exist, go through the analyser.

Analyser

The primary purpose of the analyser is to transform found community names and information about them into belonging to a specific region and confidence in it.

Currently, confidence is 0.5 for villages and 1 for cities.

An analyser goes over tags, and if it appears to be the name of a region, it adds it to the dictionary of region, confidence, and places.

The same algorithm processes words from kwsearcherv2 but does not check them since they are already processed.

It is essential to say that if an article had confidence in region 0.5 because of the found village and then a city in this region is found, then the overall confidence in the region is 1.

3.1.5 Crawler script

Crawler script file contains of two functions called *crawler* and *main*.

Main

Main is a function that never ends (in while True cycle). First, it gets the Mongo client and extracts news providers' names and last article ids from the NewsProviders collection. Then it executes crawler and gets processed article data and new last article ids from it. After that, it loads documents into Mongo collections. Finally, it halts for some amount of time.

Crawler

Crawler goes over each news provider with their last article ids. It executes the process method via already defined Python classes. Then it adds processed articles to the general list and adds updated last article ids to the defined news providers. Finally, it returns both variables.

3.2 MongoDB

3.2.1 Reason for choosing MongoDB

A non-relational database was selected for this project because of its speed and adaptivity to quickly changing schema.

3.2.2 Collections

NewsProviders

This collection contains data about sources where data is collected.

Schema for documents is:

```
{
  "_id": id,                        // id of document.
  "news_provider_name": str,        // A name of a news provider.
  "news_provider_ids": List[str]    // A list of N last article ids.
}
```

News providers may delete an article. Because of this database saves a list of the last articles ids.

Articles

This collection contains articles. Schema for documents is:

```
{
    "_id": id,                      // An id of document.
    "article_id": str,              // An id of an article in form
                                    // {news_provider_name}_{id}.
    "title": str,                   // A title of the article.
    "news_provider_name": str,      // A name of news provider.
    "article_type": str,            // A sub type of news provider.
    "link": str,                    // A link to the article.
    "time_published": str,      // The time when article is published.
    "published_timestamp": int,     // The time when article
                                    // is published in form timestamp.
    "time_collected": str,      // The time when article was “scraped”
    "text_language": str,           // The language of article.
    "tags": List[str],              // Tags of article.
    "regions": Object
}
```

where *"region"* is:

```
{
    str: {
        "confidence": float,  // A confidence belonging to region.
        "places": List[str]   // Locations found in article.
    }
}
```

Schema with *"regions"* is more complex. This project uses only 25 regions of Ukraine. Dictionary data type shows the association of article and region. Python dictionary has keys and values, where keys should be unique and hashable.

The script uses mapping from region name to its number in alphabetical order for memory efficiency.

Since MongoDB uses BSON, it does not allow for keys to be numbers. That is why in the current implementation, keys are integers cast to a string.

3.3 API

3.3.1 Reason for choosing REST

This architectural style perfectly fits fast and small applications.

3.3.2 Reason for choosing FastAPI

FastAPI is a modern Python library for building its REST APIs. It is lightweight, fast, has descriptive documentation and has a large community.

3.3.3 API paths

Index

- Type of request: *GET*
- Path: *"/"*
- Returns: *{"message": "It works!"}*

Developers use this path for debugging to understand that API is running.

Get total articles

- Type of request: *GET*
- Path: *"/articles/total/{from_time}/{to_time}"*
- Returns: *Dict[str, Dict[int, int]]*

Web UI uses this request to get total amount of articles per region filtered by period. It shows a number on a map under every region.

Calculation of this request completes on the side of MongoDB via aggregation. Aggregation in MongoDB is a series of grouping data, making operations over it and returning it. Aggregation pipeline is:

```
pipeline = [
    {"$match": {"published_timestamp": {"$gte": from_time,
                                        "$lte": to_time}}},
    {"$project": {"regions": {"$objectToArray": "$regions"}}},
    {"$unwind": "$regions"},
    {"$group": {"_id": "$regions.k", "count": {"$sum": 1}}},
]
```

First *$match* filters only those articles that were published in user-selected timestamp period. Next *$project* converts *"regions"* dict to List datatype. Next *$unwind* creates copies of article documents with separate values from unpacked *"regions"* list. Last *$group* maps region number to amount of articles where this number is found.

