Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА МОДУЛЯ ПАРСИНГУ ДЛЯ ПРОФІЛЮ НПП ЛНТУ З ВИКОРИСТАННЯМ PYTHON, NODE.JS ТА GOOGLE SCHOLAR API

DEVELOPMENT OF A PARSING MODULE FOR THE LNTU FACULTY PROFILE USING PYTHON, NODE.JS AND GOOGLE SCHOLAR API

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Рощук Олександр Романович

Керівник:
Гульчук Юрій Миколайович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Рощуку Олександру Романовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка модуля парсингу для профілю НПП ЛНТУ з використанням Python, Node.JS TA Google Scholar API »*
Керівник роботи: *асистент Гульчук Ю. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Python, Node.JS, Google Scholar API, Visual Studio Code, React, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз існуючих рішень та визначення вимог до модуля, Розробка архітектури модуля для збору даних з профілів НПП ЛНТУ та Google Scholar. Реалізація парсингу профілів НПП ЛНТУ, інтеграцію даних з Google Scholar та їх збереження, тестування розробленого модуля*
5. Перелік графічного матеріалу: *7 рисунків, 2 таблиці*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Гульчук Ю. М.*		
Специфікація вимог до розробленої системи	*Гульчук Ю. М.*		
Розробка об'єкта проектування	*Гульчук Ю. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	2,33 %		
Академічна доброчесність	*Гульчук Ю. М.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Виконано
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування та розробка	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Олександр РОЩУК

Керівник кваліфікаційної роботи _____ Юрій ГУЛЬЧУК

АНОТАЦІЯ

Рощук О. Р. Розробка модуля парсингу для профілю НПП ЛНТУ з використанням Python, Node.js та Google Scholar API. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі проведено аналіз предметної області автоматизації збору та обробки наукометричних даних. У другому розділі описано проектування модуля парсингу. Визначено функціональні та нефункціональні вимоги до системи. У третьому розділі представлено практичну реалізацію розробленого модуля парсингу. Описано процес вилучення даних з веб-сторінок профілів НПП ЛНТУ та інтеграції наукометричних показників з Google Scholar. У висновках підсумовано результати виконаної роботи, підтверджено досягнення поставленої мети щодо розробки ефективного модуля парсингу.

Ключові слова: парсинг, веб-скрапінг, наукометричні дані, Python, Node.JS, Google Scholar, профіль НПП, автоматизація, аналіз даних.

ABSTRACT

Roshchuk O. Development of a parsing module for the lntu faculty profile using Python, Node.js and Google Scholar API. Manuscript. Bachelor´s Qualification Thesis, study program "Software Engineering", specialty 121 "Software Engineering." Lutsk National Technical University. Lutsk, 2025.

The bachelor´s qualification thesis consists of an introduction, three chapters, conclusions, a list of references.

The first chapter analyzes the subject area of automating the collection and processing of scientometric data. Existing approaches and tools for web scraping and parsing academic information are reviewed. The first chapter analyzes the subject area of automating the collection and processing of scientometric data. Existing approaches and tools for web scraping and parsing academic information are reviewed. The third chapter presents the practical implementation of the developed parsing module. The process of extracting data from LNTU faculty profile web pages and integrating scientometric indicators from Google Scholar is described. The conclusions summarize the results of the work performed, confirming the achievement of the set goal to develop an effective parsing module.

Keywords: parsing, web scraping, scientometric data, Python, Node.JS, Google Scholar, faculty profile, automation, data analysis.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному цифровому середовищі все більше зростає потреба в автоматизації процесів збору та обробки наукової інформації. Раніше, щоб зібрати інформацію про публікації певного науковця, потрібно було вручну переглядати різні джерела, зокрема Google Scholar. Проте з розвитком технологій стало можливим створювати модулі, які автоматично витягують необхідні дані, що значно спрощує аналіз наукової активності.

Мета роботи – розробка функціонального модуля парсингу, що здатний отримувати дані профілю НПП ЛНТУ та Google Scholar з використанням мови програмування Python та вебтехнологій Node.js для подальшої обробки або відображення в інтерфейсі.

Об’єкт роботи – технологічні та програмні засоби створення інтернет-магазинів і інтегрування бота месенджера.

Предмет роботи – програмна реалізація модуля збору даних наукового профілю викладача з Google Scholar засобами Python, Node.js.

Для досягнення поставленої мети були поставлені наступні завдання:

– проаналізувати існуючі рішення та методи парсингу наукових профілів;

– визначити вимоги до програмного забезпечення;

– спроєктувати архітектуру системи та взаємодію її компонентів;

– реалізувати модуль парсингу на Python для збору даних з профілів ЛНТУ та Google Scholar;

– створити веб-інтерфейс на React та інтегрувати його з бекендом;

– забезпечити обробку помилок для стабільної роботи модуля;

– протестувати розроблений модуль та оцінити його точність і ефективність.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ МОДУЛЯ ПАРСИНГУ ДЛЯ ПРОФІЛЮ НПП ЛНТУ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

В умовах стрімкої цифровізації всіх сфер життя, особливо освіти та науки, критично зростає потреба у швидкому, надійному та автоматизованому доступі до актуальної інформації про наукову діяльність викладачів та науковців. Одним із найавторитетніших та найпоширеніших джерел такої інформації є міжнародна наукометрична платформа Google Scholar [1]. Вона агрегує мільйони наукових публікацій, дані про їх цитування, розраховує важливі наукометричні індекси (наприклад, індекс Гірша) та надає доступ до індивідуальних профілів науковців з усього світу, де вони можуть керувати списками своїх праць та відстежувати їхню популярність. На рисунку 1.1 зображено вигляд профілю в Google Scholar з посиланнями на наукометричні індекси.

