НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
”___”_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра
спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»
на тему: «Формування пошукових запитів та архітектура взаємодії з системою пошуку наукових статей»

Виконав:
студент IV курсу, групи ТР-71
Журавльов Роман Вікторович _____

Керівник:
Асистент
Воронько М. П. _____

Консультант:
_____

Рецензент:
Доцент, к. ф. -м. н.,
Тарнавський Ю. А. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 122 «Комп’ютерні науки»

освітня програма «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2021р.

ЗАВДАННЯ
на дипломну роботу студенту
Журавльову Роману Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Формування пошукових запитів та архітектура взаємодії з системою пошуку наукових статей

керівник роботи Воронько Максим Платонович, асистент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”24” травня 2021р. № **1267-с**

2. Строк подання студентом роботи 04.06.2021р.
3. Вихідні дані до роботи
Мова програмування Typescript, фреймворк Next.js, середовище розробки VS Code
4. Зміст розрахунково-пояснювальної записки
Проаналізувати існуючі програмні рішення для пошуку наукових статей, описати програмну реалізацію системи, розробити графічний клієнт для взаємодії з системою, реалізувати уточнення фільтрів та критеріїв пошуку та відображення результатів пошуку.
5. Перелік ілюстративного матеріалу
Актуальність, Постановка задачі, Архітектура системи, Інтерфейс головної

сторінки, Пошукові фільтри, Каталог публікацій, Діаграма архітектури, Сторінка результатів пошукового запиту, Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання "_10_"_____жовтня_____ 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	15.10.2020	
2.	Вивчення та аналіз задачі	16.10.2020-16.12.2020	
3.	Розробка архітектури та загальної структури системи	09.03.2021-23.03.2021	
4.	Розробка структур окремих підсистем	24.03.2021-10.04.2021	
5.	Програмна реалізація системи	14.04.2021-10.05.2021	
6.	Оформлення пояснювальної записки	20.05.2021-02.06.2021	
7.	Захист програмного продукту	12.05.2021	
8.	Передзахист	26.05.2021	
9.	Захист	15.06.2021	

Студент _____ _____Журавльов Р.В._____

(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Воронько М. П._____

(підпис) (прізвище та ініціали,

АНОТАЦІЯ

Мета роботи – проаналізувати вимоги і потреби користувача при роботі з системою пошуку наукових матеріалів. Розробити гнучку та багатофункціональну архітектуру, що реалізує графічний інтерфейс і дозволяє користувачу взаємодіяти з системою пошуку, вводити, змінювати та уточнювати фільтри та пошукові запити та виводити на екран результати пошукових запитів у вигляді списку.

Записка містить 55 сторінок, 34 рисунків та 9 посилань.

Ключові слова: Графічний інтерфейс, архітектура, пошук даних, візуалізація даних, формування запитів, фільтрація, клієнт, frontend, Javascript, React

SUMMARY

The purpose of the work is to analyze the requirements and needs of the user when working with the search system for scientific materials. Develop a flexible and multifunctional architecture that implements a graphical interface and allows the user to interact with the search engine, enter, modify and refine filters and search queries, and display search query results as a list.

The note contains 55 pages, 34 figures and 9 links.

Keywords: graphical interface, architecture, data search, data visualization, query generation, filtering, client, frontend, Javascript, React

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

API – прикладний програмний інтерфейс;

Frontend – клієнтська частина програми, графічний інтерфейс;

Backend – серверна частина програми;

Деплой – процес завантаження програми на сервер;

Хостинг – зберігання та надання доступу до створеної програми в мережі інтернет;

ВСТУП

Кожного року в світі публікується понад 2 000 000 наукових публікацій, дипломних робіт, статей, опитувань та досліджень. Це всесвітня безкоштовна бібліотека знань, накопичена людством за багато десятиліть клопітливої роботи сотень тисяч людей. Ця база знань дозволяє не витрачати власний час на повторне дослідження старих тем, а використовувати існуючі матеріали в своїй роботі. Результати робіт з цієї бази можуть бути використані для посилання як наукові джерела або як база для подальших, більш глибоких чи більш конкретних досліджень у обраній темі.

Актуальною стає проблема пошуку, фільтрації та сортування наукових матеріалів. На жаль, не існує єдиного універсального ресурсу, що надає доступ до всіх наукових публікацій в мережі інтернет. Дані зберігаються на різних онлайн ресурсах, в різних форматах та часто потребують аутентифікації для роботи. Для отримання найкращих результатів пошуку, потрібно збільшувати охоплення пошукової бази. Це означає одночасне використання багатьох ресурсів, що не є оптимальним чи зручним рішенням.

Часто онлайн ресурси не отримують своєчасного оновлення, надають складний та застарілий інтерфейс, не підтримують мобільні пристрої чи планшети. Або навіть не відкриваються в сучасних браузерах. Ми не можемо перенести всі доступні дані в одну єдину базу, бо це потребує багато коштів на підтримування серверів, бази даних та носіїв інформації для зберігання. Навіть якщо одночасно була б можливість Створити локальну копію всіх доступних матеріалів, вони б швидко втратили свою актуальність, бо підтримка такої системи потребує великої активної бази користувачів з усього світу, на кшталт Вікіпедії. З часом публікації втрачають свою актуальність: файли перестають відкриватись, посилання на роботи перестають працювати. Треба завантажувати нові дані та видаляти старі. Фільтрувати повтори, захищати базу від недобросовісних змін. Таку систему можна вважати нереальною на поточному етапі розвитку інформаційних технологій.

Як альтернатива, ми можемо створити нову модульну систему, яка сама не реалізує завантаження/зберігання матеріалів, але використовує вже існуючі рішення та дозволяє під'єднати інші інтернет ресурси та бази даних як модулі пошуку та використовувати їх для пошуку даних. Така архітектура уніфікує формат роботи з даними, дозволяючи одночасно працювати з декількома різними системами з одного клієнта. Об'єм даних, що потрібен для зберігання навіть великого каталогу матеріалів в такій системі значно менший, оскільки ми не зберігаємо текст публікацій, а тільки коротку інформацію та посилання на файл в інших базах. Фактично, ми отримуємо всі плюси уніфікованої системи, але не при цьому не потрібно вирішувати її проблеми. Індексування, пошук, ввід, модерація та зберігання публікацій віддається на реалізацію вже існуючим системам. Крім того, єдиний графічний клієнт простіше підтримувати. Ми можемо створити більш сучасне та зручне для використання графічне представлення, що буде працювати на всіх сучасних пристроях.

В цій роботі описана реалізація графічного клієнта для такої системи, що надає доступ користувачу до багатьох баз даних одночасно. При цьому користувач може фільтрувати, уточнювати запити та продивлятись результати пошуку. Програма доступна онлайн, не потребує завантаження, встановлення чи авторизації. Програма не є завершеним чи комерційним продуктом, а лише демонструє працездатність описаної концепції. Деталі реалізації та майбутні можливості для поліпшення програми описані далі.

1. ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ ЄДИНОГО РЕЄСТРУ НАУКОВИХ СТАТЕЙ ЗА ПОШУКОМ ПО КЛЮЧОВОМУ СЛОВУ

Спроектувати та розробити систему, що реалізує:

- Пошук по ключовому слову в обраних наукових базах
- Підтримує зручний вибір додаткових уточнюючих фільтрів по авторам, видавництвам, темам та рокам виходу
- Зберігає результати пошукових запитів у базі даних для подальшого аналізу
- Дозволяє легко додавати зовнішні бази даних до пошуку та використовувати їх як зовнішні незалежні модулі
- Виводить результати пошукових запитів у зручному, структурованому та простому для сприйняття графічному вигляді
- Працює на всіх сучасних пристроях та не потребує додаткових умов з боку користувача

Підсистема «Створення єдиного реєстру наукових статей за пошуком по ключовому слову» включає в себе клієнтську графічну частину взаємодії з системою. Сюди входить проектування та реалізація графічного інтерфейсу програми, що дозволяє користувачу формувати пошукові запити, налагоджувати фільтри, уточнювати пошуковий вираз та виводити на екран результати пошукового запиту.

Система має підтримувати видання результатів пошуку порціонно, та видавати решту результатів за бажанням користувача.

1.1 Вхідна та вихідна інформація системи

Вхідна інформація для формування пошукового запиту:

- Пошуковий рядок (String)
- Автори (Масив String)
- Видавництво (String)
- Теми (Масив String)
- Роки публікації (Діапазон початкова-кінцева дати)

Вихідна інформація результатів пошукового запиту:

- Назва публікації (String)
- Автори (Масив String)
- Видавництво (String)
- Рік публікації (Number)
- Мова публікації (String)
- Ключові слова (Масив String)
- Посилання на файл публікації (String)

1.2 Вимоги з точки зору користувача системи

Потенційними користувачами системи є наукові співробітники, студенти, педагоги, дослідники. Система призначена для надання більш швидкого, зручного та уніфікованого пошуку за багатьма науковими базами одночасно.

Користувач системи використовує свій персональний комп’ютерний пристрій щоб отримати доступ до системи та виконувати пошукові запити.

Як користувач, я очікую від створеної системи:

- Можливість робити пошукові запити по ключовому слову
- Можливість налаштовувати та уточнювати пошукові фільтри
- Можливість перегляди результати пошуку у вигляді каталогу публікацій
- Можливість відкрити повний текст обраної публікації
- Можливість зручно працювати з системою зі свого пристрою
- Високу швидкість роботи та виконання пошукових запитів

1.3 Вимоги з точки зору адміністратора системи

Система підтримується та розширюється розробниками та адміністраторами на базі ресурсів, наявних в компанії, що впроваджує систему.

Система залежна від зовнішніх існуючих пошукових баз, то ж при зміні в їх схемі роботи іноді потрібно вносити додаткові зміни, щоб відновити працездатність системи.

Як адміністратор системи, я очікую:

- Легкість в підтримці системи
- Легкість в розширенні пошукової бази
- Низькі вимоги до коштів та потужності обладнання
- Високу швидкість роботи та виконання пошукових запитів

2. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ “ СТВОРЕННЯ ЄДИНОГО РЕЄСТРУ НАУКОВИХ СТАТЕЙ ЗА ПОШУКОМ ПО КЛЮЧОВОМУ СЛОВУ”

Мережа пошукових баз недостатньо уніфікована. Публікації зберігаються в різних базах, в різних форматах, на різних сервісах. І не завжди сервіси надають зручний сучасний доступ до своїх матеріалів.

Проблеми такої сегрегації для кінцевого користувача можна описати так:

- Різні сервіси
- Різні інтерфейси
- Різні пошукові системи
- Різні фільтри
- Різні формати видачі
- Різні акаунти

В інтернеті існує безліч непов’язаних між собою наукових баз, що надають можливість робити пошукові запити. Найбільшими прикладами таких онлайн ресурсів є бібліографічна і реферативна пошукова база наукових публікацій Scopus [1], агрегатна безкоштовна бібліотека CORE [2], цифрова платформа JSTOR [3], пошукова система Google Scholar [4], наукова електронна бібліотека Кіберленінка [5].

Всі вони реалізують наступні базові функції:

- Пошук по ключовому слову
- Уточнюючі фільтри, пошук по авторам, видавництву, рокам публікації
- Вивід результатів пошуку

Окремі сервіси реалізують авторизацію та зберігати результати пошуку. При цьому деякі сервіси зберігають текст та файл публікації в своїй базі, а інші лише надають посилання на публікацію на інших ресурсах. Зберігання файлів публікації в власній БД часто гарантує, що файли з більшою долею ймовірності будуть доступні з плином часу. Коли файл завантажений на зовнішні ресурси, через деякий час вони можуть бути видалені чи стати недоступними по технічним причинам.

При огляді існуючої системи як потенційного пошукового модулю, потрібно проаналізувати наступні характеристики:

- Об’єм наявних публікацій в базі даних
- Чи є система безкоштовною
- Чи надає система офіційне публічне API для пошукових даних
- Наявні можливості для уточнення пошукових фільтрів
- Швидкість виконання пошукових запитів

Перевага надається безкоштовним публічним базам даним, що надають публічне пошукове API та документацію до нього. Крім цього, потрібно проаналізувати другорядні фактори, що можуть сказуватись на роботі системи:

- Об’єм щорічно завантажуваних статей
- Чи є у ресурсу модерація
- Чи проходять публікації перевірку на оригінальність та якість тексту
- Чи реалізує він захист від спаму, рекламних публікацій

2.1 Бібліотека CORE

Ресурс надає безкоштовний доступ до понад 200 000 000 (двохсот мільйонів) публікацій на різних мовах. Реалізує фільтрацію по рокам публікації, темі та мові публікації.

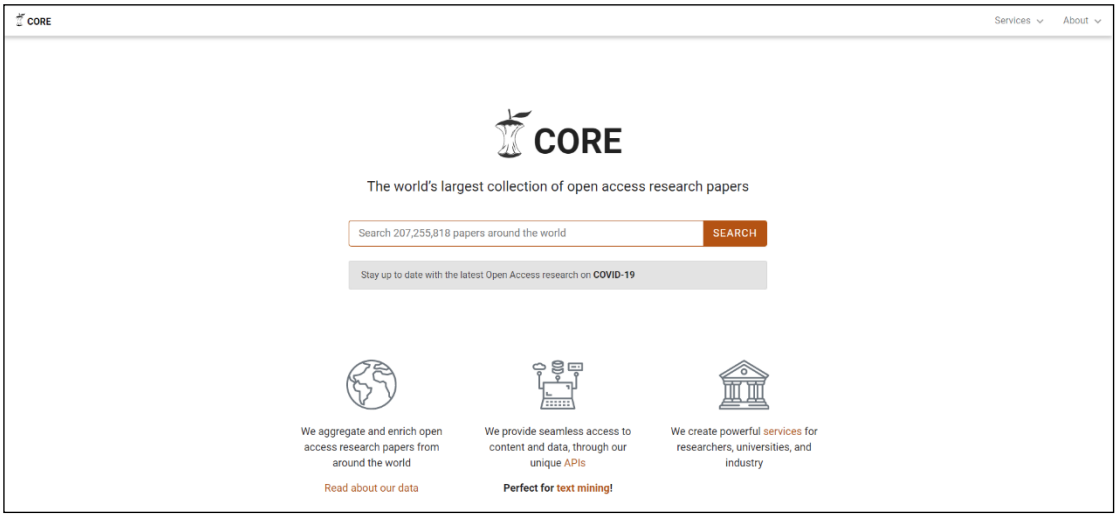

Рисунок 2.1.1 – Головна сторінка CORE

Плюси:

- Велика пошукова база публікацій
- Безкоштовний

Недоліки:

- Невелика база українсько- та російськомовних публікацій
- Часто повертає публікації з битим кодування
- Містить багато публікацій низької якості
- Багато повторів
- Часто файли публікації не відкриваються чи повертають помилку
- Погана документація

2.2 Пошукова база Scopus

Scopus надає потужну пошукову базу, але є дуже недружелюбним для нових користувачів. Потребує реєстрації для роботи. Часто працює з університетами напряму та надає доступ до бази даних обраним учбовим закладам.