Validation

For this API path the only custom validation is that *from_time* should be less than *to_time* and *from_time* should be positive or zero.

Get articles by region

- Type of request: *POST*
- Path: *"/articles/"*
- Returns: *Dict[str, List[ArticleModel]]*

This request returns full articles documents filtered by period and region number. The request requires a limit and offset because returning all documents at a time is not memory-efficient.

UI displays articles in chunks. Every time user clicks an arrow for "next page", the web sends another request with an offset value.

Calculation of this request requires two MongoDB requests. The first request returns only article ids. The second request returns full article documents based on the ids found in the first request.

Aggregation pipeline is:

```
pipeline = [
    {"$match": {"published_timestamp": {"$gte": from_time,
                                        "$lte": to_time}}},
    {"$sort": {"published_timestamp": 1}},
    {"$project": {"regions": {"$objectToArray": "$regions"}}},
    {"$unwind": "$regions"},
    {"$match": {"regions.k": {"$eq": str(region)}}},
    {"$project": {"regions": 0}},
    {"$skip": offset},
    {"$limit": limit}
]
```

First *$match* filters only those articles that were published in user-selected timestamp period. Next *$sort* sorts articles by time published. Next *$project* converts *"regions"* dict to List datatype. After that *$unwind* creates copies of article documents with separate values from unpacked *"regions"* list. Next *$match* filters only those articles where region is as selected by user. Next *$project* removes "regions" data from every document, only id is needed. After that *$skip* alias to offset: skips amount of articles. Finally *$limit* limits number of returned documents.

The second request to MongoDB is the following:

```
collection.find({"_id": {"$in": [elm["_id"] for elm in result]}})
```

It searches for those articles that have next *id's*.

Validation

For this API path the validation is more complex. Apart from validating *from_time* and *and_time* a validation for limit and offset is needed. *limit* should be positive excluding zero (specifics of MongoDB) and *offset* should be positive including zero.

Chapter 4

Web UI

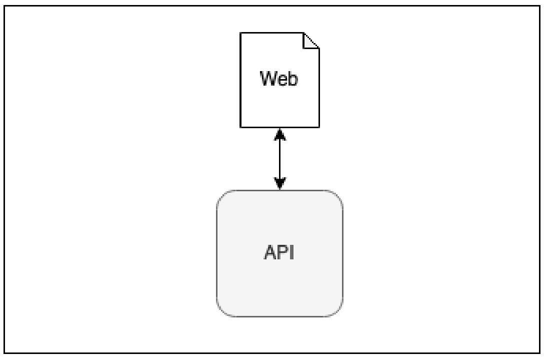

FIGURE 4.1: Web UI structure: Web, API

There are two parts for a Web UI:

1. API - FastAPI application described in backend part.
2. Web - Website built on React JS as an endpoint to interact with the user.

4.1 Why React JS

React JS is an open-source library with a component-style interface. This library is easy to understand and powerful in what it can do. React has a wide community, and most questions that arise already have answers in web.

4.2 Highcharts Map

Highchats JS is an extensive library for displaying interactive charts. It is highly configurable and smoothly integrates with React. This project needs a map visualisation, and for it Highchats provides a JSON map of Ukraine. Highchats JS library becomes a significant component in this project with these advantages.