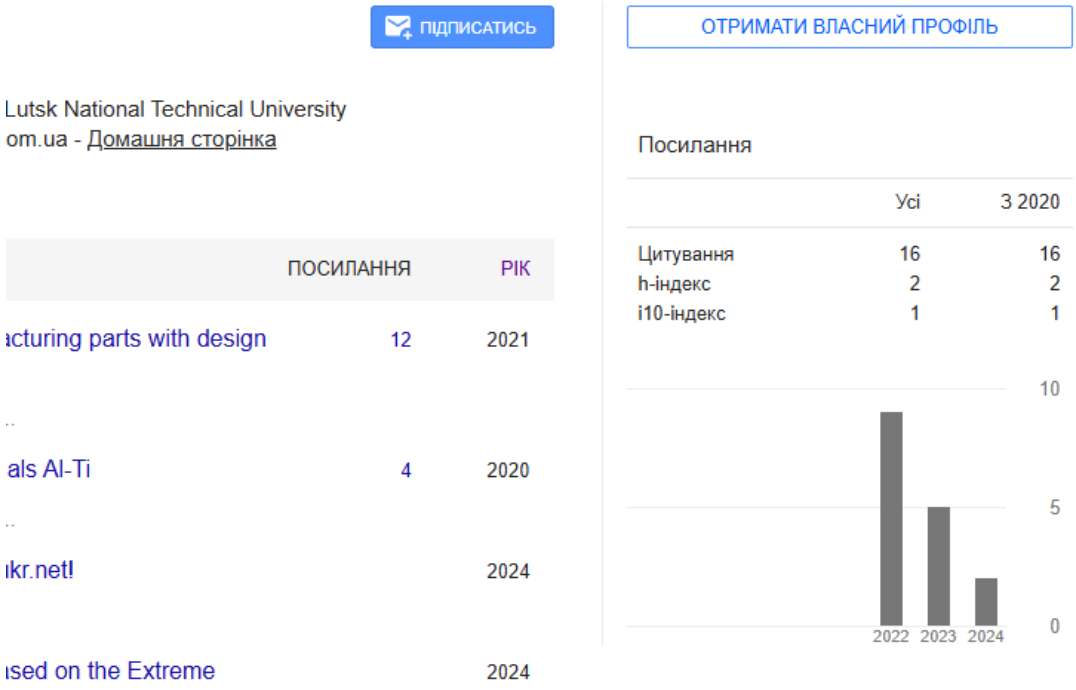
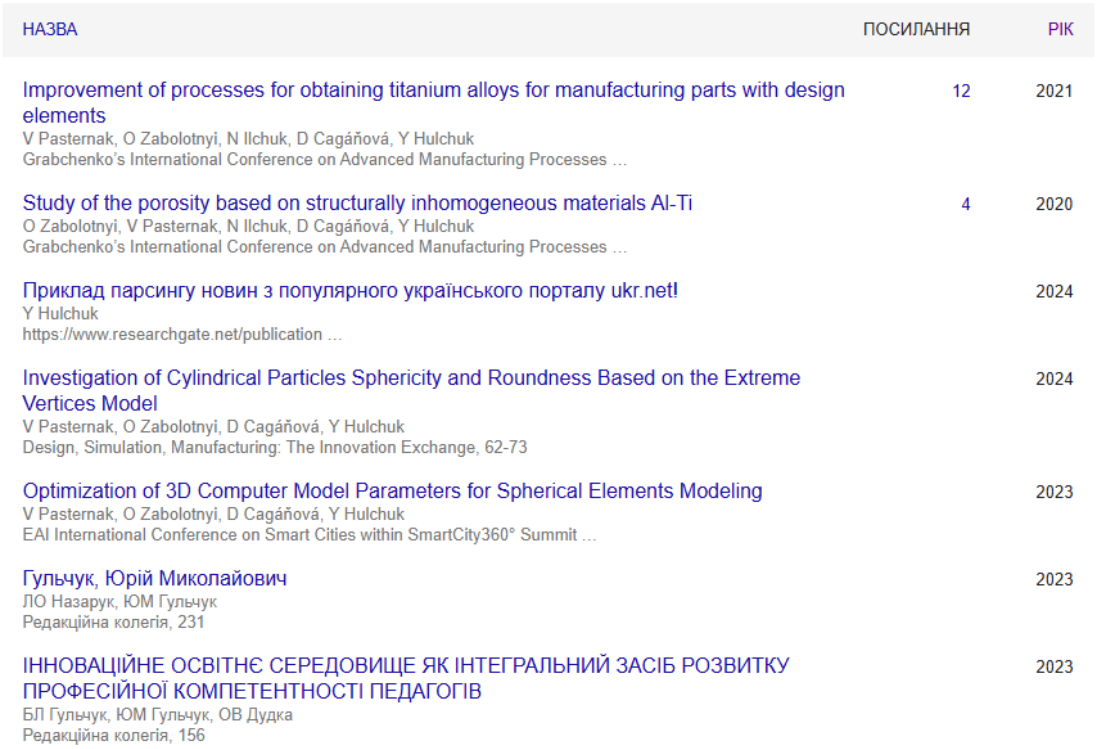


Рисунок 1.1 – Профіль в Google Scholar [2]

Однак, попри свою відкритість для перегляду, Google Scholar не надає офіційного програмного інтерфейсу (API) для автоматизованого масового збору даних. Ця обставина суттєво ускладнює розробку та впровадження систем автоматизованого аналізу наукової активності викладачів, моніторингу наукометричних показників на рівні кафедр, факультетів чи цілих закладів вищої освіти. Відсутність API змушує розробників вдаватися до альтернативних методів, таких як веб-скрапінг та парсинг [3] HTML-сторінок Google Scholar, що пов’язано з низкою технічних труднощів:

– динамічна зміна структури веб-сторінок;
– наявність механізмів захисту від автоматизованих запитів;
– необхідність постійної підтримки та оновлення парсерів.

У багатьох закладах вищої освіти, включно з Луцьким національним технічним університетом [4], збір інформації про публікації викладачів здійснюється вручну або з використанням загальних пошукових інструментів. Це займає багато часу та піддається ризику помилок або неповної інформації. На рисунку 1.2 зображено список статей які показуються в профілі Google Scholar.



НАЗВА	ПОСИЛАННЯ	РІК
Improvement of processes for obtaining titanium alloys for manufacturing parts with design elements V Pasternak, O Zabolotnyi, N Ilchuk, D Cagáňová, Y Hulchuk Grabchenko's International Conference on Advanced Manufacturing Processes ...	12	2021
Study of the porosity based on structurally inhomogeneous materials Al-Ti O Zabolotnyi, V Pasternak, N Ilchuk, D Cagáňová, Y Hulchuk Grabchenko's International Conference on Advanced Manufacturing Processes ...	4	2020
Приклад парсингу новин з популярного українського порталу ukr.net! Y Hulchuk https://www.researchgate.net/publication ...		2024
Investigation of Cylindrical Particles Sphericity and Roundness Based on the Extreme Vertices Model V Pasternak, O Zabolotnyi, D Cagáňová, Y Hulchuk Design, Simulation, Manufacturing: The Innovation Exchange, 62-73		2024
Optimization of 3D Computer Model Parameters for Spherical Elements Modeling V Pasternak, O Zabolotnyi, D Cagáňová, Y Hulchuk EAI International Conference on Smart Cities within SmartCity360° Summit ...		2023
Гульчук, Юрій Миколайович ЛО Назарук, ЮМ Гульчук Редакційна колегія, 231		2023
ІННОВАЦІЙНЕ ОСВІТНЄ СЕРЕДОВИЩЕ ЯК ІНТЕГРАЛЬНИЙ ЗАСІБ РОЗВИТКУ ПРОФЕСІЙНОЇ КОМПЕТЕНТНОСТІ ПЕДАГОГІВ БЛ Гульчук, ЮМ Гульчук, ОВ Дудка Редакційна колегія, 156		2023

Рисунок 1.2 – Список статей в профілі Google Scholar [2]

Сучасні програмні рішення, наприклад Python-бібліотека scholarly, частково вирішують проблему доступу до даних Google Scholar, пропонуючи певний рівень абстракції над прямим парсингом. Результати, отримані за допомогою таких інструментів, часто потребують додаткової ручної верифікації для забезпечення їх точності та релевантності. Також, будь-які інструменти, що покладаються на веб-скрапінг, чутливі до змін у структурі сайту Google Scholar та його анти-скрапінгових політик.

У зв’язку з вищезазначеними проблемами та обмеженнями, надзвичайно актуальним є завдання створення спеціалізованого програмного модуля, здатного ефективно та надійно автоматично витягувати ключову наукометричну інформацію безпосередньо зі сторінок профілів викладачів на платформі Google Scholar. Важливою вимогою до такого модуля є можливість подальшого представлення зібраних даних у структурованому та зручному для машинного зчитування форматі, наприклад, JSON, або через інтуїтивно зрозумілий вебінтерфейс для кінцевих користувачів.

Розробка та впровадження такого спеціалізованого модуля парсингу для потреб ЛНТУ дозволить досягти низки важливих переваг:

– підвищити точність збору публікаційної активності НПП;

– забезпечити можливість інтеграції з локальними базами даних;

– зменшити ручну працю в адміністративних та аналітичних процесах.

Отже, створення ефективного модуля парсингу профілів викладачів ЛНТУ з платформи Google Scholar є не просто технічним завданням, а важливим стратегічним кроком на шляху до цифрової трансформації освітніх та наукових процесів в університеті, забезпечуючи сучасний, точний та прозорий облік наукової діяльності його співробітників.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для досягнення поставленої мети – розробки модуля парсингу для профілю НПП ЛНТУ – необхідно вирішити наступні завдання:

1) проаналізувати існуючі рішення та методи парсингу наукових профілів;

2) визначити вимоги до програмного забезпечення;

3) спроєктувати архітектуру системи та взаємодію її компонентів;

4) реалізувати модуль парсингу на Python для збору даних з профілів ЛНТУ та Google Scholar;

5) створити веб-інтерфейс на React та інтегрувати його з бекендом;

6) забезпечити обробку помилок для стабільної роботи модуля;

7) протестувати розроблений модуль та оцінити його точність і ефективність.