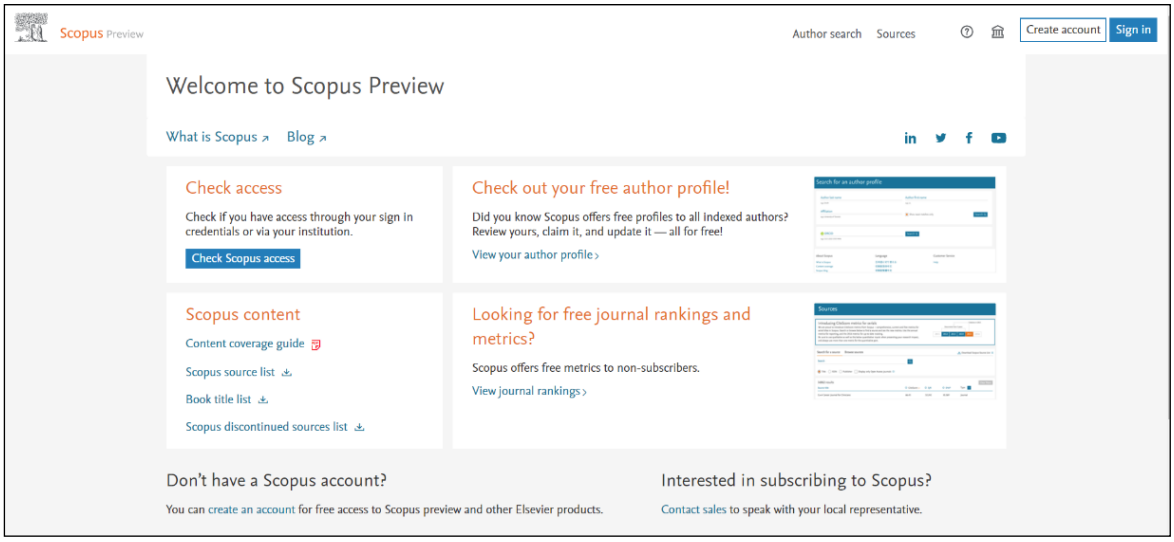

Рисунок 2.2.1 – Головна сторінка Scopus

Плюси:

- Висока якість публікацій
- Надає доступ до файлів публікації

Недоліки:

- Потребує реєстрації
- Дуже незручний інтерфейс
- Високий поріг входу
- Низька кількість українськомовних публікацій

2.3 Google Scholar (Google Академія)

Google дозволяє робити пошукові запити по багатьох наукових базах одночасно, виводячи результати у вигляді текстового списку інтернет сторінок.

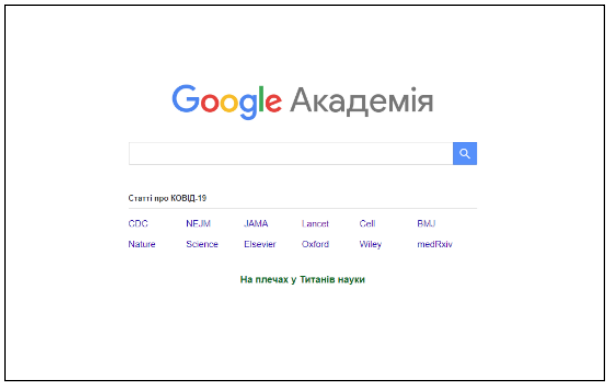

Рисунок 2.3.1 – Головна сторінка Google Scholar

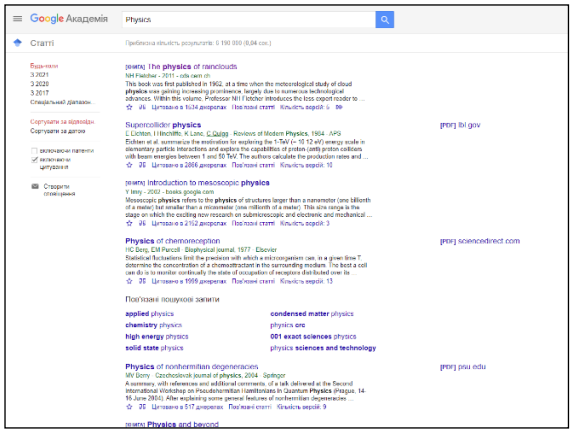

Рисунок 2.3.2 – Приклад пошукового запиту

Плюси:

- Велика пошукова база
- Показує кількість цитувань, пов’язані статті та пов’язані пошукові запити

Недоліки:

- Не надає прямого доступу до файлів публікації
- Мало фільтрів, потрібно запам’ятовувати синтаксис складання пошукових запитів

2.4 CyberLeninka (Кіберленінка)

Безкоштовна електронна бібліотека Cyberleninka надає доступ до великої наукової бази україномовних та російськомовних публікацій. Реалізує власні алгоритми пошуку та індексування публікацій.

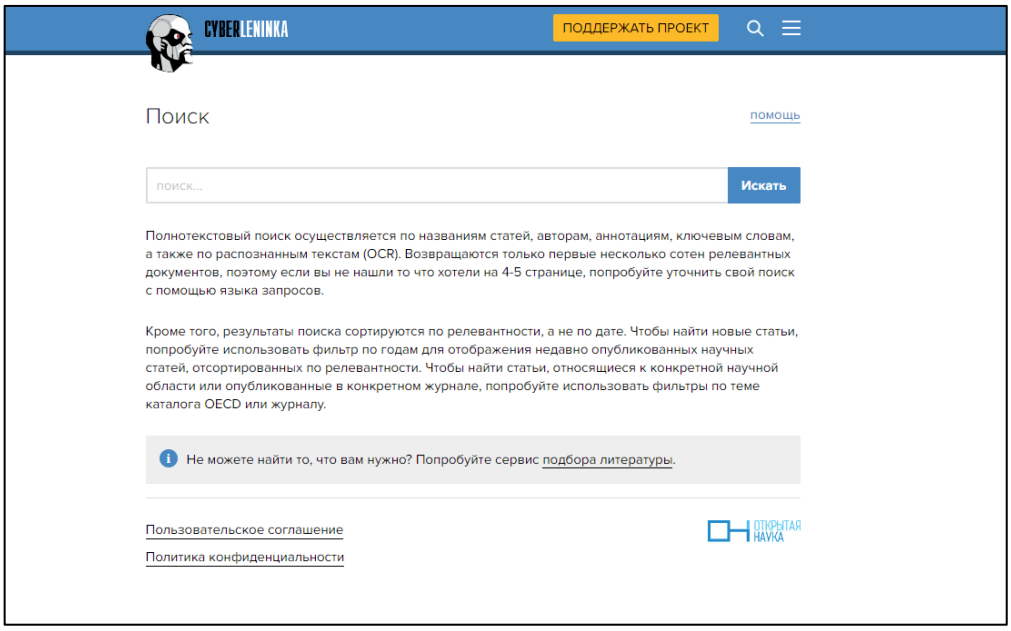

Рисунок 2.4.1 – Головна сторінка CyberLeninka

Плюси:

- Велика пошукова база україномовних та російськомовних публікацій
- Зберігає файли публікацій у власній базі даних
- Досить багато інформації про публікації, включаючи ліцензії, кількість переглядів та завантажень файлів
- Безкоштовна
- Простий інтерфейс, працює на телефонах

Недоліки:

- Пошукові фільтри потрібно писати текстом, використовуючи спеціальний синтаксис

2.5 Science Research

Електронна бібліотека, що надає платний доступ до англійськомовних наукових публікацій.

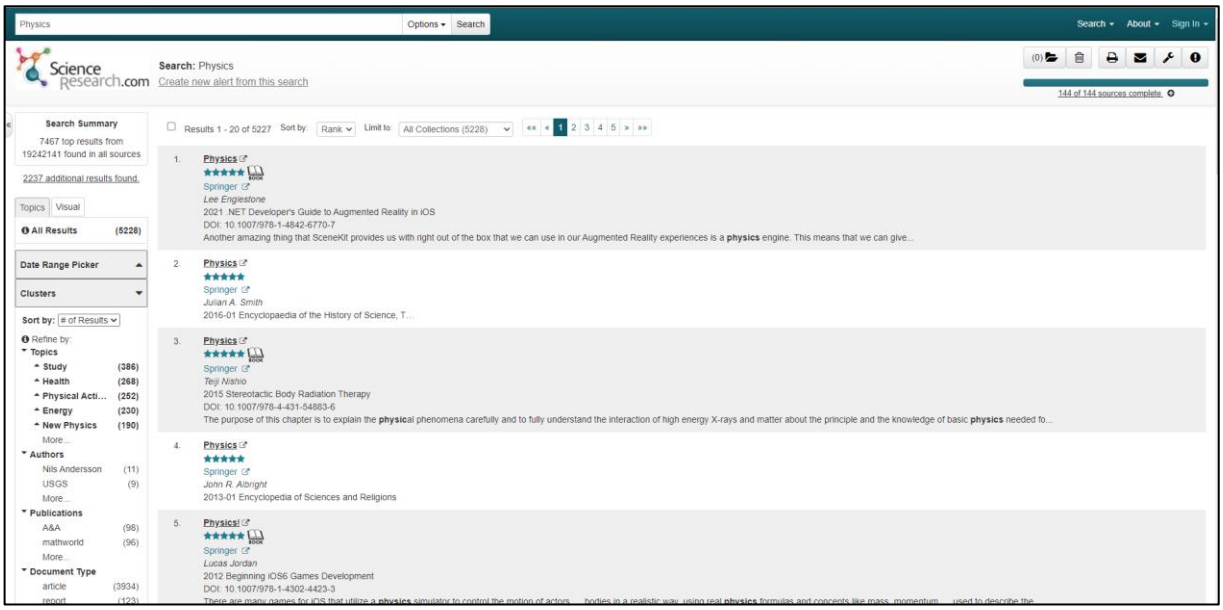

Рисунок 2.5.1 – Головна сторінка Science Research

Плюси:

- Висока якість публікацій

Недоліки:

- Не безкоштовна
- Складний та не інтуїтивний інтерфейс
- Низька кількість україномовних та російськомовних публікацій
- Відносно низький об’єм інформації карток в каталогу

3. ЗАСОБИ РОЗРОБКИ

Програма була розроблена на комп’ютері з операційною системою WINDOWS 10, але не потребує спеціальної операційної системи та може бути відкрита на комп’ютері з будь-якою ОС, включаючи MacOS та LINUX.

Для роботи з кодом використовувалося безкоштовне програмне середовище VS Code від Microsoft. На комп’ютері мають бути встановлен модуль Node версії не нижче 12 та будь-яка програма для роботи з програмними модулями, такі як npm (node package manager) чи yarn. Під час розробик використовувався Node версії 14.16.1 та Yarn версії 1.22.5.

Системні характеристики персонального комп’ютера, на якому відбувалася розробка:

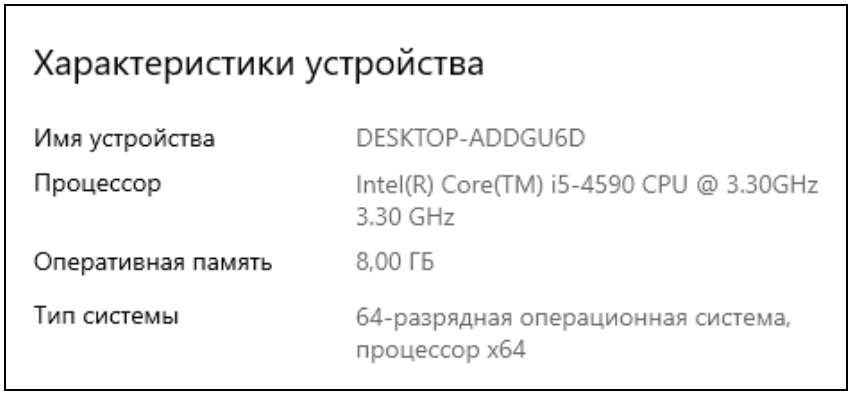

Рисунок 3.1 – Системні характеристики комп’ютера

Проект не визначає мінімальних чи рекомендованих системних характеристик для роботи, і може бути запущений на всіх сучасних персональних комп’ютерах і ноутбуках.

3.1 Необхідний об’єм пам’яті

Програма займає приблизно 700 Кілобайт (0.7 Мегабайт) пам’яті комп’ютера, включаючи зображення та файли типографії, але без додаткових зовнішнії модулів.

Включаючи додаткові зовнішні модулі, загальний розмір програми досягає приблизно 250 Мегабайт. Також для локального запуску програми потрібно додатково виділити до 300 Мегабайт. Тож загальний розмір створеної програми на диску під час локальної розробки досягає 600 Мегабайт.

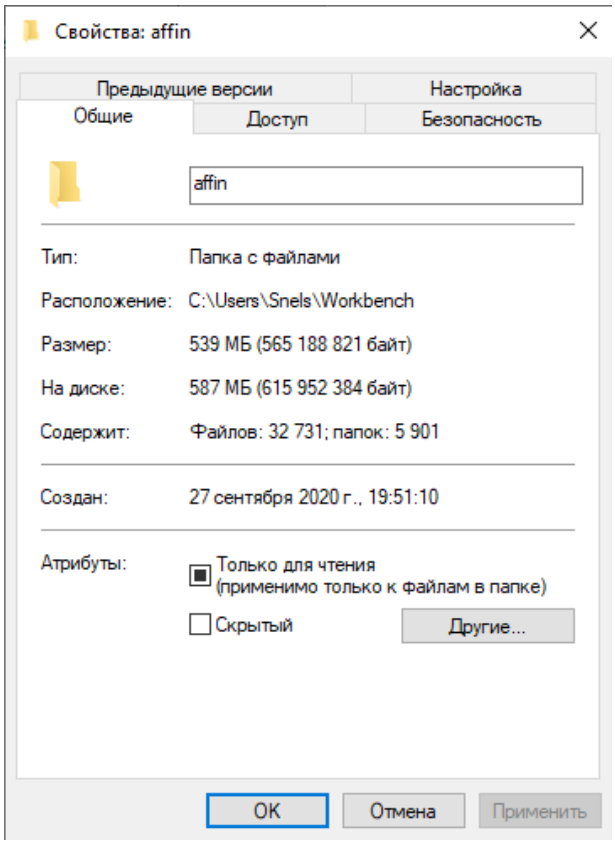

Рисунок 3.2 – Об’єм займаємої пам’яті програми на пристрої

Розмір кінцевої програми, що користувач завантажує під час відкриття вебсайту – до 1 мегабайту при першому завантаженні, і від 250 кілобайт до 500 кілобайт при наступних завантаженнях.

Архітектура розробленої програми не потребує серверу для роботи, а також підтримує технологію “Serverless”, коли файли програми не використовують активну оперативну чи процесорну пам’ять, автоматично вимикаються, економлячи ресурси, та потребують комп’ютерного обчислення та виконання тільки під час виклику від клієнта.

Вимоги до системних характеристик онлайн-хостингу залежать від кількості активних користувачів та об’єму трафіку/відвідувань сторінок в місяць. Теоретично, об’єм до 10 000 (десяти тисяч) завантажень в місяц може бути завантажений на будь який безкоштовний онлайн сервіс для хостингу вебсайтів, таких як Vercel, Netlify, Glitch, Digital Ocean чи AWS. Всі файли та сторінки розробленого вебсайту є статичними, тож не потребують додаткової процесорної роботи під час виклику.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Схематично загальна система складається з трьох частин: клієнта, сервера та модульного зовнішнього API. Сервер оголошує структуру та формат даних, в яких очікує отримати дані від API, а окремі пошукові модулі виконують пошуковий запит на зовнішньому ресурсі та конвертують отримані результати у єдиний формат. Знайдені результати передаються клієнту як єдиний об’єднаний пошуковий запит.

Рисунок 4.1 Архітектура розробленої системи

Основна мета розробки програми – надати швидку та зручну можливість користувачу виконувати пошукові запити. Для цього було прийнято архітектурне рішення реалізувати програму як онлайн веб-сайт. Таким чином користувачу не буде потрібно завантажувати чи встановлювати додаткові файли на свій пристрій. Також це дозволяє користувачу користуватись розробленою програмою не тільки зі стаціонарного комп’ютера, а також з телефонів, планшетів та будь-якого іншого пристрою з виходом в інтернет.

Програма реалізована на строго типізованій мові програмування Typescript, розробленій корпорацією Microsoft. Це надає можливість проводити статичний аналіз типів коду та виявляти проблеми та помилки на стадії написання коду. Перед тим як програма завантажується на хостинг, Typescript компілюється в динамічну мову програмування Javascript.

Як база для програми була обрана графічна бібліотека React. Альтернативно, замість обраної бібліотеки можна було взяти Vue, Angular, Svelte чи реалізувати всі додаткові елементи власноруч, але бібліотека React була обрана через свою велику базу готових модулів та бібліотек, що дають готові компоненти та значно прискорюють розробку.

Також проект був побудований навколо фреймворку Next.js, що працює поверх бібліотеки React та реалізує роботу з різними сторінками на клієнті в рамках одної програми, дозволяє використовувати технологію SSR (Server Side Rendering) та надає готове рішення для створення кінцевих, оптимізованих версій програми.

Повний перелік використаних бібліотек, їх версії та короткий опис надані далі.