4.3 Structure

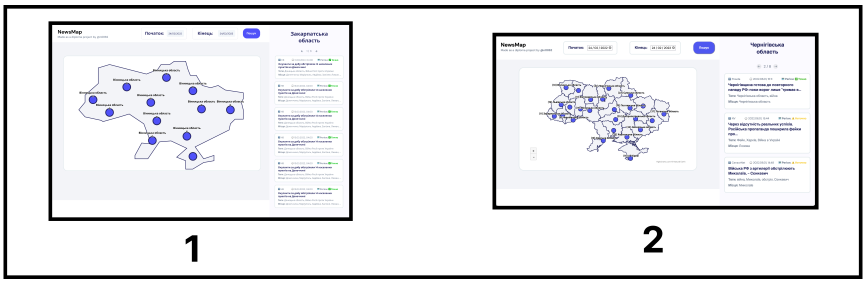

FIGURE 4.2: Web design: Expected and actual result

The website should have a minimal interactive design. It also should give an ability for the user to filter news by time and, most importantly, by region.

Articles with essential information appear when a user selects both variables.

A new tab with content should appear when the user clicks on the selected news article.

4.3.1 Main panel

The main panel takes up most of the screen and is always visible. It contains filters.

Top panel

The top panel shows the project's name, start date, end date and search button.

When the user clicks the search button, the system requests API and shows the number of articles for each region on a map.

Map container

The map container contains an only interactive map with regions and buttons. When a user filters the date and clicks on the preferred region, the system requests API and shows a piece of articles on the side panel.

4.3.2 Side panel

The side panel contains articles and buttons for switching between pages. Here page is a set of articles that fit a screen.

Region name with page buttons

Region name shows the selected region, two buttons to iterate between pages, and pages count.

Articles

Showing articles is the primary goal of this project. Information about them should be descriptive but not contain the article body. That is why the article component contains the next parameters:

1. Name of news provider.

2. Time when an article was published.
3. Confidence that article belongs to a selected region.
4. Article title.
5. Article tags if they exist.
6. Community names found in each article.

Chapter 5

Experiments

In this section, potential users express their thoughts about the website.

“I like the way how easy it separates news exactly by region. It looks comfortable, and I would use this website for my work. However, it is not clear for me what are all the list of news sources that provide articles.” - Taras, student of Journalism Studies.

“Overall good. The whole process requires no human interaction, and as you said, script checks for new articles every 5 minutes. I want to add that it will be better if you have more news providers, at least 5. You can also add local news to be more specific about separate regions.” - Roman, reads news daily.

“I would use it. Why not? The only request from me is to make all news in Ukrainian language and show only important news.” - Serhii, pensioner.

Chapter 6

Summary

From the technical part, this project required thinking about the whole product workflow: designing architecture, writing and rewriting algorithms for searching keywords, building a small and efficient database and thinking about how to present data to the user so that it would be easy to configure and use.

From the real value part, this project gathers multiple popular news providers into one place and segments news by region. Answers from the "Experiments" part showed that it appears useful and will save time when users want to check news from a specific region next time. Respondents suggested improvements that would make the website more clear and contain more articles.

6.1 What to improve and develop

This project has some things to improve. The "crawler" part needs the development of requests that bypass bot detection on news websites. This problem became vital during the production rollout. The database currently saves new article data quickly but aggregates data on output slowly. It means that writes and quick and reads are slow. This is not correct and should be vice versa. To solve this problem database should have either another schema or more specific region-dependent collections.

The "web" part would look better if "buttons" were under each city, not only region.

Some features can be developed, like adding new article sources, a mobile version of a website, and more filters on UI.

Bibliography

[1] *Datetime - Python library for managing datetime*. `https://docs.python.org/3/library/datetime.html`.

[2] *FastAPI - Python REST API framework*. `https://fastapi.tiangolo.com/`.

[3] *Highcharts JS - JavaScript library for visualisation*. `https://www.highcharts.com/docs/index`.

[4] *Liveuamap - An independent global news websiteAPI*. `https://liveuamap.com/about`.

[5] *Logging - Python library for logging*. `https://docs.python.org/3/library/logging.html`.

[6] *MongoDB - non-relational database*. `https://www.mongodb.com/docs/`.

[7] *React JS - JavaScript library for building UI*. `https://reactjs.org/docs/getting-started.html`.