Результатом проєкту має стати готовий до використання модуль парсингу, який можна буде адаптувати до потреб ЛНТУ або інших освітніх закладів України.

Висновки до розділу 1

У результаті проведеного в даному розділі аналізу предметної області та існуючих рішень було чітко встановлено, що попри незаперечну важливість та широке поширення платформи Google Scholar як одного з ключових джерел інформації про наукову активність викладачів та науковців, ефективні та універсальні офіційні інструменти для автоматизованого збору цих даних або повністю відсутні, або ж мають суттєві функціональні обмеження. Це створює значні труднощі при систематичному формуванні аналітичних звітів, моніторингу публікаційної активності, оцінці наукових досягнень та загалом при управлінні науковою діяльністю у закладах вищої освіти. Особливо гостро ця проблема постає для таких установ, як Луцький національний технічний університет, де актуалізація даних про наукові здобутки викладацького складу є важливим компонентом як внутрішньої звітності, так і зовнішнього позиціонування університету.

Таким чином, у першому розділі було всебічно обґрунтовано актуальність обраної теми дослідження, виявлено основні недоліки існуючих підходів до автоматизації збору наукометричної інформації. На основі цього аналізу було сформовано початкові функціональні та нефункціональні вимоги до майбутнього програмного продукту, що включають гнучкість парсингу, обробку помилок, зручність використання та можливість інтеграції. Також було окреслено основні технічні напрями реалізації, зокрема вибір Python для логіки парсингу та Node.js у поєднанні з React для серверної та клієнтської частини веб-додатку, як це детально описано у наступних розділах.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Парсинг профілю користувача або науковця в мережі є актуальним завданням для автоматизації збору інформації, аналізу публікацій, цитувань та інших показників. З огляду на сучасні обсяги даних, ручний збір інформації є надто трудомістким і неефективним. Автоматизований парсинг дозволяє оперативно отримувати структуровані дані з відкритих джерел, таких як Google Scholar.

При проєктуванні архітектури також враховувавались ключові технічні ризики. Основний ризик полягає у «крихкості» парсера, будь-яка зміна HTML-структури на сайтах ЛНТУ чи Google Scholar може призвести до відмови системи. Для пом'якшення цього ризику я спроєктував код модуля scholar_profiles.py таким чином, щоб логіка для кожного джерела була ізольована в окремій функції, що спрощує її подальшу підтримку та оновлення. Також я усвідомлював ризик блокування IP-адреси з боку Google, тому заклав у функціонал механізми імітації людської поведінки, що є компромісом між швидкістю роботи та стабільністю.

Крім технічних аспектів, важливим було дотримання етичних принципів веб-скрапінгу. Реалізовані в модулі випадкові затримки між запитами є не лише технічним прийомом, але й етичною нормою, спрямованою на уникнення створення надмірного навантаження на сервери сайтів-джерел. Я також виходив з того, що зібрані публічні дані будуть використовуватися виключно в академічних та аналітичних цілях в межах університету. Прозорість роботи парсера забезпечується використанням реалістичного заголовка User-Agent, що дозволяє ідентифікувати трафік від мого додатку.

Для розробки мого модуля парсингу я визначив наступні функціональні та нефункціональні вимоги, а також продумав його архітектуру.

Я зосередився на наступних функціональних можливостях:

– автоматичне отримання основної інформації профілю включає в себе вилучення ключових даних науковця, таких як ПІБ, місце роботи, контактна інформація та галузь наукових інтересів;

– збір даних про публікації та наукометричні показники повинен збирати інформацію про публікації, включаючи їх назви, авторів, рік видання, а також важливі метрики, такі як кількість цитувань та індекси Гірша (h-index);

– механізм для виявлення та обробки помилок, що можуть виникнути під час парсингу. Також важливою є здатність модуля адаптуватися до можливих змін у HTML-структурі цільових сайтів, хоча це може вимагати періодичного оновлення коду парсера;

– уникнення блокувань з боку веб-ресурсів, я застосував такі техніки, як встановлення реалістичних HTTP-заголовків (User-Agent) та використання випадкових затримок між запитами, щоб імітувати поведінку людини.

З точки зору нефункціональних аспектів, було поставлено такі цілі:

– модуль надійності та швидкості роботи повинен працювати стабільно та надавати результати за прийнятний час, хоча швидкість може залежати від кількості запитів та необхідних затримок для уникнення блокувань;

– модуль інтерфейсу (API) надає програмний інтерфейс для отримання даних, який має бути простим та зрозумілим для інтеграції з іншими частинами моєї системи;

– модуль кросплатформеності на Python з використанням Flask., де API дозволяє інтеграцію з іншими компонентами;

– використання відкритих інструментів з відкритим кодом, уникаючи залежності від комерційних API, що робить рішення більш доступним та гнучким.

Архітектуру проєкту було реалізовано за модульним підходом, що забезпечує кращу організацію коду та полегшує його підтримку. Хоча у мене немає формальної бази даних для зберігання результатів парсингу, логічно можна виділити наступні модулі:

1) модуль отримання даних відповідає за виконання HTTP-запитів до цільових веб-сторінок та отримання їх HTML-коду. Він також обробляє початкові налаштування запитів, такі як заголовки та керування затримками;

2) модуль парсингу обробляє його, знаходить та витягує потрібну інформацію на основі визначених CSS-селекторів та структури сторінок;

3) модуль обробки та формування відповідає за структурування вилучених даних у формат JSON. У моєму Flask-додатку він формує кінцеву JSON-відповідь, яка надсилається клієнту. Вхідні дані про викладачів ЛНТУ читаються з JSON-файлу, що можна вважати простою файловою системою для початкових налаштувань;

4) модуль взаємодії (API) надає HTTP ендпоінти, через які клієнтська частина може надсилати запити на парсинг та отримувати оброблені наукометричні дані у форматі JSON.

Користувач взаємодіє з розробленою системою через веб-інтерфейс, який створювався за допомогою бібліотеки React. Я намагався зробити цей інтерфейс інтуїтивно зрозумілим та функціональним для ефективного пошуку та перегляду наукометричної інформації.

Початковий екран веб-додатку зустрічає користувача заголовком «Парсер профілів НПП ЛНТУ» та логотипом університету. Ключовим елементом тут є форма пошуку з текстовим полем, призначеним для введення «ПІБ викладача для пошуку», та кнопкою «Знайти».

Дизайн початкового екрану був розроблений з акцентом на простоті та цілеспрямованості. Я свідомо уникнув будь-яких зайвих елементів, таких як меню чи бічні панелі, щоб сфокусувати всю увагу користувача на єдиній ключовій дії – пошуку профілю. Такий мінімалістичний підхід знижує когнітивне навантаження та робить взаємодію з системою максимально очевидною навіть для нового користувача, якому не потрібно витрачати час на вивчення інтерфейсу.

Технічно, цей інтерфейс реалізовано як односторінковий додаток (Single-Page Application) на React. Це означає, що після початкового завантаження всі

подальші дії, такі як відображення результатів, індикаторів завантаження чи повідомлень про помилки, відбуваються динамічно, без необхідності перезавантажувати сторінку в браузері. Це забезпечує швидкий та плавний досвід користування, що є стандартом для сучасних веб-додатків та робить роботу з модулем більш комфортною. На рисунку 2.1 зображено початковий екран веб-додатку.