4.1 Використані бібліотеки та технології

Проект підмикає зовнішні бібліотеки та node модулі, що реалізують додаткові інструменти розробки та функції. Точні версії бібліотек наведені на рисунку 4.1.1. Незначущі другорядні бібліотеки в описі пропущено.

```
  },
  "dependencies": {
    "@chakra-ui/react": "^1.6.1",
    "@emotion/react": "^11.4.0",
    "@emotion/styled": "^11.3.0",
    "@react-three/fiber": "^6.1.2",
    "axios": "^0.21.1",
    "framer-motion": "4.1.16",
    "next": "10.2.0",
    "next-i18next": "^8.2.0",
    "react": "17.0.2",
    "react-dom": "17.0.2",
    "react-icons": "^4.2.0",
    "react-query": "^3.16.0",
    "react-range": "^1.8.7",
    "react-select": "^4.3.1",
    "recoil": "^0.3.0",
    "three": "^0.128.0"
  },
  "devDependencies": {
    "@types/node": "^15.3.0",
    "@types/react": "^17.0.5",
    "@types/react-dom": "^17.0.5",
    "@types/three": "^0.128.0",
    "@typescript-eslint/eslint-plugin": "^4.23.0",
    "@typescript-eslint/parser": "^4.23.0",
    "eslint": "^7.26.0",
    "eslint-config-alloy": "^4.1.0",
    "eslint-plugin-jsx-a11y": "^6.4.1",
    "eslint-plugin-react": "^7.23.2",
    "next-compose-plugins": "^2.2.1",
    "typescript": "4.2.4"
  }
}
```

Рисунок 4.1.1 – Перелік залежностей проекту

- Next – Програмний фреймворк, що працює на базі React. Реалізує SSR (Server Side Rendering), надає готове рішення для роботи з веб-сторінками, автоматично підмикає та налаштовує Webpack, компілює Typescript файли в Javascript, реалізує технологію Tree Shaking, оптимізує та збирає робочі версії програми.

- React – Сучасна бібліотека для побудови графічних інтерфейсів, зберігання поточного стану компонентів та реалізації додаткової логіки, розроблена командою Facebook
- Typescript – Строго типізована мова програмування, розроблена корпорацією Microsoft
- Axios – Бібліотека, що реалізує готові методи для більш простої роботи з інтернет запитами. Використовується для взаємодії з Backend API
- React-query – Бібліотека, що надає готові компоненти для роботи з інтернет запитами та автоматичного зберігання в кеш результатів виконання пошукових запитів
- React-range – Бібліотека, що реалізує компонент вибору діапазону року публікації
- React-select – Бібліотека, що реалізує компонент вибору авторів, видавця та тегів
- Recoil – Експериментальна технологія, розроблена командою Facebook, що надає доступ до глобального стану програми з будь-якого компоненту системи
- Chakra – Бібліотека готових компонентів, кнопок та елементів форм
- Emotion – Бібліотека, що дозволяє визначати стилі компонентів, використовуючи спеціальний CSS-in-JS синтаксис
- Next-i18next – Бібліотека для перекладу програми на інші мови
- Eslint – Бібліотека для аналізу коду, виявлення та виправлення лексичних та синтаксичних помилок

4.2 Файлова структура проекту

Проект містить стандартну структуру файлів для такого типу програм. Більшість цих файлів несуть в собі другорядну роль, тому їх зміст та опис пропущено. Всього, не рахуючи папку з зовнішніми модулями “node_modules” в проекті знаходиться 63 файли загальним неоптимізованим об’ємом 800 кілобайт, з яких коду – 300 кілобайт.

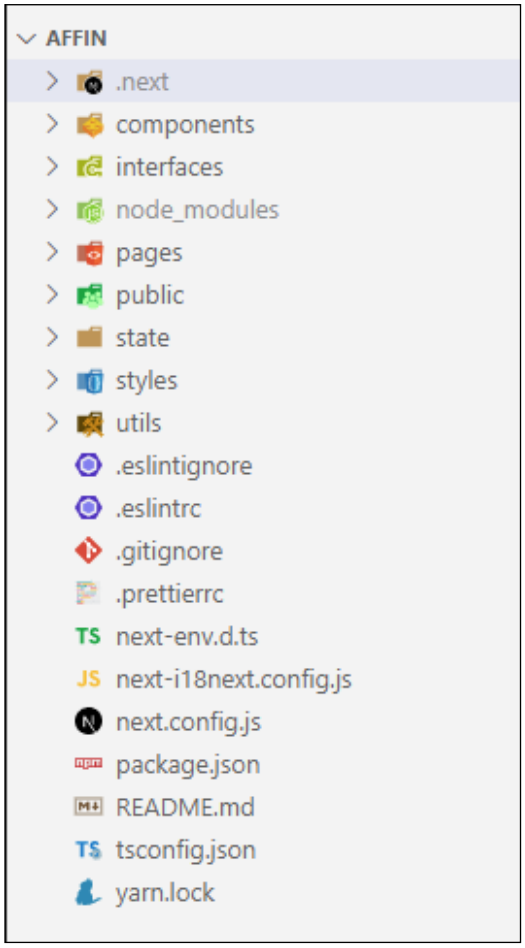

Рисунок 4.2.1 – Вигляд файлової структури проекту

Основна логіка програми розбита на 7 основних папок:

- Components – Папка, що зберігає всі створені компоненти, картки, поля вводу, кнопки, текстові компоненти та розмітку сторінок

- Interfaces – Папка, в якій знаходяться визначення типів для роботи з API. Вони документують які параметри зберігаються в пам’яті, їх назву, типи, та перелік значень, що потрібно передавати в пошукові функції, та які параметри вони повертають
- Pages – Папка, в якій оголошено перелік сторінок програми, а також їх вміст та url
- Public – Папка для публічних статичних асетів програми: ілюстрацій, типографії, переводів програми на інші мови, тощо
- State – Папка, що реалізує глобальний стан програми за допомогою бібліотеки Recoil
- Styles – Глобальні CSS стилі
- Utils – Допоміжні функції та компоненти

4.3 Глобальний стан програми

Для зберігання поточного стану програму в пам’яті, введений пошукова стрічка та обрані фільтри були виведені за межі локальних компонентів та розміщені у глобальному стані за допомогою бібліотеки Recoil. Таким чином введені параметри зберігаються при переході між сторінками.

```
...
import { atom } from "recoil";
import { Filters } from "interfaces/filters";

export const searchSearchState = atom<string | null>({
  key: "searchSearchState",
  default: "",
});

export const searchFiltersState = atom<Filters>({
  key: "searchFiltersState",
  default: {
    authors: null,
    publishers: null,
    topics: null,
    publishedAfter: null,
    publishedBefore: null,
  },
});

```

Рисунок 4.3 – Глобальний Recoil стан програми

Глобальний стан розбитий на дві базові частини: пошукову стрічку та фільтри. Це рішення обумовлено тим, що ввід фільтрів є опціональним, і вони часто можуть бути невизначені. Пошукова стрічка, з іншого боку, часто змінюється та має бути визначена для пошуку.

Доступ до цих параметрів є з будь-якого місця програми. Але при цьому вони не використовуються напряму для створення пошукового запиту. Оскільки кожна пошукова сторінка має мати свою унікальну url адресу, програма зчитує фільтри, визначені в url сторінки як параметри. При натисканні кнопки “пошук”, програма просто змінює поточний url сторінки на /search?..., вставляючи після знаку питання

ненульові визначені фільтри. Семантично, зміна цього url вручну користувачем виконує такий саме пошуковий запит, що і кнопка “пошук”. Тож при перезавантаженні сторінки програма запам’ятає обрані фільтри і користувач побачить туж сторінку з тим самим пошуковим запитом. Це можна використовувати для того, щоб поширювати посилання на пошукові сторінку між користувачами чи зберігання обраного посилання на пошуковий запит.

Крім того, в програму додана перевірка на тип параметрів, визначених у посиланні. Якщо програма очікує String, а користувач визначить масив String, помилки не буде. І навпаки, програма автоматично конвертує String в масив String для параметрів, де очікується декілька значень.

4.4 Пошукові запити до API

Для роботи з API використовується бібліотека React-Query. Вона автоматично зберігає в кеш пам'ять результати виклику пошукових запитів. В функцію передаються лише пошукова стрічка, обрані фільтри та посилання на API. Поточний стан виконання та помилки будуть згенеровані бібліотекою.

```
33
34  export const getArticles = async (
35    query: { search: string; filters: Filters | null },
36    limit = 20,
37    offset = 1,
38  ) => {
39    const { search, filters } = queryToKey(query);
40
41    return axios.post("https://kpi-affin-2021.herokuapp.com/articles/", {
42      searchQuery: search,
43      offset,
44      limit,
45      filter: filters,
46    });
47  };
48
```

Рисунок 4.4.1 – Виклик API пошуку наукових статей

Запит формується за допомогою бібліотеки Axios, що дозволяє реалізує дуже зручний інтерфейс для роботи з завантаженням даних. Хоча семантично це GET виклик, він реалізований як POST виклик, щоб мати можливість передавати обрані параметри в тілі запиту.

4.5 Адаптивний інтерфейс

Розроблена програма підтримує широку варіативність розмірів екранів пристроїв. Клієнт коректно працює на маленьких мобільних пристроях та великих широких комп’ютерних екранах.

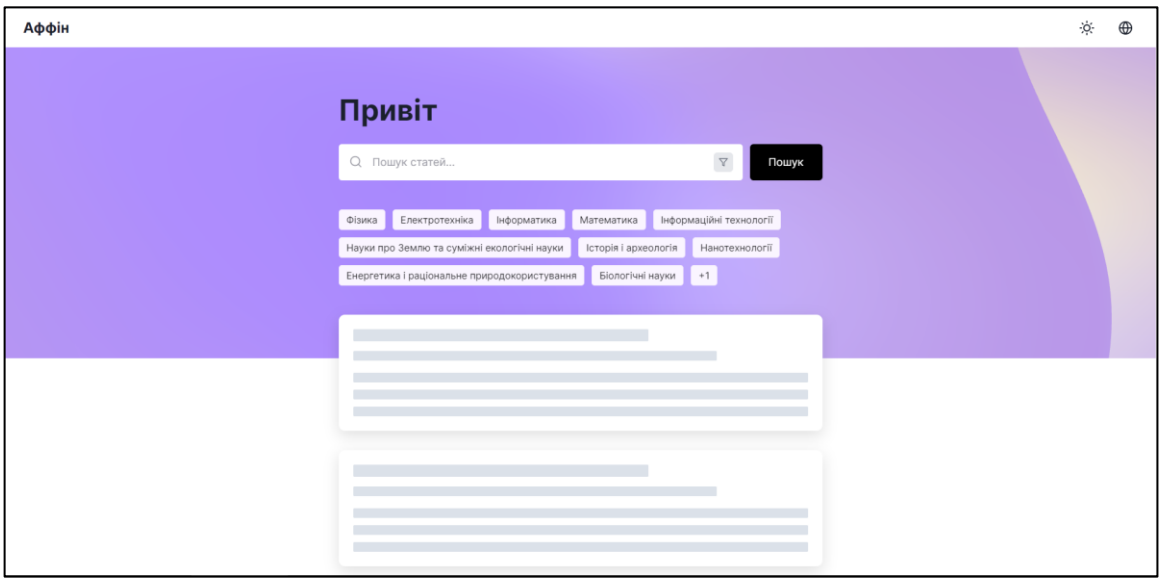

Рисунок 4.5.1 – Вигляд головної сторінки на великих екранах 1920 пікселів

Головні блоки структуруються в одну єдину колонку по середині екрана, що дозволяє користувачу зручно працювати з усіма елементами без необхідності переміщати курсор на великі відстані. Інтерфейс може нескінченно поширюватись у висоту в залежності від кількості карток публікацій та тегів.

Мінімальний підтримуємий розмір екранів – 320 пікселів в ширину. Екрани менше 320 пікселів будуть відображати програму у вигляді вікна 320 пікселів з горизонтальним скролом сторінки. Вертикальний розмір екранів не обмежений.

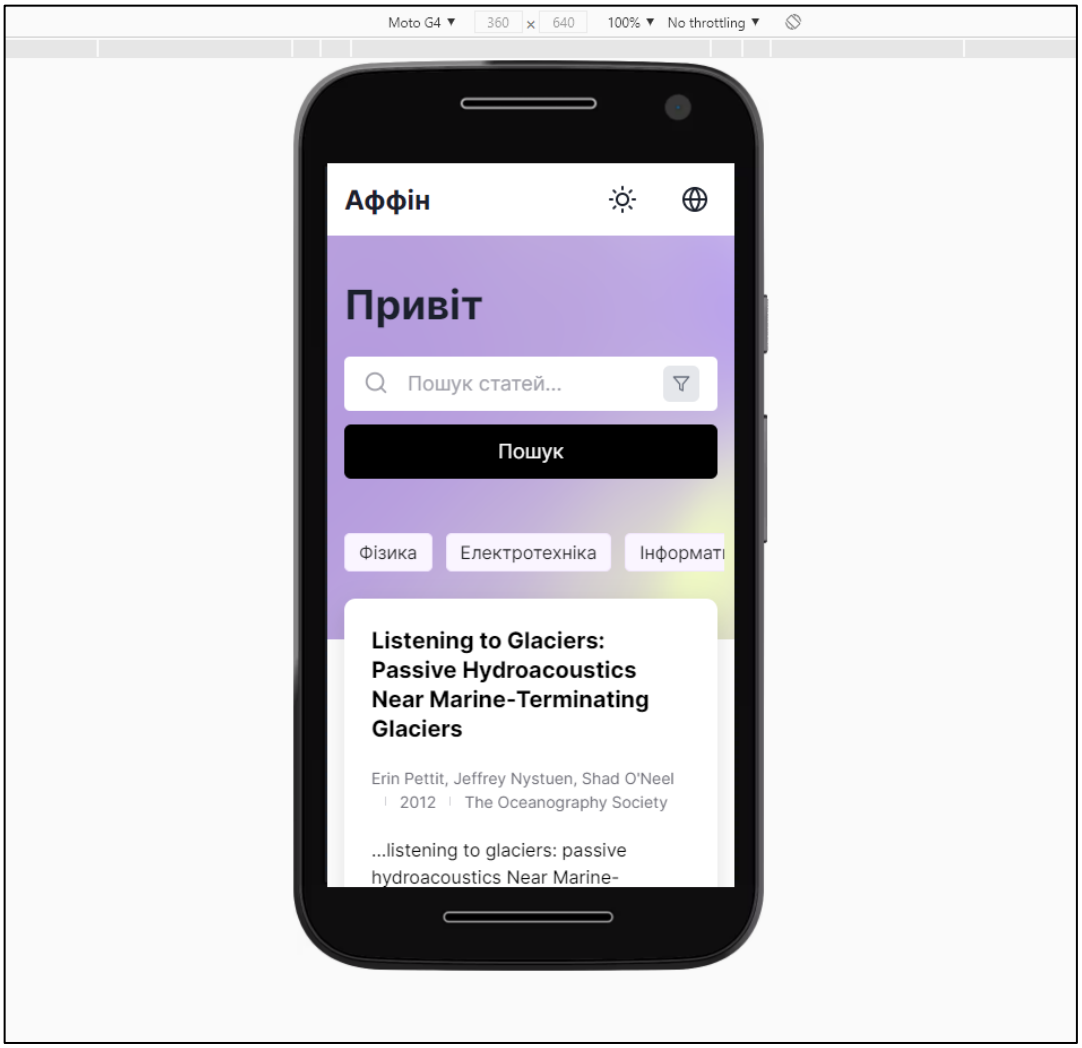

Рисунок 4.5.2 – Вигляд головної сторінки на маленьких та мобільних екранах 360 пікселів

Результати пошукового запиту розташовані у вигляді зручного вертикального каталогу, дозволяючи користувачу працювати з одною публікацією за раз або перегляді декілька карток одночасно. Всі основні елементи інтерфейсу розташовані в зонах, зручних для сенсорних екранів, та не потребують від користувача тягнутись до елементів в верхній частині екрану для роботи.

4.6 Пошукові фільтри

По замовчуванню додаткові пошукові фільтри вимкнені та не відображаються на головній сторінці. Для того, щоб ввімкнути їх, необхідно натиснути відповідну кнопку з зображенням фільтра всередині пошукового поля.

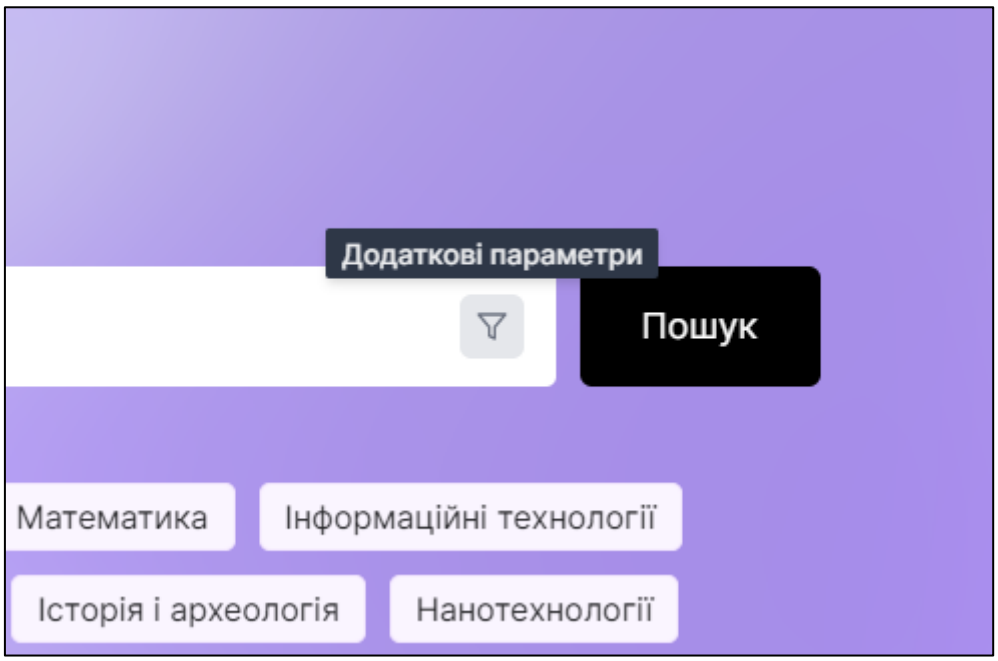

Рисунок 4.6.1 – Неактивна кнопка відображення та вимикання додаткових параметрів

При наведенні курсора на кнопку з’явиться допоміжне вікно з назвою “Додаткові параметри”, що повідомляє користувачу призначення та роль цього елемента інтерфейсу. Також цей перемикач можна виділити та натиснути використовуючи клавіатуру, за допомогою навігації кнопкам “Tab” та “Shift+Tab”. При створенні фільтрів були використані бібліотеки “react-select” та “react-range”.

Після натискання, кнопка повинна стати зеленого кольору, це означає, що додаткові фільтри ввімкнені. При цьому під полем вводу з’явиться додатковий блок з фільтрами по іменам авторів публікацій, назви видавництва, темам та рокам публікації.

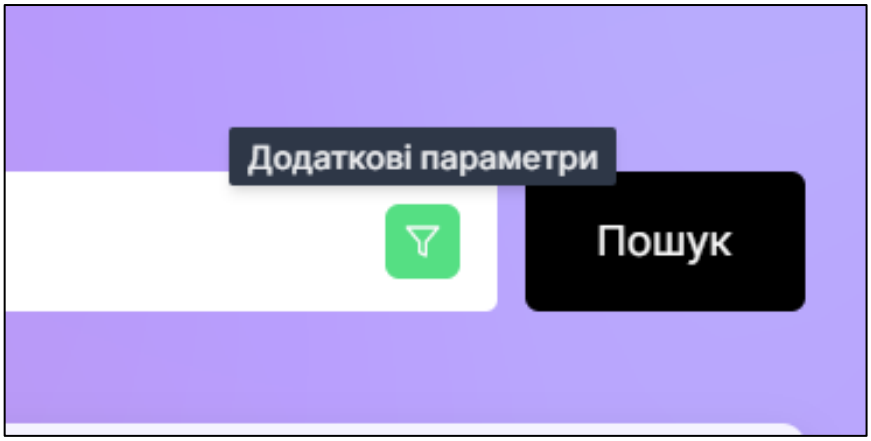

Рисунок 4.6.2 – Активна кнопка відображення та вимикання додаткових параметрів

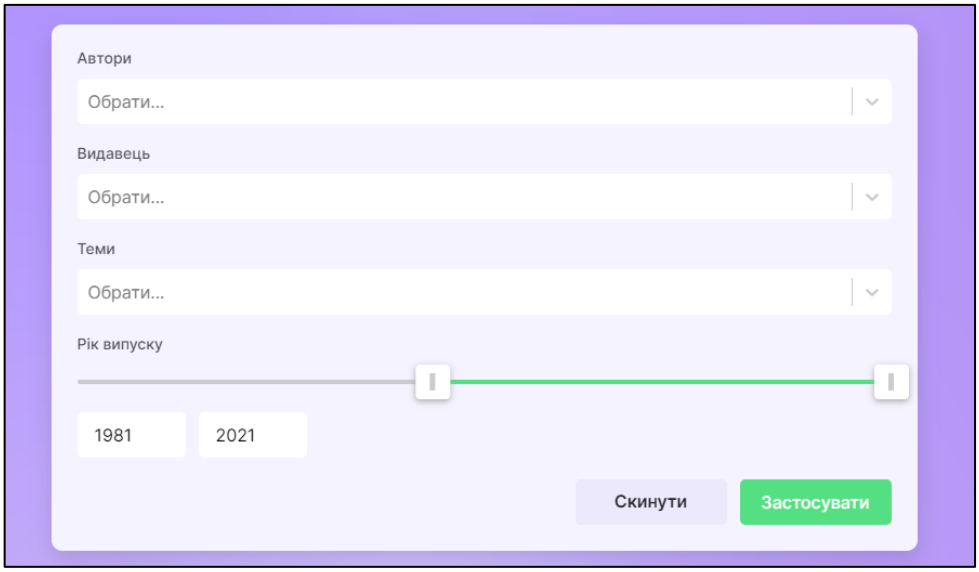

Рисунок 4.6.3 – Модуль додаткових пошукових фільтрів

Модуль додаткових пошукових фільтрів надає користувачу два типи фільтрів: поле вибору з автодоповненням та діапазон вибору років публікації. По замовчуванню, діапазон пошукових років публікації включає в себе публікації, видані

за останні 40 років, включаючи поточний рік. Фільтр по рокам публікацій неможливо вимкнути, але при бажанні користувача можливо розширити рамки пошукового діапазону до потрібного значення. Користувач може визначити діапазон за допомогою спеціального перемикача, перетягнувши крайові елементи використовуючи курсор чи сенсорний ввід. Альтернативно, користувач може ввести пошукові роки вручну, використовуючи два поля вводу, де ліве поле відображає мінімальний рік публікації, а праве поле – максимальний рік публікації.

Рисунок 4.6.4 – Заповнені пошукові фільтри для авторів та видавництва

Поля вибору зовні виглядають як поля вводу, але коли користувач натискає на таке поле, відкривається додаткове вікно, де користувач може ввести та обрати один чи декілька текстових варіантів для відповідного пошукового фільтра. Також обрані варіанти можливо вимкнути, натиснувши відповідну кнопку всередині поля вводу.

Користувач може обрати один або більше значень для авторів та тем та одне видавництво. При збільшенні кількості обраних фільтрів розмір компонента збільшується.

4.7 Автодоповнення

Програма використовує пошукові запити та надає готовий перелік авторів та видавництв, які користувач може обрати. Коли користувач фокусується на полі вводу, система виводить вікно з переліком знайдених варіантів.

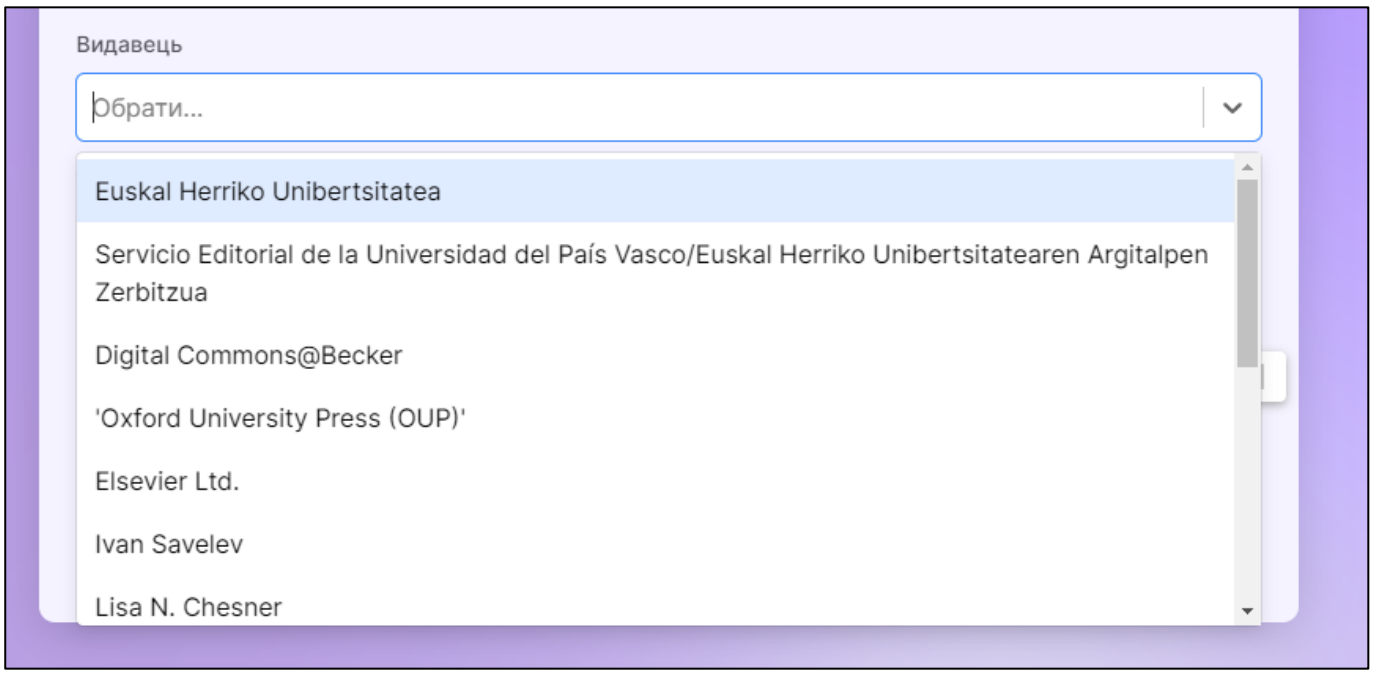

Рисунок 4.7.1 – Перелік існуючих видавництв

Користувач може натиснути на обраний варіант та він автоматично обереться як пошуковий фільтр. Якщо потрібного варіанта немає в списку, користувач може уточнити пошуковий запит або ввести повну назву самостійно та натиснути Enter.

За один раз програма надає до 20 знайдених варіантів назв. Це зробленого для того, щоб не завантажувати великі об’єми даних на кожний пошуковий запит.

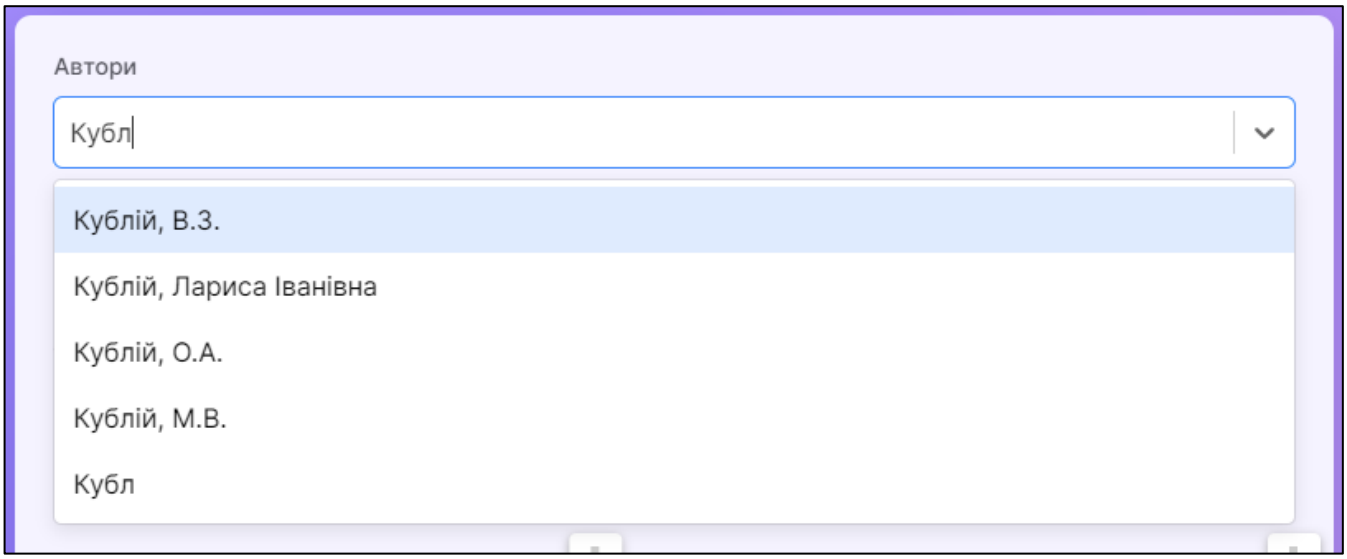

Рисунок 4.7.2 – Автодоповнення по ключовому слову

Після того як користувач обрав пошуковий фільтр, він буде відображений у пошуковому полі. Якщо поле підтримує декілька значень, вони будуть відображені всередині поля як окремі теги. Кожен тег можливо самостійно видаляти чи змінювати, не змінюючи інші значення.

4.8 Розгортання програми

Для розгортання створеної програми був обраний безкоштовний сервіс Vercel. Ця платформа дозволяє швидко та легко завантажувати веб-програми на свої домени. Vercel надає безкоштовний план, якого достатньо для підтримки досить великого інтернет трафіку. Можна очікувати, що програма не перебільшить задані ліміти та не буде потребувати коштів для підтримки.

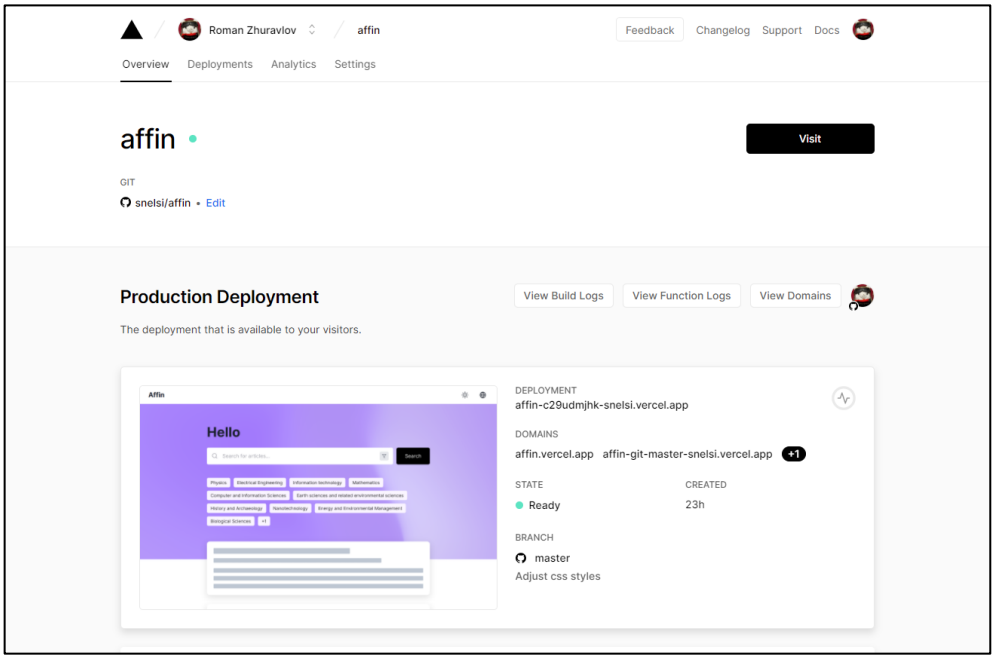

Рисунок 4.8.1 – Сторінка програми на Vercel

При цьому безкоштовного url другого рівня ‘vercel.app’ достатньо для працездатності створеного клієнта. Як альтернатива, для хостингу можна було використати схожі платформи, такі як Netlify. Але Vercel був обраний як платформа від авторів фреймворка Next.js, який ми використовуємо для розробки програмної частини. Вона надає певні додаткові функції для інтеграції та аналітики.