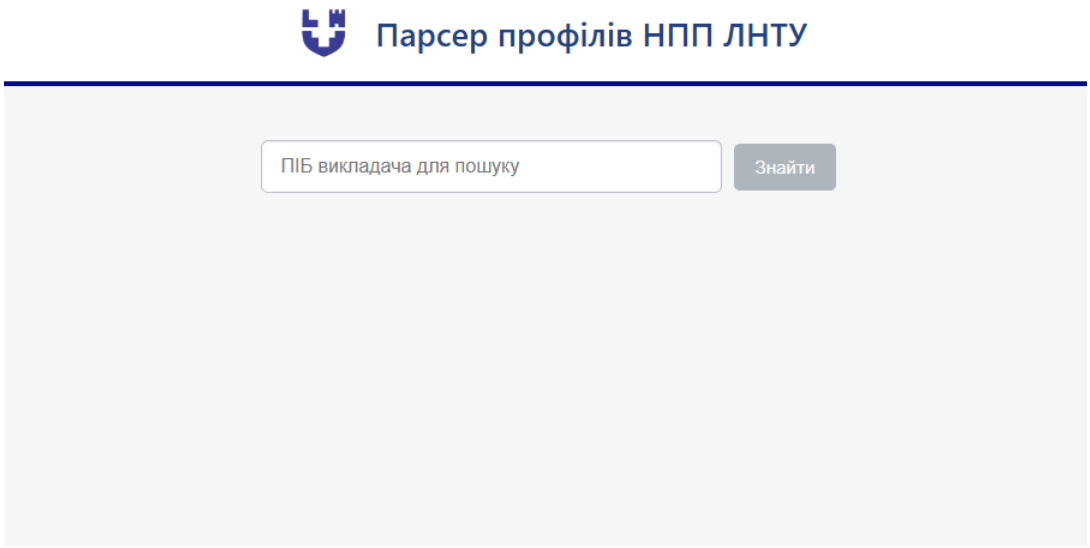


Рисунок 2.1 – Початковий екран веб-додатку

Користувач вводить необхідне ім’я або його частину і натискає кнопку «Знайти». Ця дія ініціює асинхронний запит до мого Python/Flask бекенду через спеціально створений ендпоінт /api/profile, передаючи введене ім’я як параметр. Під час обробки цього запиту на сервері, інтерфейс інформує користувача про процес завантаження, наприклад, змінюючи текст на кнопці «Знайти» на «Пошук... » та тимчасово блокуючи її для уникнення повторних запитів.

Після отримання запиту, Flask-бекенд починає багатоетапний процес пошуку інформації. Спочатку він намагається знайти дані про вказаного викладача на офіційному сайті Рейтингу НПП ЛНТУ, використовуючи або заздалегідь відоме посилання з локального JSON-файлу lntu_faculty_urls.json, або здійснюючи парсинг сторінки ЛНТУ, якщо URL профілю передано напряму.

Одночасно або послідовно, залежно від реалізованої логіки, бекенд також виконує пошук профілю цього науковця в системі Google Scholar.

Коли всі необхідні дані зібрані (або визначено, що їх немає), фронтенд відображає агреговану інформацію профілю науковця у спеціально структурованому блоці. У верхній частині цього блоку я розмістив ключову ідентифікаційну інформацію: повне ім’я викладача з даних ЛНТУ, його фото (пріоритет надається фото з сайту ЛНТУ, за його відсутності – з Google Scholar), а також посаду, кафедру та факультет. Якщо ім’я в Google Scholar відрізняється від імені на сайті ЛНТУ, я також показую це альтернативне ім’я для повноти картини. Для зручності перегляду детальної інформації реалізовано можливість перемикання між двома основними розділами за допомогою кнопок-вкладок: «Профіль НПП ЛНТУ» та «Дані Google Scholar». Активна вкладка підсвічується, і користувач може легко переходити між ними. На рисунку 2.2 зображено профіль викладача у моєму веб-додатку.

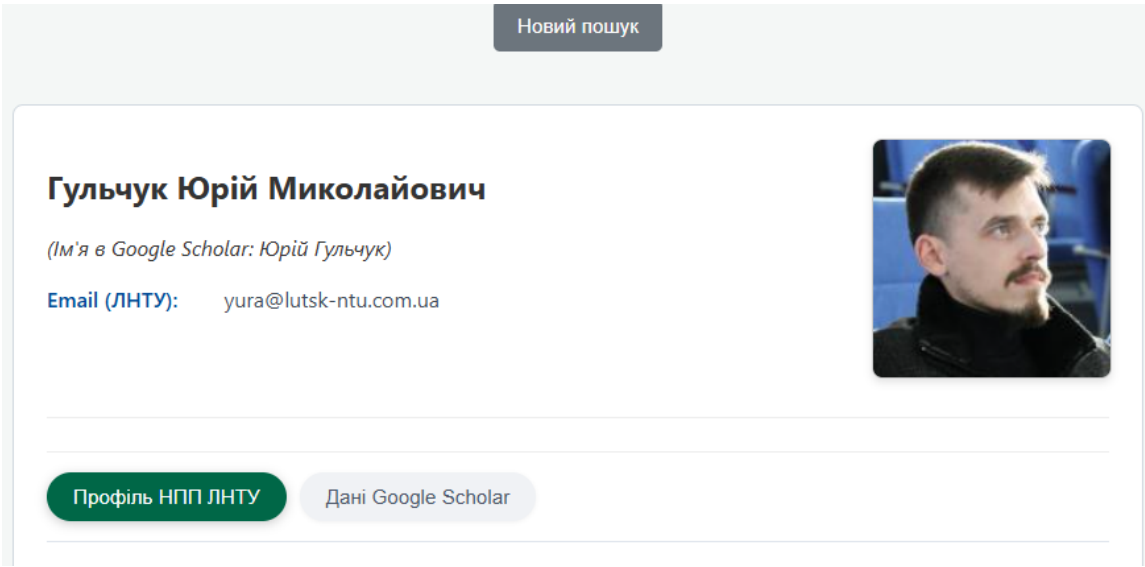


Рисунок 2.2 – Профіль викладача у веб-додатку

При виборі вкладки «Профіль НПП ЛНТУ», користувачеві надається вся інформація, зібрана з сайту НПП ЛНТУ [5]. Це включає пряме посилання для переходу на повний профіль на сайті Рейтингу ЛНТУ. Далі йдуть наукометричні показники, які університет агрегує самостійно: дані про ORCID [6], зведені дані

з Scopus [7], Google Scholar та Web of Science [8]. Окремими блоками я виводжу розгорнуту біографію науковця, перелік етапів його освіти, досвід роботи, інформацію про участь у проєктній діяльності та сформульовані наукові інтереси. Також тут відображається список публікацій, зазначених безпосередньо на сайті ЛНТУ, згрупованих за типом та роком, з можливістю переходу за посиланнями, якщо вони є. На рисунку 2.3 зображено вкладку «Профіль НПП ЛНТУ».

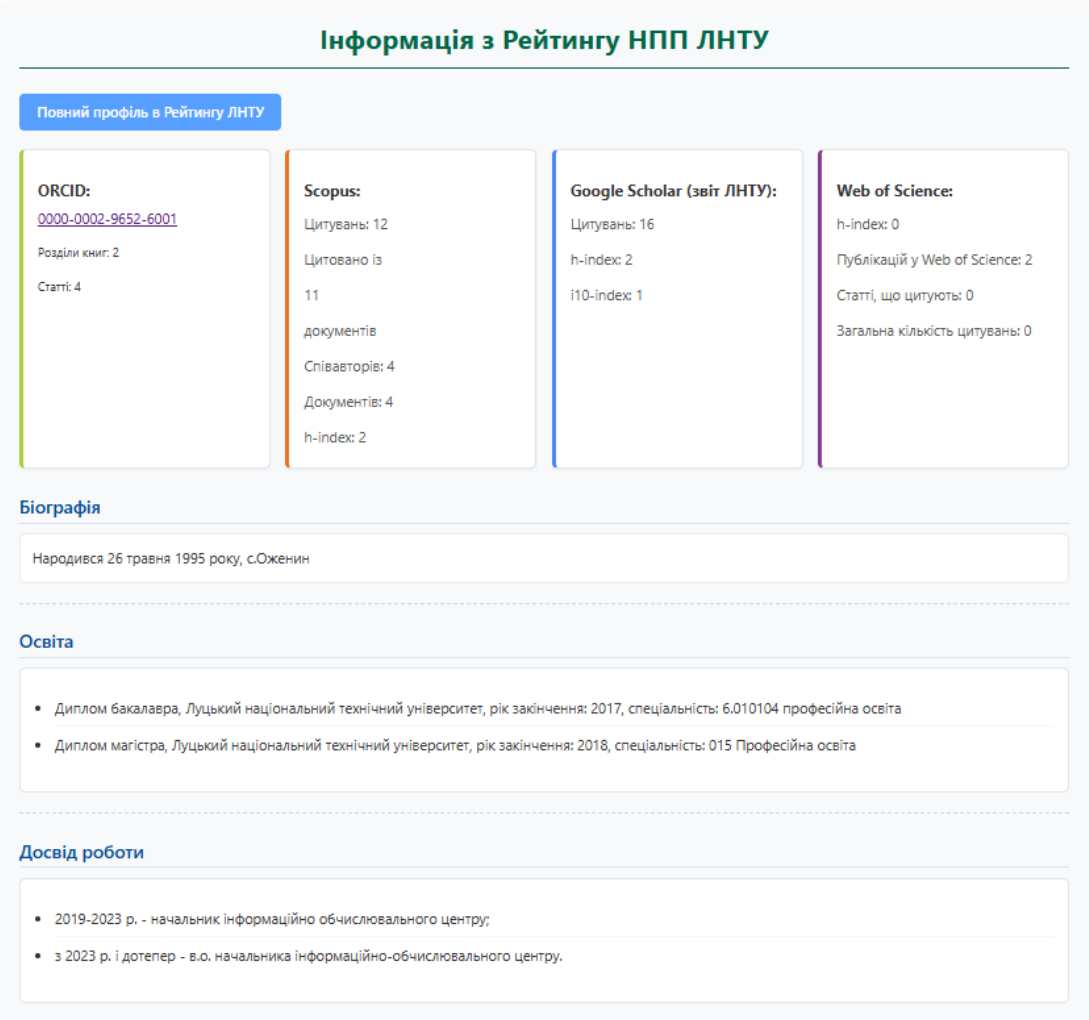


Рисунок 2.3 – Вкладка «Профіль НПП ЛНТУ»

Переключившись на вкладку «Дані Google Scholar», користувач бачить інформацію, отриману безпосередньо з профілю науковця в Google Scholar. Тут також є пряме посилання для переходу на оригінальну сторінку профілю Google Scholar. Відображаються ПІБ, афіліація та контактний email, якщо вони були

підтверджені та опубліковані науковцем у своєму GS профілі. Якщо ж профіль GS не було знайдено, але вдалося отримати загальний список статей за ПІБ, цей список буде відображено тут з відповідним інформаційним повідомленням. Мій інтерфейс також обробляє та показує користувачеві різні інформаційні повідомлення або повідомлення про помилки, наприклад, «Профіль ... не знайдено», «Знайдено кілька профілів... » або помилки, що могли виникнути під час роботи бекенду. На рисунку 2.4 зображено вкладка «Дані Google Scholar».

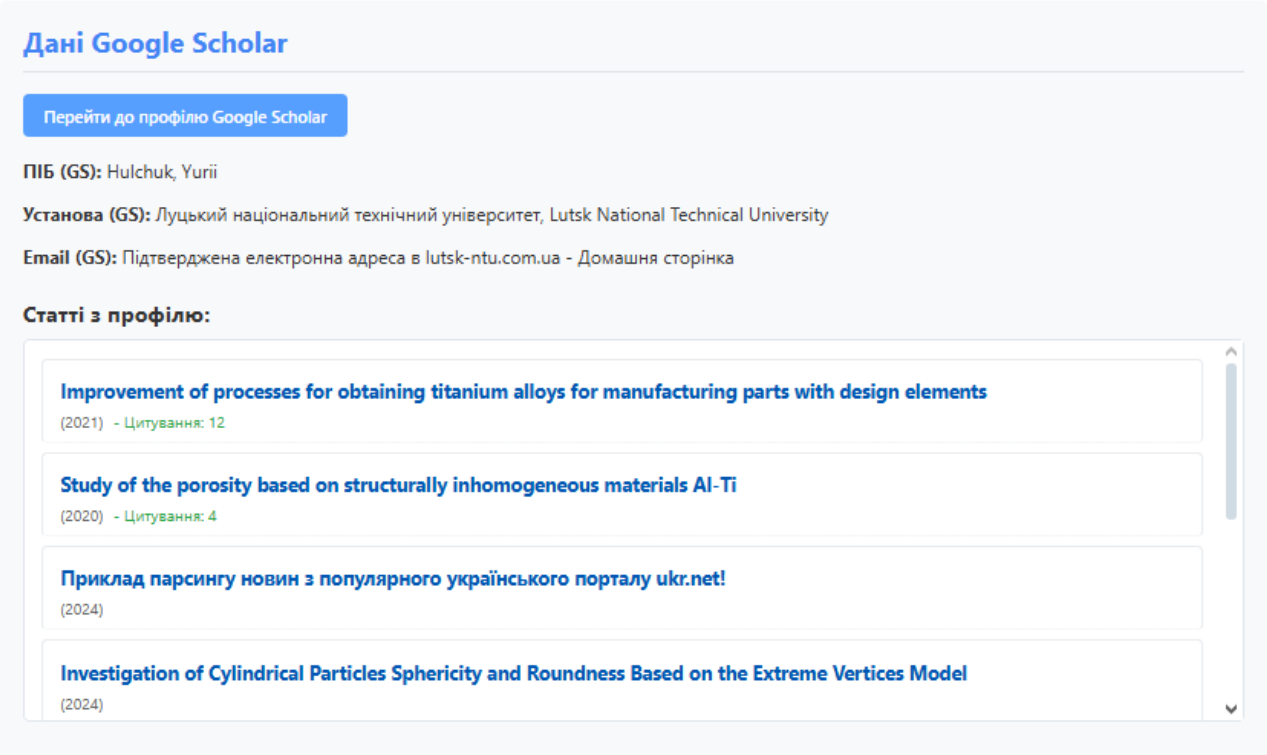


Рисунок 2.4 – Вкладка «Дані Google Scholar»

Для забезпечення прозорості та можливості верифікації даних, я передбачив у моєму інтерфейсі наявність прямих клікабельних посилань. Користувач може легко перейти на оригінальну сторінку профілю НПП на сайті Рейтингу ЛНТУ або на сторінку його профілю в Google Scholar. Ці посилання зазвичай відкриваються у новій вкладці браузера, що дозволяє користувачеві одночасно переглядати дані в моєму додатку та на першоджерелах.

Після завершення роботи з поточним профілем, користувач має можливість ініціювати новий пошук. Для цього я розмістив кнопку «Новий

пошук» у верхній частині сторінки, коли відображаються дані профілю. Натискання цієї кнопки очищує всю раніше завантажену інформацію про науковця, скидає можливі повідомлення про помилки та повертає користувача до початкового стану інтерфейсу з активним полем для введення ПІБ нового викладача. Це дозволяє послідовно працювати з профілями різних науковців без необхідності перезавантажувати сторінку.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Розробка мого модуля парсингу наукометричних даних передбачала створення системи, що складається з бекенд-сервісу для збору та обробки даних і фронтенд-частини для взаємодії з користувачем.

Бекенд мого модуля я реалізував на мові програмування Python з використанням мікрофреймворку Flask. Python я обрав через його простоту, потужні бібліотеки для веб-скрапінгу та обробки даних. Flask дозволив мені легко створити REST API ендпоінти, які приймають запити від клієнтської частини та повертають дані у форматі JSON.