Програма доступна онлайн за посиланням https://affin.vercel.app/.

4.9 Темна тема інтерфейсу

Програма використовує компоненти з безкоштовної бібліотеки Chakra UI для створення єдиного та узгодженого інтерфейсу. Це також надає можливість користувачу обирати альтернативну темну тему інтерфейсу. Це зменшує яркість та напругу на очі при низькому освітленні.

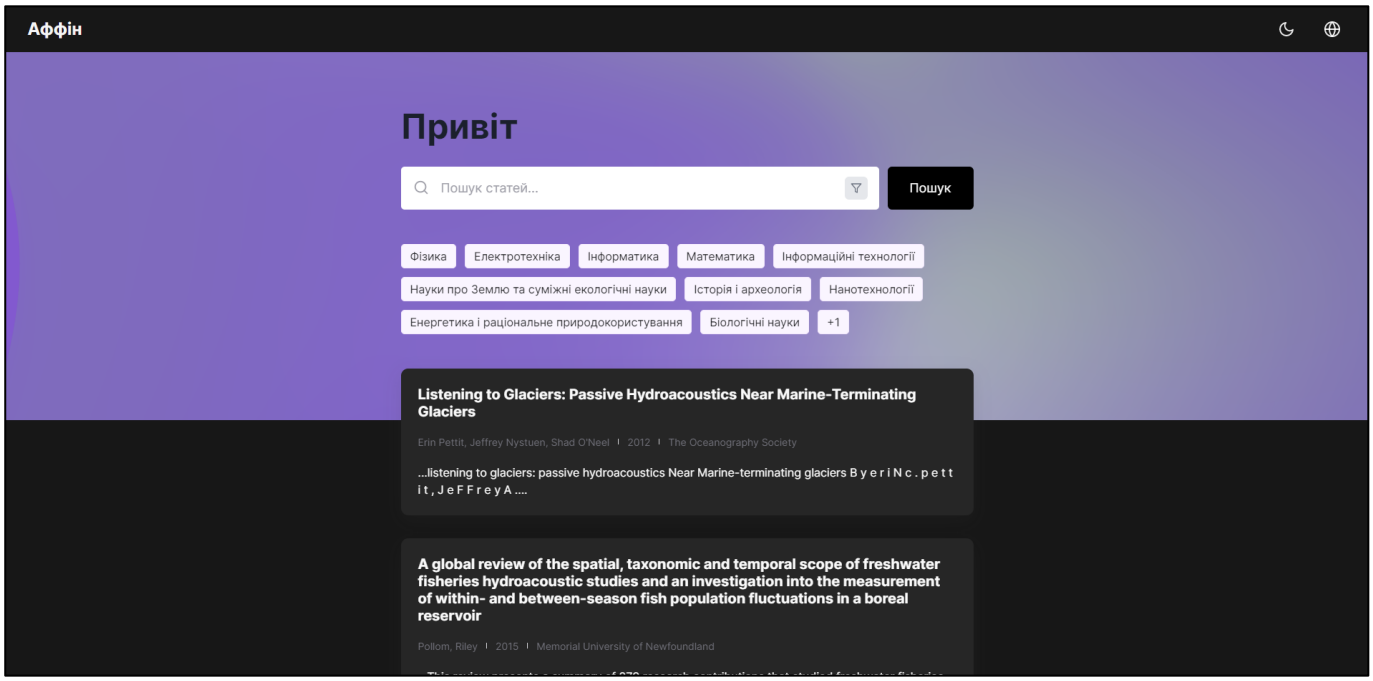

Рисунок 4.9.1 – Темний варіант інтерфейсу

Для того щоб змінити кольорову тему програми необхідно натиснути відповідну кнопку в правому верхньому кутку програми з зображенням сонця. Ілюстрація зміниться на місяць та тема зміниться зі світлої на темну.

Програма автоматично запам’ятає обрану тему. Коли користувач зайде на сайт наступного разу, буде відображена відповідна кольорова тема.

4.10 Переклади на інші мови

Програма використовує бібліотеку i18n для того щоб підтримувати декілька варіантів перекладу. Користувач може обрати українську, англійську або російську мови інтерфейсу.

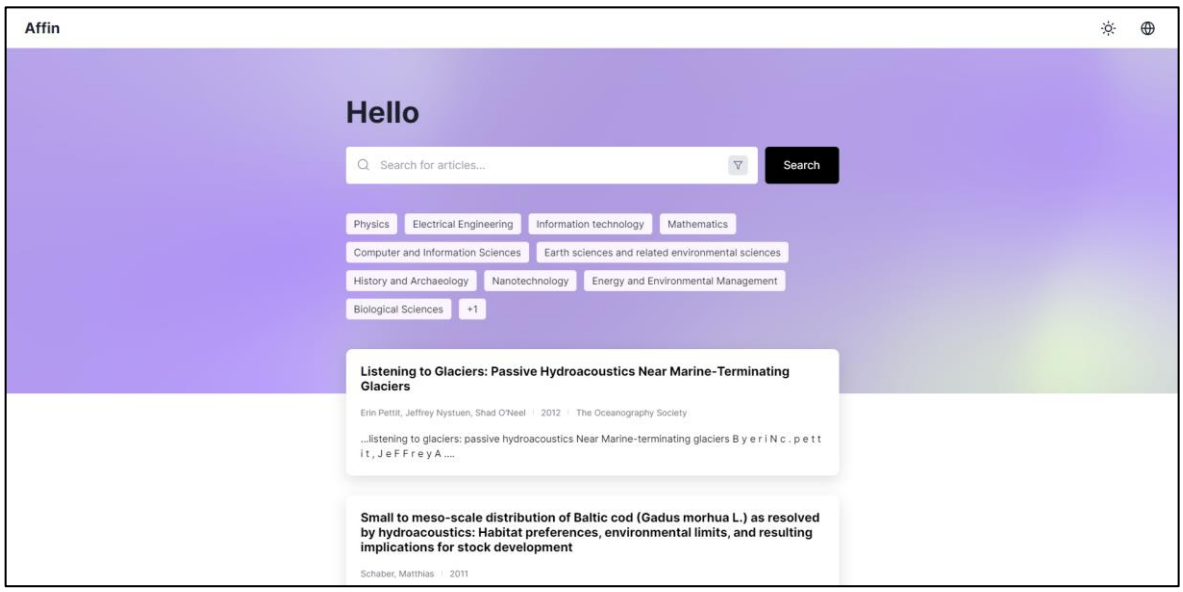

Рисунок 4.10.1 – Англійський переклад програми

Коли користувач заходить на головну сторінку веб-сайту, програма автоматично перевіряє обрану мову браузера. Якщо користувач використовує українську мову інтерфейсу, програма також буде відображена, використовуючи українські переклади. Якщо перевірити мову інтерфейсу користувача неможливо або потрібної мови немає в перекладах, буде відображено англійський переклад програми.

Переклади зберігаються у форматі JSON. Кожна мова має свій окремий файл з перекладами. У кожної текстової стрічки всередині програми є унікальний код, який використовуються для знаходження перекладів для потрібної мови.

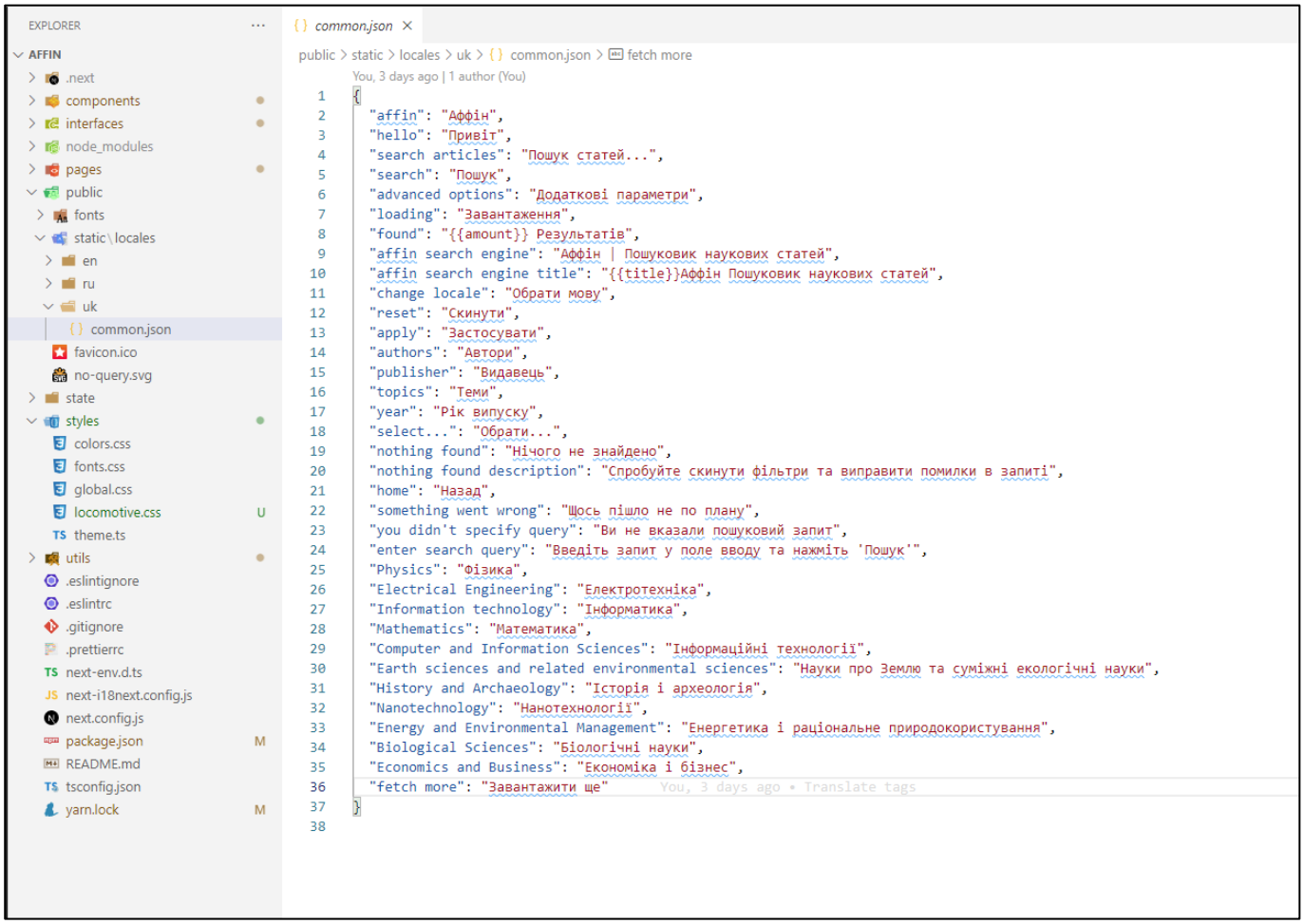

Рисунок 4.10.2 – Файл з українськими перекладами

Розробники системи можуть легко додавати нові переклади та підтримувати існуючі. Якщо для певної мови не існує потрібного перекладу, буде відображений англійський варіант перекладу.

4.11 Типографія

Програма використовує безкоштовне динамічне сімейство типографії ‘Inter’, розроблене Размусом Андерсоном у 2016 році. Цей шрифт було обрано через те, що він підтримує латиницю, кирилицю та створює узгоджене візуальне представлення на різних мовах, комп’ютерних пристроях та платформах.

Це досить популярне рішення, що часто використовуються на сучасних англійськомовних сайтах та серед розробників. Inter гарантує гарний контраст та читаємість тексту. Для забезпечення найкращої читаємості всі багаторядкові текстові абзаци мають висоту рядка 1.5 та не менше розміру 14 пікселів. Заголовки публікацій та компонентів є жирними для забезпечення візуальної ієрархії та розбиття компонентів на логічні групи.

Вся типографія зберігається на стороні сервера та підмикається як єдиний файл за допомогою технології ‘Variable font’, що зменшує кількість та об’єм файлів, необхідних програмі для роботи.

4.12 Доступність

Програма спроектована та розроблена з урахуванням цифрової доступності та націлена на просту та зручну роботу для людей всіх вікових категорій. Всі елементи програми мають високий контраст та можуть бути виділені та обрані з клавіатури.

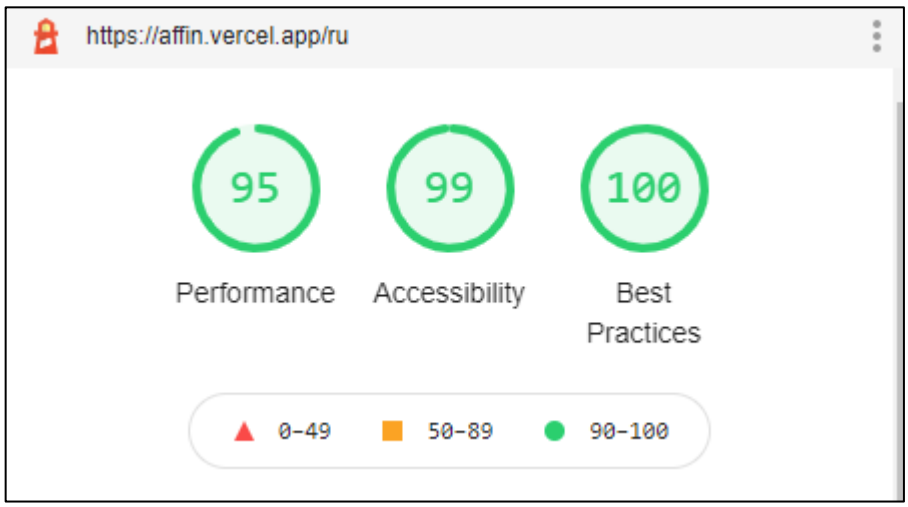

Рисунок 4.12.1 – Результати тестування сайту через програму Lighthouse від Google

Програма використовує всі сучасні рекомендації, вимоги та найкращі практики по створенню сучасних, швидкодійних та доступних веб-сайтів від Google та Airbnb. Створений веб-сайт також маєш відносно високий пошуковий індекс SEO для відображення в пошукових фільтрах Google та Bing, але це не ставилося ціллю розробки, а є результатом інших факторів, таких як швидкодія та доступність програми.

4.13 Допоміжні інструменти

Для підтримки єдиного кодового стилю та автоматичної статичної перевірки файлів всередині програми на логічні та синтаксичні помилки використовуються бібліотеки Eslint та Prettier.

Prettier автоматично форматує код, забезпечуючи стандартне форматування для всіх розробників програми, незалежно від комп'ютера чи операційної системи. Eslint проводить розширений аналіз кодової бази на наявність логічних, синтаксичних чи стильових помилок, а також повідомляє розробника про знайдені погані кодові шаблони та практики. Eslint може автоматично виправляти більшу частину знайдених помилок, підвищуючи якість кінцевого коду програми та додаючи додаткове тестове покриття файлів та логіки.

За основу для налаштування Eslint був взятий публічна конфігурація 'Alloy'. Вона надає гнучкий сучасний перелік правил та практик, що задає високий стандарт написання чистого функціонального коду та при цьому не обмежує розробника у реалізації внутрішньої логіки програми.

Більш популярний варіант конфігурації від Airbnb в цьому випадку може бути занадто вимогливим до розробників. Часто рекомендації та практики з цієї конфігурації знижують читаємість коду, потребують додаткового часу та сил від розробника та при цьому проблеми, які вони намагаються виправити не зустрічаються на проектах такого малого розміру.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Програма не потребує встановлення на комп’ютер користувача. Розроблена програма працює онлайн та доступна в будь-якому браузері як веб-сайт.

5.1 Головна сторінка

Головна сторінка програми складається з поля вводу пошукового запиту, кнопки пошуку, додаткових фільтрів, переліку популярних топіків та каталогу останніх статей. В правому верхньому кутку програми знаходяться кнопка перемикання кольорової теми програми (світлої або темної) та кнопка перемикання мови.

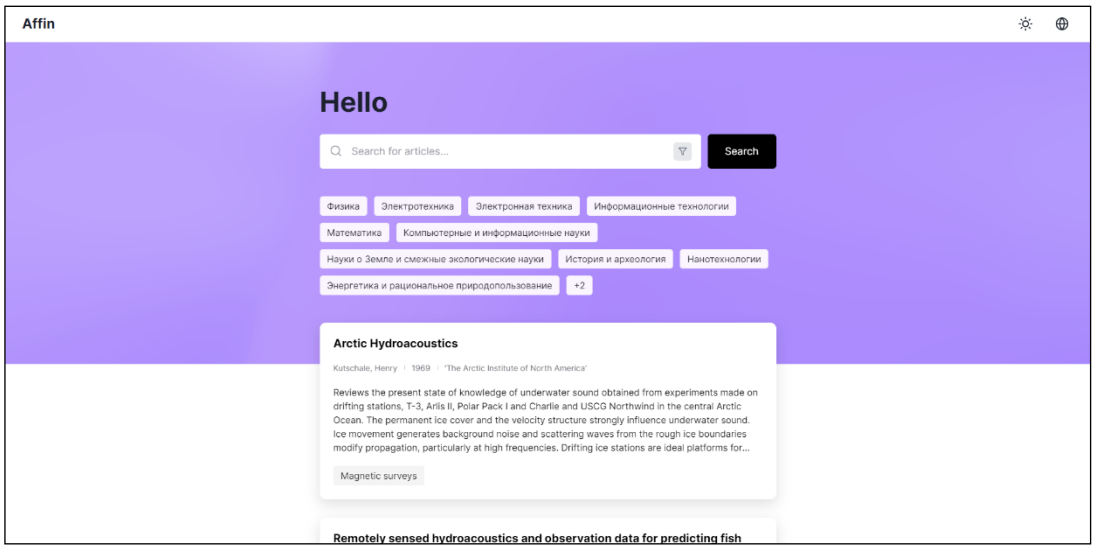

Рисунок 5.1.1 – Головна сторінка програми

Для створення пошукового запиту необхідно ввести ключові слова в відповідне поле і натиснути кнопку ‘Пошук’. Також користувач може відкриті меню з додатковими фільтрами, натиснувши на сіру кнопку з зображенням фільтра всередині поля вводу.

Відкриється додаткове меню, всередині якого користувач може обрати додаткові пошукові фільтри, а саме ввести імена авторів публікації, назву видавця, теми публікації та діапазон років публікації.

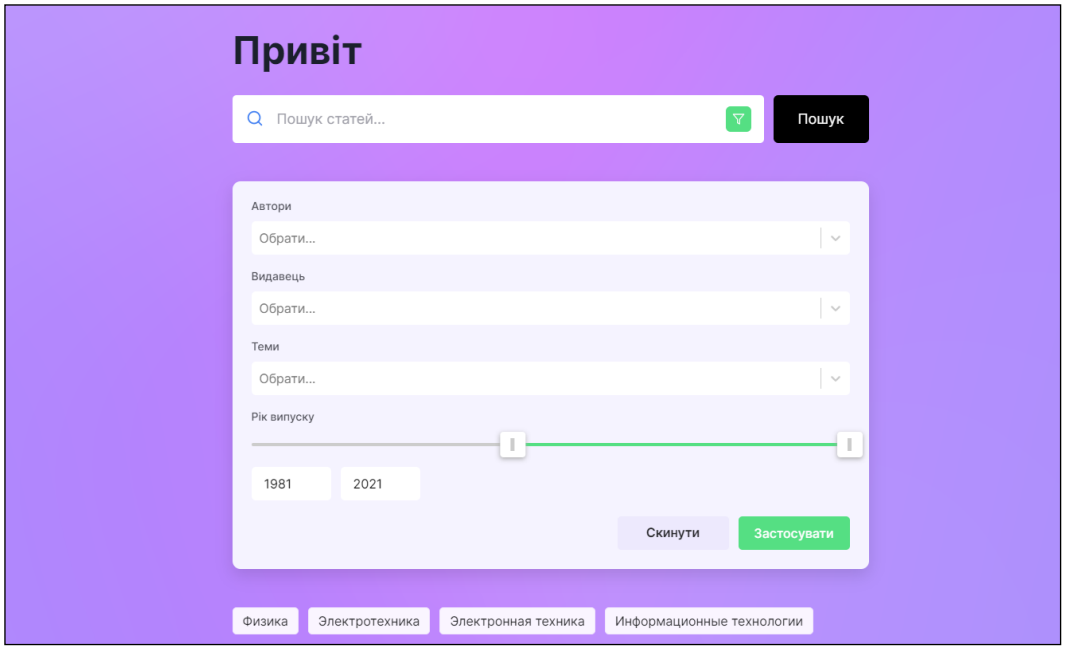

Рисунок 5.1.2 – Меню вибору додаткових фільтрів

Ці фільтри не є обов’язковими для заповнення, але можуть дати більш точні результати. Після налаштування фільтрів користувач може натиснути кнопку ‘Застосувати’, щоб оновити результати пошуку з обраними фільтрами.

Також користувач може натиснути на будь-яку з тем, що знаходяться під полем ввода. Обрана тема буде автоматично обрана як фільтр для пошуку статей. Цей ж функціонал працює і на картках знайдених наукових статей.

5.2 Результати пошукового запиту

Після того, як пошуковий запит буде виконано, на екран виведеться каталог карток знайдених наукових статей. Кожна картка може складатись з назви публікації, імен авторів, року публікації, видавця, переліку тем публікації та короткого опису. Якщо будь-який з атрибутів не заданий, відповідна інформація на картці відображена не буде.

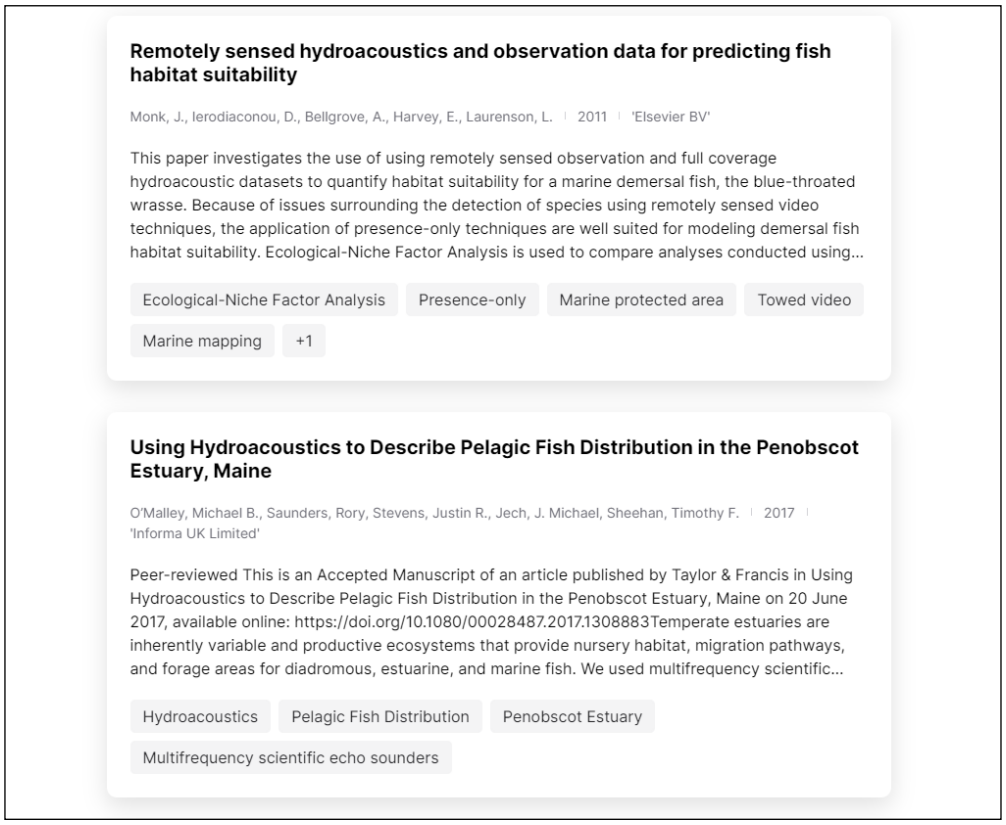

Рисунок 5.2.1 – Знайдені накові статі у вигляді катологу карток

Якщо наданий опис містить занадто багато інформації, він буде укорочений до 5 рядків. Якщо тем публікації більше п'яти, решта тем буде схована. Їх можна знов відобразити, натиснувши відповідну кнопку в кінці тем. Повторне натискання кнопки знов заховає решту тем.

Користувач може натиснути на назву публікації і відкрити її повний текст у новій вкладці браузера.

5.3 Сторінка пошуку

Після вводу пошукового запиту, програма надсилає обрані параметри до серверу. Розроблена система автоматично виконає паралельний пошук по багатьом базам даних.

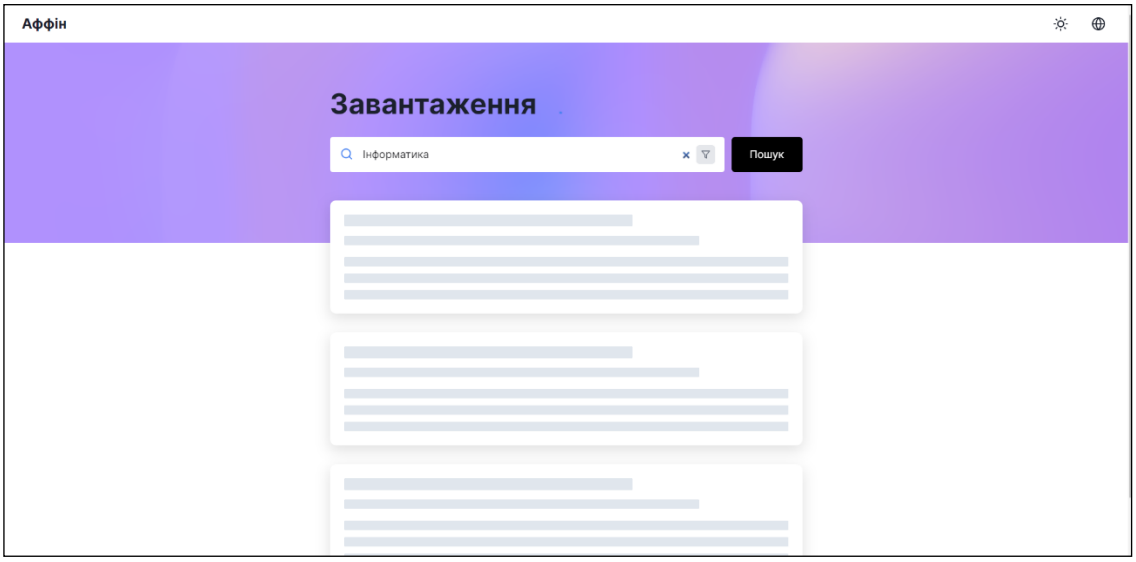

Рисунок 5.3.1 – Вигляд програми, коли пошуковий запит ще в процессі виконання

Коли дані завантажуються, на місці знайдених статей відображаються відповідні тимчасові картки, що показують, що процес ще в процесі виконання. І над полем вводу відображається відпопідне слово “Завантаження”. При цьому саме поле ввода та модуль з фільтрами активні і користувач може змінити/оновити пошуковий запит, не чекаючи завершення виконання попереднього запиту.

Якщо пошуковий запит виконано успішно і були повернені знайдені наукові статі, вони будуть виведені у вигляді каталогу карток. При цьому над полем вводу виведеться загальна кількість знайдених статей.

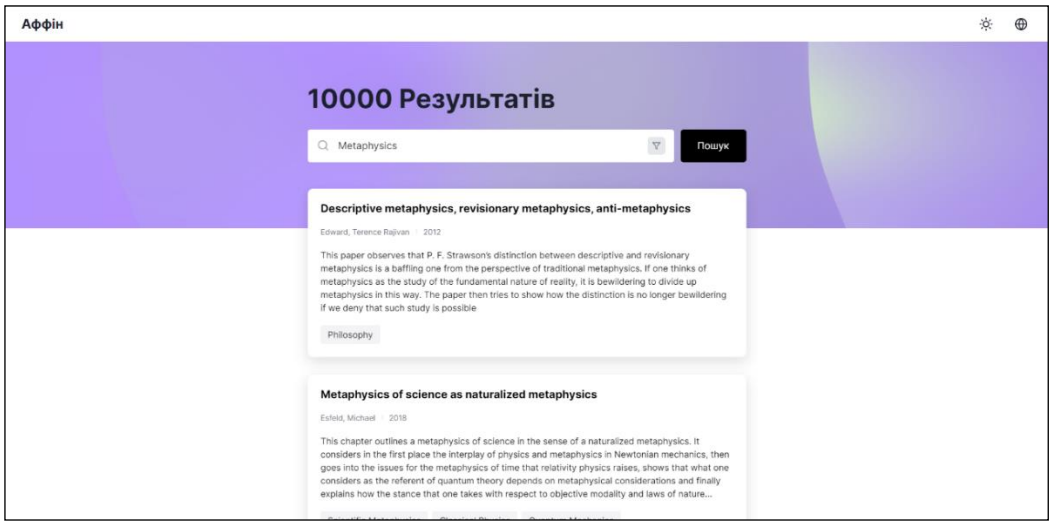

Рисунок 5.3.2 – Вивід результатів пошукового запиту

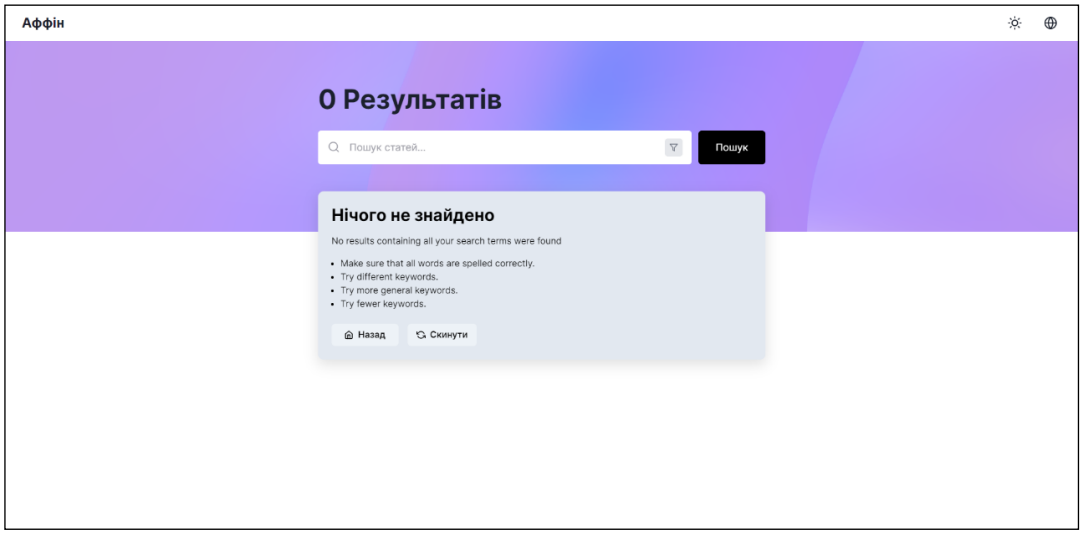

Рисунок 5.3.3 – Вигляд програми, коли пошук не дав результатів

Якщо пошуковий запит не дав результатів, буде відображений відповідний блок з пошуковими рекомендаціями та кнопками, що повертають користувача на головну сторінку чи скидують обрані пошукові фільтри.