Для виконання HTTP-запитів та отримання HTML-вмісту сторінок з сайту ЛНТУ та деяких сторінок Google Scholar я використав бібліотеку Requests [9].

Для розбору отриманого HTML-коду та вилучення необхідних даних я застосував бібліотеку BeautifulSoup4 [10].

Оскільки сторінки профілів Google Scholar часто використовують JavaScript, для парсингу детальної інформації з конкретного профілю Google Scholar я використав бібліотеку Selenium разом з веб-драйвером для Chrome в headless-режимі.

Початковий список викладачів ЛНТУ та посилання на їхні профілі на сайті університету я завантажую з локального JSON-файлу (lntu_faculty_urls.json). Це єдине місце, де дані зберігаються персистентно на стороні сервера у вигляді файлу перед обробкою. Зібрані в результаті парсингу дані не зберігаються мною

у формальній базі даних на сервері; натомість вони обробляються «на льоту» і передаються у форматі JSON на клієнтську частину для відображення.

Для запобігання блокуванню я використав випадкові затримки (time.sleep(random.uniform(...))) між запитами та встановлював реалістичні HTTP-заголовки. Також я реалізував базове логування (logging).

У таблиці 2.1 зображено порівняння підходів до парсингу які використовуються у розробці.

Таблиця 2.1 – Порівняння обраних мною підходів до парсингу які використовуються у розробці

Критерій	Requests + BeautifulSoup4	Selenium (з Chrome WebDriver)
Складність освоєння	Легкий	Середній
Швидкість розробки	Висока для статичних сайтів	Середня, вимагає налаштування веб-драйвера
Робота з JavaScript	Не підтримує (обробляє лише отриманий HTML)	Повна підтримка, виконує JavaScript як браузер
Продуктивність	Висока, мінімальні накладні витрати	Нижча через запуск та керування екземпляром браузера
Ресурсоємність	Низька	Вища (пам’ять, CPU)
Використання в проєкті	Парсинг сайту ЛНТУ, пошук авторів та статей в Google Scholar	Парсинг детальних сторінок профілів Google Scholar
Переваги	Простота, швидкість для відповідних задач	Здатність обробляти динамічний контент, обходити деякі захисти
Недоліки	Не підходить для сайтів, що сильно залежать від JavaScript	Повільніший, більш ресурсоємний, крихкість до змін у структурі сайту

При розробці модуля парсингу було враховано популярність, стабільність і сумісність з іншими компонентами системи. Вибір кожного з них їхньою функціональність, популярностю в розробці подібних програм та сумісностю їх роботи між собою. В стоворені програми було використано достатньо інструментарію для розкриття багатьох функцій, які будуть ефективно справлятися з поставленими їм задачами.

Python [11] – це високорівнева, інтерпретована мова програмування загального призначення, що вирізняється лаконічним і зрозумілим синтаксисом, що робить її особливо зручною для новачків, а також ефективною для досвідчених розробників. Вона підтримує декілька парадигм програмування,

включаючи об’єктно-орієнтовану, процедурну та функціональну, що дозволяє адаптувати стиль розробки до конкретних завдань.

Одна з ключових переваг Python – це величезна стандартна бібліотека та велика кількість зовнішніх модулів, які значно розширюють її функціональність. Існують бібліотеки для всього – від обробки зображень і роботи з мережею до штучного інтелекту, машинного навчання, мовного розпізнавання та аналізу даних. Python отримав широке визнання в академічних колах, наукових дослідженнях і промислових рішеннях, завдяки своїй відкритості, активній спільноті та швидкому розвитку. Її перевагами є:

– чистий синтаксис;

– переносність програм;

– стандартний дистрибутив має велику кількість корисних модулів;

– можливість використання Python в діалоговому режимі;

– стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке називається IDLE і яке написане мовою Python;

– зручний для розв’язання математичних проблем.

Python, завдяки своєму багатому набору бібліотек, став основою для реалізації логіки парсингу. Зокрема, бібліотека requests використовувалася для здійснення HTTP-запитів до веб-сторінок профілів НПП ЛНТУ та Google Scholar. Для розбору отриманого HTML-коду та вилучення необхідних даних застосовувалася бібліотека BeautifulSoup4. У випадках, коли сторінки Google Scholar вимагали виконання JavaScript для відображення повного контенту або мали складні механізми захисту, для взаємодії з ними та отримання фінального HTML-коду використовувалася бібліотека Selenium [12] з веб-драйвером для браузера Chrome. Такий набір інструментів дозволив гнучко підходити до парсингу сторінок різної складності.

Node.js [13] – це кросплатформне середовище виконання JavaScript, побудоване на рушії V8 від Google Chrome, яке дозволяє виконувати JavaScript-код поза межами веб-браузера, зокрема на серверній стороні. Це відкриває

можливість для розробників використовувати єдину мову програмування як для клієнтської, так і для серверної розробки, що сприяє уніфікації процесу та спрощує формування команд розробників. Node.js надає багатий набір вбудованих модулів для роботи з файловою системою, мережевими протоколами, потоками даних та іншими системними операціями. У контексті даної кваліфікаційної роботи Node.js використовується для створення проксі-сервера API. Цей сервер виступає посередником між клієнтською частиною, реалізованою на React.js, та основним Python/Flask API, що відповідає за парсинг.

Ключовою особливістю Node.js є його подія-орієнтована архітектура та неблокуюча модель вводу/виводу. Асинхронна модель дозволяє Node.js ефективно обробляти велику кількість одночасних з'єднань та запитів з мінімальними накладними витратами на створення потоків, на відміну від традиційних багатопотокових серверних технологій. Це робить його особливо придатним для розробки швидких та масштабованих мережевих застосунків, таких як API-сервери, веб-сервіси реального часу та мікросервіси. Для управління залежностями та пакетами Node.js використовує менеджер пакетів npm, який надає доступ до величезної екосистеми відкритих бібліотек та інструментів, що значно прискорює та спрощує розробку.

Visual Studio Code [14] – це редактор початкового коду, створений Microsoft із Electron Framework для Windows, Linux і macOS. Функції включають підтримку налагодження, підсвічування синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований Git. Користувачі можуть змінювати тему, комбінації клавіш, параметри та встановлювати розширення, які додають функціональність. Він активно використовується розробниками програмного забезпечення завдяки своїй гнучкості, продуктивності та широкому набору функціональних можливостей. Редактор підтримує велику кількість мов програмування, таких як JavaScript, Python, C++, C#, Java, PHP, HTML, CSS та багато інших, що забезпечується завдяки системі розширень. Через вбудований магазин розширень можна легко додавати нові

функції, теми, інструменти форматування коду, інтеграції з фреймворками, базами даних і навіть середовищами виконання.

У процесі реалізації проєкту з парсингу профілю НПП ЛНТУ активно використовувалося середовище Visual Studio Code, яке стало основним інструментом для написання, налагодження та структурування коду як на Python, так і на JavaScript. Завдяки вбудованому терміналу, автоматичному форматуванню коду та підтримці системи контролю версій Git, стало можливим ефективно організовувати робочий процес і швидко виявляти помилки під час розробки.

React [15] – це сучасна JavaScript-бібліотека для створення динамічних інтерфейсів користувача, розроблена компанією Meta. Вона базується на компонентному підході та віртуальному DOM, що дозволяє ефективно оновлювати сторінки без перезавантаження, забезпечуючи високу швидкість роботи та зручність взаємодії для користувача. React був використаний для створення фронтенду вебзастосунку, який дозволяє здійснювати пошук профілів викладачів ЛНТУ у Google Scholar. Завдяки React було реалізовано інтуїтивно зрозумілий інтерфейс із формою для введення імені викладача, кнопкою запуску парсингу, а також компонентами для відображення результатів.

React дозволив гнучко організувати взаємодію з бекендом через API-запити, забезпечивши динамічне оновлення даних без потреби в оновленні сторінки. Таким чином, React виконує ключову роль у формуванні зручного та сучасного інтерфейсу для взаємодії з парсером Google Scholar у межах студентського проєкту.