5.4 Додаткові екрани

Для більшої зручності використання програма надає додаткові екрани, коли користувач не вказав жодних параметрів для пошукового запиту, пошуковий запит не дав результатів або коли на сервері виникла помилка та запит не може бути завершений.

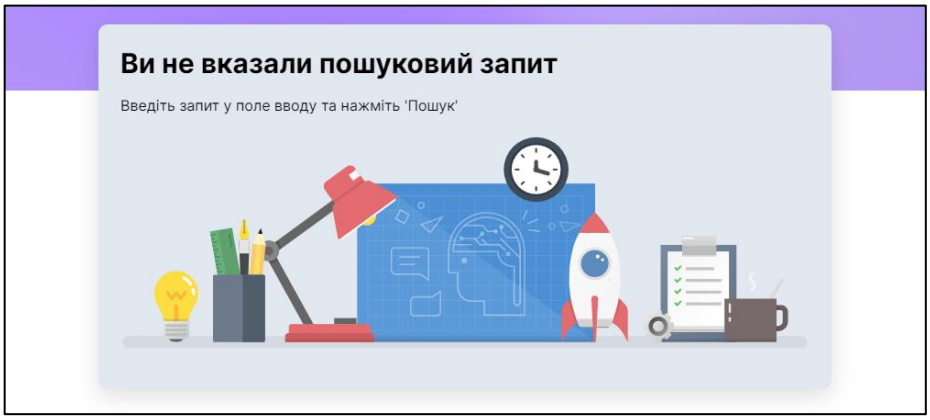

Рисунок 5.4.1 – Екран пустого пушокового запиту

Рисунок 5.4.2 – Вигляд пустої сторінки 404

При ідеальній роботі веб-сервісу користувач може не побачити жодного додаткового екрану, але ми все одно реалізуємо ці компоненти на випадок неочікуваних труднощів чи помилок. При виникненні таких ситуацій буде відображена відповідна картка з описом проблеми та її можливими рішеннями.

ВИСНОВКИ

Під час проходження практики я закріпив та поглибив теоретичні знання і практичні навички, набуті мною під час навчання. Я ознайомився та проаналізував існуючі системи, що реалізують пошук по науковим базам. Був проведений аналіз можливостей адаптації цих систем до поставленої задачі. Я застосував здобуті знання та набув навичок розробки та проектування сучасних клієнтських веб програм, формування і редагування пошукових запитів, відображення результатів пошуку у вигляді каталогу.

В результаті була розроблена програма на мові програмування Typescript, що реалізує клієнтську “frontend” частину системи. Програма працює в усіх сучасних браузерах, доступна онлайн за посиланням та не потребує реєстрації.

Всі основні вимоги до програмного продукту були виконані. Програма надає доступ до модульного паралельного пошуку за фільтрами та ключовим словом по декількох наукових базах. Користувач може виконувати пошук за ключовим словом, уточнювати авторів, видавництво, теми та роки публікації. Результати пошуку відображаються у вигляді зручного для взаємодії каталогу карток наукових статей.

Програма відповідає сучасним вимогам до програмного забезпечення та побудована за допомогою найбільш популярних та оптимальних технологій. Не використовує ресурси, захищені авторським правом, та не потребує ліцензії.

Розроблений графічний інтерфейс можна легко модифікувати та додати нові фільтри, виводити додаткову інформацію. Проект не потребує додаткового налаштування, сумісний з технологією Serverless хостингу та може бути завантажений на будь-який безкоштовний онлайн сервіс хостингу веб-сайтів.

Клієнт реалізує агностичний підхід до технологій серверної частини та не потребує додаткових змін в коді для підключення нових баз даних для пошуку.

Інтернет запроси між клієнтською частиною та сервером реалізовані на базі технології REST.

Веб-сайт працює на всіх сучасних версіях веб-браузерів: Chrome, Firefox, Edge, Opera, Safari. Основуючись на світовій статистиці використання браузерів, поточна очікувана підтримка програми на пристроях – 95%. Інтерфейс оптимізовано під великі та малі екрани та має коректно працювати на мобільних пристроях, планшетах та комп’ютерах.

Клієнт підтримує світлу та темну теми, вибір української, англійської чи російської мов інтерфейсу.

Розроблений проект можна використовувати як базу для подальших, більш детальних розробок у сфері пошуку наукових статей. Система може бути інтегрована у роботу вищих навчальних закладів та лабораторій. Відповідна тема може бути обрана як тема для подальшої дипломної роботи над системою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пошукова база Scopus – https://www.scopus.com/ – Інтернет ресурс.
2. Бібліотека CORE – https://core.ac.uk/ – Інтернет ресурс.
3. Цифрова платформа JSTOR - https://www.jstor.org/ - Інтернет ресурс.
4. Google Scholar - https://scholar.google.com/ - Інтернет ресурс.
5. Електронна бібліотека Кіберленінка - https://cyberleninka.ru/ - Інтернет ресурс.
6. Шокін Ю.І., Федотов А.М., Барахнін В.Б. - Проблеми пошуку інформації. – Новосибірск: Наука, 2010. – 220 с.
7. Костров Б.В., Борисов С.Г., Кострова Ю.Б. - Обґрунтування вибору користувацького інтерфейсу для інформаційної пошукової системи - Санкт-Петербург : НП-Прінт, 2015, - 4 с.
8. Кузьменко Г.Є., Литвинов В.А., Майстренко С.Я., Оксанич І.Н. - Інтелектуальний інтерфейс користувача інформаційно-пошукової системи в задачі пошуку за ключовим словом – Математичні машини і системи, 2011 – 11 с.
9. Ковалев І.В., Джиоєва Н.Н., Карасєва М.В., - Кросплатформна пошукова мультиагентна система – Твер: Програмні продукти і системи, 2008 – 24 с.

ДОДАТОК А

Формування пошукових запитів та архітектура взаємодії з системою пошуку наукових статей

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР7191_05Б

Аркушів 2

Київ — 2021

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР7191_05Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР7191_05Б 12-1	package.json	Компонент, що містить перелік залежностей та їх версії
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР7191_05Б 12-2	index.tsx	Головна сторінка програми
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР7191_05Б 12-3	search.tsx	Сторінка пошукових запитів програми

ДОДАТОК Б

Формування пошукових запитів та архітектура взаємодії з системою пошуку наукових статей

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР7191_05Б 12-1

Аркушів 27

Київ — 2021

package.json

```
{
  "name": "affin",
  "version": "1.0.0",
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "eslint . --ext js,jsx,ts,tsx --max-warnings=0"
  },
  "dependencies": {
    "@chakra-ui/react": "^1.6.3",
    "@emotion/react": "^11.4.0",
    "@emotion/styled": "^11.3.0",
    "@react-three/fiber": "^6.2.2",
    "axios": "^0.21.1",
    "framer-motion": "4.1.17",
    "locomotive-scroll": "^4.1.1",
    "next": "10.2.3",
    "next-i18next": "^8.3.0",
    "react": "17.0.2",
    "react-dom": "17.0.2",
    "react-icons": "^4.2.0",
    "react-query": "^3.16.0",
    "react-range": "^1.8.8",
    "react-select": "^4.3.1",
    "recoil": "^0.3.1",
    "three": "^0.129.0"
  },
  "devDependencies": {
    "@types/node": "^15.6.1",
    "@types/react": "^17.0.8",
    "@types/react-dom": "^17.0.5",
    "@types/three": "^0.128.0",
```

```
    "@typescript-eslint/eslint-plugin": "^4.25.0",
    "@typescript-eslint/parser": "^4.25.0",
    "eslint": "^7.27.0",
    "eslint-config-alloy": "^4.1.0",
    "eslint-plugin-jsx-a11y": "^6.4.1",
    "eslint-plugin-react": "^7.24.0",
    "next-compose-plugins": "^2.2.1",
    "typescript": "4.3.2"
  }
}
```

index.tsx

```
const IndexPage: NextPage = () => {
  const { t } = useTranslation("common");
  const containerRef = React.useRef<HTMLDivElement>(null);

  const { isLoading, error, articles, total, fetchMore } = useHot();
  const { active } = useSearch();

  return (
    <LocomotiveScrollProvider
      options={{
        smooth: true,
      }}
      watch={[articles, active, isLoading]}
      containerRef={containerRef}
    >
      <Layout ref={containerRef}>
        <Head>
          <title>{t("affin search engine")}</title>
        </Head>

        <Hero>
          <Heading as="h1" size="2xl">
```

```
            {t("hello")}
          </Heading>
          <SearchBar />
          <Topics />
        </Hero>
        <div data-fix-width>
          {error && <ErrorCard error={error} />}
          <ArticleCardsList
            loading={isLoading}
            articles={articles}
            total={total}
            fetchMore={fetchMore}
          />
        </div>
      </Layout>
    </LocomotiveScrollProvider>
  );
};
```

search.tsx

```
interface SearchPageProps {}
const SearchPage: NextPage<SearchPageProps> = () => {
  const containerRef = React.useRef<HTMLDivElement>(null);
  const { query } = useRouter();
  const { setSearch, setActive, setFilters, active } = useSearch();
  const { t } = useTranslation("common");

  React.useEffect(() => {
    const { search, ...filters } = parseQuery(query);
    const active = Object.values(filters).some((f) =>
      Array.isArray(f) ? f.length > 0 : Boolean(f),
    );
```

```
    filters.publishedAfter = filters.publishedAfter || new
Date().getFullYear() - 40;
    filters.publishedBefore = filters.publishedBefore || new
Date().getFullYear();

    setSearch(search);
    setActive(active);
    setFilters(filters);
  }, [query]);

  const { isLoading, error, articles, total, enabled, fetchMore } =
useArticles();

  const title = Array.isArray(query.q) ? query.q[0] : query.q;

  return (
    <LocomotiveScrollProvider
      options={{
        smooth: true,
      }}
      watch={[articles, active, isLoading]}
      containerRef={containerRef}
    >
      <Layout ref={containerRef}>
        <Head>
          <title>{t("affin search engine title", { title: title ? `${title} |
` : "" })}</title>
        </Head>

        <Hero>
          <Heading as="h1" size="2xl">
            {isLoading ? (
              <>
                {t("loading")} <Spinner />
```

```
              </>
            ) : (
              <>{t("found", { amount: round(total) })}</>
            )}
          </Heading>
          <SearchBar />
        </Hero>
        <div data-fix-width>
          {error && <ErrorCard error={error} />}
          {enabled ? (
            <ArticleCardsList
              loading={isLoading}
              articles={articles}
              total={total}
              fetchMore={fetchMore}
            />
          ) : (
            <NoSearch />
          )}
        </div>
      </Layout>
    </LocomotiveScrollProvider>
  );
};
```

filters.ts

```
export interface Filters {
  authors: string[] | null;
  publishers: string | null;
  topics: string[] | null;
  publishedAfter: number | null;
  publishedBefore: number | null;
}
```

```
export interface SearchFilters extends Filters {
  search: string | null;
  page?: number;
}
```

Article.ts

```
interface IArticle {
  id: string;
  title: string;
  authors: string[];
  year: number;
  publisher: string;
  description: string;
  topics: string[];
  downloadUrl: string;
}
```

search.ts

```
import { atom } from "recoil";
import { Filters } from "interfaces/filters";

export const searchSearchState = atom<string | null>({
  key: "searchSearchState",
  default: "",
});

export const searchFiltersState = atom<Filters>({
  key: "searchFiltersState",
  default: {
    authors: null,
    publishers: null,
    topics: null,
    publishedAfter: null,
    publishedBefore: null,
```

```
  },
});

export const searchFiltersOpenState = atom<boolean>({
  key: "searchFiltersOpenState",
  default: false,
});
```

useSearchParams.ts

```
import { useRouter } from "next/router";
import { Filters, SearchFilters } from "interfaces/filters";

const checkArray = (value: string | string[]): string[] =>
  Array.isArray(value) ? value : value ? [value] : null;
const checkString = (value: string | string[]): string | null =>
  Array.isArray(value) ? value[0] : value || null;
const checkNumber = (value: string | string[]): number | null => {
  const n = Array.isArray(value) ? value[0] : value;
  const num = Number.parseInt(n, 10);
  return Number.isNaN(num) ? null : num;
};

export const parseQuery = (query: { [x: string]: string | string[] }):
SearchFilters => {
  try {
    const search = checkString(query.q) || "";
    const authors = checkArray(query.authors);
    const publishers = checkString(query.publishers);
    const topics = checkArray(query.topics);
    const publishedAfter = checkNumber(query.publishedAfter) || null;
    const publishedBefore = checkNumber(query.publishedBefore) || null;
    const page = checkNumber(query.page) || 0;
```

```
    return { search, authors, publishers, topics, publishedAfter,
publishedBefore, page };
  } catch {
    return {
      search: null,
      authors: null,
      publishers: null,
      topics: null,
      publishedAfter: null,
      publishedBefore: null,
      page: 0,
    };
  }
};

interface Params {
  search: string | null;
  filters: Filters;
  active: boolean;
  page: number;
}
const useSearchParams = (): Params => {
  const { query } = useRouter();
  const { search, page, ...filters } = parseQuery(query);
  const active = Object.values(filters).some((f) => (Array.isArray(f) ?
f.length > 0 : Boolean(f)));

  return {
    search,
    filters,
    active,
    page,
  };
};
```

useSearch.ts

```
import { useRecoilState } from "recoil";
import { useRouter } from "next/router";
import { searchSearchState, searchFiltersOpenState, searchFiltersState } from
"state/search";

const useSearch = () => {
  const router = useRouter();

  const [search, setSearch] = useRecoilState(searchSearchState);
  const [active, setActive] = useRecoilState(searchFiltersOpenState);
  const [filters, setFilters] = useRecoilState(searchFiltersState);

  const searchArticles = () => {
    const query = { q: search, ...(active ? filters : {}) };

    router.push({
      pathname: "/search",
      query,
    });
  };

  const resetFilters = () =>
    setFilters({
      authors: null,
      publishers: null,
      topics: null,
      publishedAfter: new Date().getFullYear() - 40,
      publishedBefore: new Date().getFullYear(),
    });

  return {
    search,
    setSearch,
```

```
    active,
    setActive,
    filters,
    setFilters,
    searchArticles,
    resetFilters,
  };
};
```

useArticles.ts

```
import axios, { AxiosResponse } from "axios";
import { useInfiniteQuery } from "react-query";
import useSearchParams from "utils/useSearchParams";
import { Filters } from "interfaces/filters";
import IArticle from "interfaces/IArticle";

interface IFilters {
  topics?: string;
  authors?: string;
  publishers?: string;
  years?: string;
}

interface APIResponseOK {
  data: IArticle[];
  status: "OK";
  total: number;
}
interface APIResponseError {
  data: null;
  status: "ERROR";
  total: null;
}
export type APIResponse = APIResponseOK | APIResponseError;
```

```
const queryToKey = ({ search, filters }: { search: string; filters: Filters |
null }) => {
  const topics = getUniqueTrimmed(filters?.topics) || null;
  const authors = getUniqueTrimmed(filters?.authors) || null;
  const publishers = filters?.publishers?.trim() || null;
  const years =
    [filters?.publishedAfter,
filters?.publishedBefore].filter(Boolean).join("-") || null;

  const searchQuery = search || publishers || topics?.[0] || authors?.[0] ||
years;

  const topicsString = topics?.join("|");
  const authorsString = authors?.join("|");

  let filter: IFilters = {};
  if (topicsString) filter.topics = topicsString;
  if (authorsString) filter.authors = authorsString;
  if (publishers) filter.publishers = publishers;
  if (years) filter.years = years;

  return { search: searchQuery, filters: Object.keys(filter).length === 0 ?
null : filter };
};

export const getArticles = async (
  query: { search: string; filters: Filters | null },
  limit = 20,
  offset = 1,
) => {
  const { search, filters } = queryToKey(query);
```

```
  return axios.post<APIResponse>("https://kpi-affin-
2021.herokuapp.com/pullarticles/", {
    searchQuery: search,
    offset,
    limit,
    filter: filters,
  });
};

const getUniqueTrimmed = (values: string[] | null) =>
  values ? Array.from(new Set(values.map((v) => v?.trim()))) : values;

const getScore = (article: IArticle) => {
  let score = 0;
  if (article.description?.trim()) score += 2;
  if (article.topics?.length > 0) score += 1;
  return score;
};
const sortCards = (articles: IArticle[]) => articles?.sort((a, b) =>
getScore(b) - getScore(a));

export const getUniqueArticles = (articles: IArticle[]): IArticle[] => {
  if (!articles) return articles;

  const urls = [];
  const res = [];
  for (let article of articles) {
    if (article.downloadUrl && !urls[article.downloadUrl]) {
      res.push(article);
      urls[article.downloadUrl] = true;
    }
  }

  return sortCards(
```

```
    res.map((article) => ({
      ...article,
      authors: getUniqueTrimmed(article.authors),
      topics: getUniqueTrimmed(article.topics),
    })),
  );
};

const useArticles = () => {
  const { search, active, filters } = useSearchParams();

  const enabled = Boolean(search?.trim() || active);

  const { data, error, ...props } =
useInfiniteQuery<AxiosResponse<APIResponse>>(
    ["articles", queryToKey({ search, filters })],
    ({ pageParam = 0 }) => getArticles({ search, filters }, 20, pageParam),
    {
      getNextPageParam: (_lastPage, pages) => pages.length,
      enabled,
    },
  );

  const articles: IArticle[] =
    data?.pages?.map((page) => getUniqueArticles(page?.data?.data)).flat() ||
null;
  const total: number = data?.pages?.[0]?.data?.total || 0;

  const typedError = error as {
    response: {
      status: string;
      statusText: string;
    };
  };
```

```
  return { articles, total, error: typedError, enabled, ...props };
};
```

getPublishers.ts

```
const getPublishers = async (inputValue: string): Promise<{ value: string;
label: string }[]> => {
  const result = await axios.get(
    `https://kpi-affin-
2021.herokuapp.com/publishers/?term=${encodeURIComponent(inputValue || "")}`,
  );
  const publishers: string[] = result?.data?.publishers || [];
  return publishers.map((author) => ({ value: author, label: author }));
};
```

getAuthors.ts

```
const getAuthors = async (inputValue: string): Promise<{ value: string;
label: string }[]> => {
  const result = await axios.get(
    `https://kpi-affin-
2021.herokuapp.com/authors/?term=${encodeURIComponent(inputValue || "")}`,
  );
  const authors: string[] = result?.data?.authors || [];
  return authors.map((author) => ({ value: author, label: author }));
};
```