Крім того, React чудово інтегрується з іншими технологіями, які були використані в проєкті: Node.js, Flask та бібліотекою scholarly для парсингу. Кожна зміна стану програми відображається у відповідному компоненті без необхідності ручного оновлення сторінки. Таким чином, React виступає центральною технологією для реалізації користувацького інтерфейсу, забезпечуючи інтерактивну та зручну роботу з системою парсингу наукових профілів. Це повністю відповідає сучасним підходам до створення

вебзастосунків та підвищує якість взаємодії користувача з програмним продуктом.

Google Scholar – це безкоштовна пошукова система, створена компанією Google для індексації наукової літератури: статей, дисертацій, книг, технічних звітів, конференційних матеріалів тощо. Вона охоплює широке коло академічних джерел з різних галузей знань і є однією з найпопулярніших платформ для пошуку наукової інформації в усьому світі.

Індекс Google Scholar включає в себе більшість рецензованих онлайн-журналів Європи та Америки найбільших наукових видавництв.

Таким чином, Google Scholar став критичним джерелом даних для збору та візуалізації наукових досягнень викладачів ЛНТУ, а програмна взаємодія з ним була реалізована засобами Python з використанням бібліотек requests, BeautifulSoup4 та Selenium для парсингу його веб-сторінок.

Цей підхід дозволив створити вебзастосунок, який автоматично показує наукову активність обраного викладача, що може бути корисним для студентів, адміністрації, абітурієнтів, науковців тощо. Проєкт також демонструє практичне застосування сучасних технологій у навчальному процесі та формуванні цифрових освітніх інструментів.

Flask [16] – це легкий вебфреймворк для мови програмування Python, призначений для створення вебдодатків та API. Він належить до категорії мікрофреймворків, оскільки надає мінімальний набір інструментів «із коробки», дозволяючи розробникам самостійно обирати додаткові компоненти залежно від потреб проєкту. Незважаючи на свою простоту, Flask є потужним інструментом, який підтримує створення масштабованих вебрішень. У межах дипломної роботи Flask був використаний для реалізації бекенд-частини вебзастосунку, що здійснює парсинг профілю науково-педагогічного працівника ЛНТУ з Google Scholar.Flask у цьому проєкті слугує посередником між клієнтською частиною та парсинг-модулем. Такий розподіл дозволяє легко масштабувати архітектуру, реалізовувати додаткові API, обробляти авторизацію, логування, зберігання історії запитів тощо.

Окрім того, Flask дозволяє тестувати серверну частину окремо, незалежно від інтерфейсу користувача. Це покращує модульність, спрощує налагодження та розширення проєкту.

Завдяки використанню Flask у поєднанні з розробленими на Python функціями парсингу вдалося реалізувати надійний та гнучкий API, який забезпечує взаємодію з сайтом ЛНТУ та Google Scholar у режимі реального часу.

2.3 Проєктування архітектури системи та взаємодія модулів

Розроблений програмний модуль для парсингу профілів науково-педагогічних працівників ЛНТУ характеризується багатокомпонентною архітектурою, яка реалізує клієнт-серверний принцип взаємодії з використанням проміжного проксі-сервера. Такий підхід був обраний для забезпечення чіткого розмежування функціональних обов’язків між окремими частинами системи, що позитивно впливає на її гнучкість, можливість модифікації та потенціал для подальшого масштабування.

Загальний потік даних та взаємодії у системі організований наступним чином:

1) користувач ініціює запит через React-інтерфейс;
2) цей запит надходить на проксі-сервер Node.js, який перенаправляє його на Python/Flask API;
3) Flask API координує роботу модуля scholar_profiles.py для виконання парсингу сторінок ЛНТУ та Google Scholar;
4) зібрані та агреговані дані у форматі JSON повертаються через Flask API на проксі-сервер, а звідти – на React-клієнт, де відбувається їх фінальне відображення.

Вся комунікація між компонентами здійснюється за протоколом HTTP, а для обміну даними використовується універсальний формат JSON. Системне логування на стороні Python забезпечує можливість моніторингу стану системи та діагностики проблем.

Вибір такої багатокомпонентної архітектури був зумовлений прагненням до чіткого розділення відповідальностей:

– Python з його потужними бібліотеками оптимально підходить для завдань веб-скрапінгу та обробки даних;

– Flask забезпечує швидке створення легковагого API;

– Node.js добре зарекомендував себе для створення асинхронних проксі-серверів та обробки HTTP-запитів;

– React.js надає ефективні інструменти для побудови динамічних та інтерактивних користувацьких інтерфейсів.

Така модульність спрощує незалежну розробку, тестування та модифікацію кожної частини системи. Використання Selenium для взаємодії з Google Scholar, хоча й додає певної складності, є виправданим для обробки динамічного контенту та спроб обходу базових механізмів захисту від автоматизованого збору даних. Архітектура також закладає основи для потенційного майбутнього масштабування, наприклад, шляхом винесення Python-парсерів у окремі мікросервіси або додавання більш складних механізмів кешування на рівні Node.js проксі. Таким чином, обрана архітектура не тільки ефективно вирішує поточні завдання кваліфікаційної роботи, але й демонструє гнучкість, необхідну для довготривалого розвитку та підтримки програмного продукту в реальних умовах експлуатації.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи я здійснив обґрунтування та вибір технологічного стеку для розробки модуля парсингу наукометричних даних НПП ЛНТУ з сайту університету та Google Scholar. Розроблена мною система складається з Python/Flask бекенду та React фронтенду.

Для реалізації бекенд-логіки було обрано Python. Flask використано для створення API. Для взаємодії з веб-сторінками застосувався комбінований підхід:

– Requests та BeautifulSoup4 для отримання та розбору статичного HTML-контенту;

– Selenium з Chrome WebDriver для парсингу динамічних сторінок профілів Google Scholar.

Фронтенд, розроблений на React, забезпечує взаємодію з користувачем. Середовищем розробки слугував VS Code.

Обрані технічні засоби дозволили ефективно вирішити поставлені завдання. Хоча формальні UML-діаграми не створювалися, розуміння архітектури системи, її компонентів та потоків даних було ключовим для успішної реалізації. Майбутній розвиток мого проєкту може передбачати інтеграцію модуля з інформаційними системами ЛНТУ та впровадження більш стійких механізмів обходу захисту від парсингу.

РОЗДІЛ 3
РОЗРОБКА МОДУЛЯ ПАРСИНГУ ДЛЯ ПРОФІЛЮ НПП ЛНТУ З ВИКОРИСТАННЯМ PYTHON, NODE.JS ТА GOOGLE SCHOLAR API

3.1 Практична реалізація об’єкта проєктування

У рамках кваліфікаційної роботи було реалізовано програмний модуль, призначений для автоматизованого збору, обробки та представлення наукометричних даних науково-педагогічних працівників Луцького національного технічного університету. Основна мета модуля – парсинг інформації з офіційного сайту Рейтингу НПП ЛНТУ та з профілів науковців у Google Scholar. Для реалізації серверної частини було обрано мову програмування Python та мікрофреймворк Flask. Для веб-скрапінгу використовувалися бібліотеки Requests, BeautifulSoup та Selenium. Клієнтська частина розроблена на React. Початкові дані про викладачів ЛНТУ зберігаються у JSON-файлі, а результати парсингу передаються на фронтенд у форматі JSON без збереження у традиційній базі даних на сервері.

Ключовими компонентами мого Python/Flask бекенду є:

– основний файл app.py Flask-додатку який містить визначення маршрутів API (ендпоінтів), логіку обробки запитів від клієнта, координацію викликів функцій парсингу з модуля scholar_profiles.py та формування JSON-відповідей;

– модуль scholar_profiles.py., в якому була реалізована основну логіку веб-скрапінгу;

– файл lntu_faculty_urls.json, що містить початковий список НПП ЛНТУ та посилання на їхні персональні сторінки на сайті rating2.lntu.edu.ua. Цей файл слугує вхідною точкою для парсингу даних з сайту університету.