SearchBar.ts

```
interface SearchBarProps {}
const SearchBar: React.FC<SearchBarProps> = () => {
  const { search, setSearch, active, setActive, searchArticles } =
useSearch();
  const { t } = useTranslation("common");
```

```
const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {
  e?.preventDefault();
  searchArticles();
};

const toggleAdvancedOptions = (e) => {
  e?.stopPropagation();
  setActive((prev) => !prev);
};

return (
  <StyledForm onSubmit={handleSubmit}>
    <div className="search-input">
      <StyledInputGroup>
        <InputLeftElement pointerEvents="none">
          <FiSearch />
        </InputLeftElement>
        <Input
          id="search-input"
          type="search"
          placeholder={t("search articles")}
          required
          value={search}
          onChange={(e) => setSearch(e.target.value)}
          spellCheck="false"
          aria-autocomplete="list"
          autoComplete="off"
        />
        <InputRightElement>
          <Tooltip
            label={t("advanced options")}
            aria-label={t("advanced options")}
            placement="top"
          >
```

```
              <Button
                onClick={toggleAdvancedOptions}
                aria-label={t("advanced options")}
                type="button"
                data-active={active}
              >
                <FiFilter />
              </Button>
            </Tooltip>
          </InputRightElement>
        </StyledInputGroup>
        <SubmitButton type="submit">{t("search")}</SubmitButton>
      </div>
      <Collapse in={active} animateOpacity>
        <Filters />
      </Collapse>
    </StyledForm>
  );
};
```

Filters.ts

```
interface FiltersProps {}
const Filters: React.FC<FiltersProps> = () => {
  const { t } = useTranslation("common");
  const { filters, setFilters, searchArticles, resetFilters } = useSearch();
  const { colorMode } = useColorMode();

  const handleChange = (name: string, value: string | string[] | number) =>
    setFilters((prev) => ({ ...prev, [name]: value }));

  const applyChanges = () => searchArticles();

  React.useEffect(() => {
    if (!filters.publishedAfter) {
```

```
      setFilters((f) => ({ ...f, publishedAfter: new Date().getFullYear() -
40 }));
    }
  }, [filters.publishedAfter]);
  React.useEffect(() => {
    if (!filters.publishedBefore) {
      setFilters((f) => ({ ...f, publishedBefore: new Date().getFullYear()
}));
    }
  }, [filters.publishedBefore]);

  return (
    <Card data-theme={colorMode}>
      <Select
        label={t("authors")}
        isMulti
        value={filters.authors || []}
        onChange={(value) => handleChange("authors", value)}
        instanceId="authors"
        promiseOptions={getAuthors}
      />
      <Select
        label={t("publisher")}
        value={filters.publishers}
        onChange={(value) => handleChange("publishers", value)}
        instanceId="publisher"
        promiseOptions={getPublishers}
        isClearable
      />
      <Select
        label={t("topics")}
        isMulti
        value={filters.topics || []}
        onChange={(value) => handleChange("topics", value)}
```

```
        instanceId="topics"
        promiseOptions={getTopics}
      />
      <Year
        value={[
          filters.publishedAfter || new Date().getFullYear() - 40,
          filters.publishedBefore || new Date().getFullYear(),
        ]}
        onChange={(years) => {
          handleChange("publishedAfter", years[0]);
          handleChange("publishedBefore", years[1]);
        }}
        label={t("year")}
      />

      <Actions>
        <Button type="button" className="secondary" onClick={resetFilters}>
          {t("reset")}
        </Button>
        <Button type="button" className="primary" onClick={applyChanges}>
          {t("apply")}
        </Button>
      </Actions>
    </Card>
  );
};
```

Search.ts

```
interface Value {
  label: string;
  value: string;
}
interface SelectProps {
  label: React.ReactNode;
```

```
  isMulti?: boolean;
  value?: string | string[];
  onChange?: (value: string | string[]) => void;
  promiseOptions?: (inputValue: string) => Promise<{ value: string; label:
string }[]>;
  instanceId?: string;
  placeholder?: string;
  isClearable?: boolean;
}
const Select: React.FC<SelectProps> = ({
  label,
  isMulti = false,
  value,
  onChange,
  instanceId,
  placeholder,
  promiseOptions,
  isClearable,
}) => {
  const { t } = useTranslation("common");

  let v: Value | Value[] = null;
  if (isMulti && Array.isArray(value)) {
    v = value.map((v) => ({ label: v, value: v }));
  } else {
    if (Array.isArray(value)) {
      v = { label: value[0], value: value[0] };
    } else if (value) {
      v = { label: value, value: value };
    }
  }

  return (
    <StyledLabel>
```

```
      <LabelTitle>{label}</LabelTitle>
      <AsyncCreatableSelect
        cacheOptions
        defaultOptions
        loadOptions={promiseOptions}
        classNamePrefix="react-select"
        allowCreateWhileLoading
        formatCreateLabel={(inputValue: any) => inputValue}
        isMulti={isMulti}
        value={v}
        onChange={(v: Value | Value[] | null) => {
          const selected = Array.isArray(v) ? v.map((v) => v.value) :
v?.value;
          onChange(selected);
        }}
        instanceId={instanceId}
        placeholder={placeholder || t("select...")}
        isClearable={isClearable}
      />
    </StyledLabel>
  );
};
```

Year.ts

```
const clamp = (min: number, num: number, max: number) =>
Math.min(Math.max(num, min), max);

interface YearProps {
  value: [number, number];
  onChange: (data: [number, number]) => void;
  min?: number;
  max?: number;
  step?: number;
  label?: React.ReactNode;
```

```
}
const Year: React.FC<YearProps> = ({
  value,
  onChange,
  min = 1950,
  max = new Date().getFullYear(),
  step = 1,
  label,
}) => {
  const rangeRef: any = React.useRef<Range>();

  return (
    <div>
      <LabelTitle>{label}</LabelTitle>
      <Wrapper>
        <Range
          ref={rangeRef}
          values={value.map((year) => clamp(min, year, max)).sort((a, b) => a
- b)}
          step={step}
          min={min}
          max={max}
          onChange={onChange}
          renderTrack={({ props, children }) => (
            <Track
              onMouseDown={props.onMouseDown}
              onTouchStart={props.onTouchStart}
              style={props.style}
            >
              <div
                ref={props.ref}
                style={{
                  background: getTrackBackground({
                    values: value,
```

```
                    colors: getColors(value.length),
                    min,
                    max,
                  }),
                }}
              >
                {children}
              </div>
            </Track>
          )}
          renderThumb={({ props, isDragged }) => (
            <Thumb {...props} style={props.style}>
              <div className="thumb-block" data-is-dragged={isDragged} />
            </Thumb>
          )}
        />
      </Wrapper>

      <YearInput values={value} onChange={onChange} min={min} max={max} />
    </div>
  );
};
```

ArticleCard.ts

```
const useParams = (article: IArticle): IArticle =>
  React.useMemo(
    () =>
      article
        ? {
            ...article,
            authors: article.authors?.map(trim),
            publisher: trim(article.publisher),
            topics: article.topics?.map(trim),
          }
```

```
        : article,
    [article],
  );

interface ArticleCardProps {
  article: IArticle;
}
const ArticleCard: React.FC<ArticleCardProps> = ({ article, ...props }) => {
  const { colorMode } = useColorMode();

  const title = React.useMemo(() => stripHtml(article?.title),
[article?.title]);
  const description = React.useMemo(() => stripHtml(article?.description),
[article?.description]);

  const { authors, year, publisher, topics, downloadUrl } =
useParams(article) || {};
  if (!article) return null;

  return (
    <Card as="article" data-theme={colorMode} {...props}>
      <Heading as="h3" fontSize="xl">
        <a href={downloadUrl} target="_blank" rel="noopener noreferrer">
          {title}
        </a>
      </Heading>
      <Info className="info">
        {authors?.length > 0 && <span>{authors.join(", ")}</span>}
        {year && (
          <>
            <Divider />
            <Link
href={`/search?publishedAfter=${year}&publishedBefore=${year}`} passHref>
              <a>{year}</a>
```

```
            </Link>
          </>
        )}
        {publisher && (
          <>
            <Divider />
            <Link
href={`/search?publishers=${encodeURIComponent(publisher)}`} passHref>
              <a>{publisher}</a>
            </Link>
          </>
        )}
      </Info>
      {description && <Description>{description}</Description>}
      <StyledList topics={topics} maxTags={5} spacing="8px" />
    </Card>
  );
};
```

ArticleCardList.ts

```
interface ArticleCardsListProps {
  articles: IArticle[];
  loading?: boolean;
  total: number;
  fetchMore?: () => void;
  isFetchingNextPage?: boolean;
}
const ArticleCardsList: React.FC<ArticleCardsListProps> = ({
  articles,
  loading = false,
  total,
  fetchMore,
  isFetchingNextPage = false,
  ...props
```

```
}) => {
  const { t } = useTranslation("common");

  if (!articles || articles.length === 0) {
    if (loading) {
      return (
        <Stack spacing="clamp(16px, 5vw, 32px)" {...props}>
          <ArticleCardPlaceholder />
          <ArticleCardPlaceholder />
          <ArticleCardPlaceholder />
        </Stack>
      );
    }
    return <NotFound />;
  }

  return (
    <Stack spacing="clamp(16px, 5vw, 32px)" as="ul" {...props}>
      {articles.map((article) => (
        <li key={article.id || `${article.title}-${article.publisher}`}>
          <Fade in>
            <ArticleCard article={article} />
          </Fade>
        </li>
      ))}
      {articles.length < total && fetchMore && (
        <Button onClick={fetchMore} isLoading={isFetchingNextPage}>
          {t("fetch more")}
        </Button>
      )}
    </Stack>
  );
};
```

NotFound.ts

```
interface NotFoundProps {}
const NotFound: React.FC<NotFoundProps> = ({ ...props }) => {
  const { colorMode } = useColorMode();
  const { t } = useTranslation("common");
  const { resetFilters } = useSearch();

  return (
    <Card data-theme={colorMode} {...props}>
      <Heading size="lg" mb="16px">
        {t("nothing found")}
      </Heading>
      <Text mb="16px">{t("no results containing your search")}</Text>

      <UnorderedList mb="24px">
        <ListItem>{t("make sure that all words are spelled
correctly")}</ListItem>
        <ListItem>{t("try different keywords")}</ListItem>
        <ListItem>{t("try more general keywords")}</ListItem>
        <ListItem>{t("try fewer keywords")}</ListItem>
      </UnorderedList>

      <div className="actions">
        <Link href="/" passHref>
          <Button as="a" leftIcon={<FiHome />}>
            {t("home")}
          </Button>
        </Link>
        <Button onClick={resetFilters} leftIcon={<FiRefreshCcw />}>
          {t("reset")}
        </Button>
      </div>
    </Card>
  );
```

```
};
```

ErrorCard.ts

```
interface ErrorCardProps {
  error: { response: { status: string; statusText: string } };
}
const ErrorCard: React.FC<ErrorCardProps> = ({ error, ...props }) => {
  const { colorMode } = useColorMode();
  const { t } = useTranslation("common");

  if (!error?.response) return null;
  return (
    <Card data-theme={colorMode} {...props} mb="clamp(16px, 5vw, 32px)">
      <Heading size="lg" mb="16px">
        {error.response.status} | {t("something went wrong")}
      </Heading>
      <Text>{error.response.statusText}</Text>
    </Card>
  );
};
```

ДОДАТОК В

Формування пошукових запитів та архітектура взаємодії з системою пошуку наукових статей

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР7191_05Б 13-1

Аркушів 9

Київ — 2021

АНОТАЦІЯ

В цій роботі описана реалізація графічного клієнта для системи, що надає доступ користувачу до багатьох пошукових баз даних одночасно. При цьому користувач може фільтрувати, уточнювати запити та продивлятись результати пошуку. Програма доступна онлайн, не потребує завантаження, встановлення чи авторизації. Програма не є завершеним чи комерційним продуктом, а лише демонструє працездатність описаної концепції.

Мобільний застосунок розроблений мовою програмування Typescript з використанням технології Flutter у редакторі вихідного коду Visual Studio Code.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку міститься опис основних компонентів вебпрограми для роботи з пошуковими запитами науових статей по ключовому слову, що виконує деякі завдання, визначенні в розділі 1. Додаток Б містить програмний код цих компонентів.

Для роботи програми потрібно мати комп’ютерний пристрій з доступом до мережі інтернет та будь-який браузер що підтримує веб стандарт не нижче ES6

Вебсайт розроблений мовою програмування Typescript з використанням технології Next.js у редакторі вихідного коду Visual Studio Code.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений графічний клієнт надає користувачу наступний функціонал:

- Можливість робити пошукові запити по ключовому слову
- Можливість налаштовувати та уточнювати пошукові фільтри
- Можливість перегляди результати пошуку у вигляді каталогу публікацій
- Можливість відкрити повний текст обраної публікації
- Можливість зручно працювати з системою зі свого пристрою
- Високу швидкість роботи та виконання пошукових запитів

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Графічний клієнт використовує зовнішнє API для створення пошукових запитів та виведення результатів пошукових запитів на екран. Програма реалізована як вебпрограма на базі фреймворка Next.js та Typescript.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для роботи графічного клієнта підходять будь-які технічні пристрої з виходом в інтернет, наприклад, мобільні телефони, планшети чи персональні комп’ютери.

ВХІДНІ ДАНІ

Вхідна інформація для формування пошукового запиту:

- Пошуковий рядок (String)
- Автори (Масив String)
- Видавництво (String)
- Теми (Масив String)
- Роки публікації (Діапазон початкова-кінцева дати)

ВИХІДНІ ДАНІ

Вихідна інформація результатів пошукового запиту:

- Назва публікації (String)
- Автори (Масив String)
- Видавництво (String)
- Рік публікації (Number)
- Мова публікації (String)
- Ключові слова (Масив String)
- Посилання на файл публікації (String)