Файл app.py визначає основні ендпоінти API, через які мій React-фронтенд взаємодіє з бекендом:

– /api/faculty-list надає початковий список викладачів;

– /api/profile приймає ім’я науковця або URL профілів як параметри запиту, ініціює процес парсингу та повертає агреговані дані.

Логіка роботи ендпоінту /api/profile в app.py є центральною. Він отримує параметри запиту, визначає, яку інформацію потрібно зібрати, викликає відповідні функції з scholar_profiles.py для скрапінгу, об'єднує отримані результати та повертає їх у вигляді єдиного JSON-об'єкта. Я також реалізував логування для відстеження процесу та діагностики помилок.

Основне відбувається у функціях модуля scholar_profiles.py. Наприклад функція get_lntu_profile_data використовує бібліотеку Requests для завантаження HTML-сторінки профілю з сайту ЛНТУ та BeautifulSoup для її розбору. Я визначив CSS-селектори для пошуку та вилучення конкретних даних: ПІБ, фото, факультет, кафедра, посада, біографія, наукові інтереси, публікації, посилання на Google Scholar тощо.

Для парсингу профілів Google Scholar я застосував два підходи. Функція get_author_profiles_from_search також використовує Requests та BeautifulSoup для пошуку списку можливих профілів авторів. Однак, для детального парсингу конкретної сторінки профілю Google Scholar, яка активно використовує JavaScript для відображення контенту, було використано Selenium з Chrome WebDriver у headless-режимі. Це дозволяє коду бачити сторінку так, як її бачить браузер після виконання всіх скриптів, і таким чином отримувати доступ до динамічно завантажених даних, таких як h-індекс, i10-індекс, список публікацій та їх цитування. Для зменшення ймовірності блокувань було додано випадкові затримки між запитами та налаштував User-Agent.

У лістингу 3.1 наведено фрагмент коду з функції get_author_profile_data_selenium, що ілюструє використання Selenium для отримання інформації зі сторінки профілю Google Scholar.

Лістинг 3.1 – Фрагмент коду парсингу Google Scholar профілю за допомогою Selenium

```
<def get_author_profile_data_selenium(profile_url,
headers_for_selenium_user_agent):
     options = webdriver.ChromeOptions()
   options.add_argument('--headless=new')
```

```
     options.add_argument(f"user-
agent={headers_for_selenium_user_agent.get('User-Agent','Mozilla/5.0')}")
    driver = None
    try:
        driver = webdriver.Chrome(options=options)
        driver.get(target_profile_url_with_pagesize)
        WebDriverWait(driver,
20).until(EC.presence_of_element_located((By.ID, "gsc_prf_in")))
         soup = BeautifulSoup(driver.page_source, 'html.parser')
         name_el = soup.select_one('#gsc_prf_in')
        if name_el: profile_info["name"] = name_el.text.strip()
        stats_tds = soup.select('#gsc_rsb_st table td.gsc_rsb_std')
        if len(stats_tds) >= 3:
           profile_info["citations_total"] = stats_tds[0].text.strip()
           profile_info["h_index"] = stats_tds[1].text.strip()
            profile_info["i10_index"] = stats_tds[2].text.strip()
    finally:
        if driver: driver.quit()
     return profile_info
div class="mt-2 w-full xl:max-w-[320px] flex items-center gap-2">
<label class="input input-bordered flex items-center gap-2 w-full">
     <i class="ri-search-line"></i>
      <input wire:model.live.debounce.200ms="form.searchText" class="w-
full" type="text" class="grow" placeholder="Пошук клієнта" />
</label>
      <button wire:click="resetFilter" class="btn btn-sm btn-primary px-1
min-w-[40px] min-h-[40px]" aria-label="Reset filter">
           <i class="ri-reset-left-line ri-lg"></i>
     </button>
```

кінець лістингу 3.1

Фронтенд-частина проєкту, реалізована на React, вона забезпечує користувацький інтерфейс для взаємодії з модулем парсингу. Головна сторінка містить поле для введення ПІБ науковця та кнопку «Знайти ». Після відправки запиту, фронтенд асинхронно звертається до API мого Python/Flask бекенду.

Після отримання відповіді від бекенду, фронтенд відображає результати. Якщо було знайдено кілька профілів Google Scholar, користувачеві пропонується список для вибору. Якщо дані успішно завантажені, вони представляються у структурованому вигляді: окремі секції для інформації з сайту ЛНТУ та з Google Scholar. Користувач може перемикатися між цими секціями. Було реалізовано відображення фото, ПІБ, посади, кафедри, факультету, наукометричних показників, біографії, списків публікацій з обох джерел та іншої інформації.

У лістингу 3.2 наведено фрагмент JSX-коду з компонента App.js, який відповідає за відображення списку публікацій з Google Scholar.

Лістинг 3.2 – Фрагмент JSX-коду для відображення списку публікацій

```
<direturn (
     <ul className="articles-list scrollable-content">
       {articles.map((pub, index) => (
         <li key={pub.link || `${sourceLabel}-pub-${index}`}
className="article-item">
           <strong>{pub.title || "Без назви"}</strong>
           {pub.year && <span className="article-year">
({pub.year})</span>}
          {pub.citations && pub.citations !== "0" && (<span
className="article-citations"> - Цитування: {pub.citations}</span>)}
           {pub.link && sourceLabel === "Google Scholar" && (<a
href={pub.link} target="_blank" rel="noopener noreferrer"
className="article-link"> [Детальніше в GS]</a>)}
           {pub.snippet && <p className="snippet">{pub.snippet}</p>}
         </li>
       ))}
     </ul>
 );
```

кінець лістингу 3.2

Стилізацію фронтенд-частини виконано за допомогою CSS, описаного у файлі App.css. Я намагався зробити інтерфейс простим, функціональним та візуально зрозумілим, зосереджуючись на зручності представлення даних. Адаптивність для різних розмірів екранів також бралася до уваги на базовому рівні.

Розгортання мого проєкту на даному етапі передбачає локальний запуск. Python/Flask бекенд запускається на порту 5001, а React-фронтенд, ймовірно, на стандартному для нього порту (наприклад, 3000) і взаємодіє з бекендом через Node.js проксі-сервер, налаштований на порт 3001, який вже перенаправляє запити до Flask API.

Тестування та відлагодження я проводив переважно вручну, перевіряючи різні сценарії пошуку, коректність відображення даних для різних профілів НПП. На стороні Python активно використовувався вбудований модуль logging для

запису інформації про хід виконання запитів, URL-адреси, що запитуються, та можливі помилки. Також, як видно із закоментованих рядків у коді, під час налагодження зберігався HTML-вміст сторінок у файли для детального аналізу їх структури. На стороні React використовувався console.log для відстеження стану компонентів та даних, а також інструменти розробника в браузері.

У лістингу 3.3 наведено приклад використання логування в моєму Python-коді.

Лістинг 3.3 – Приклад використання логування в app.py

```
@app.route('/api/profile', methods=['GET'])
 def api_profile():
    try:
         name_from_request = request.args.get('name')
      logging.info(f"==== API /api/profile викликано.
name='{name_from_request}', ... ====")        # ...
        logging.info(json.dumps(final_response_data, ensure_ascii=False,
indent=2))
        return jsonify(final_response_data)
  except Exception as e:
        logging.error(f"Критична помилка в /api/profile:
{e}\n{traceback.format_exc()}")
        return jsonify({"error": "Внутрішня помилка сервера Python API.",
"details": str(e)}), 500
```

кінець лістингу 3.3

Такий підхід до реалізації дозволив створити функціональний прототип модуля для парсингу та відображення наукометричних даних НПП ЛНТУ, який вирішує поставлені в роботі завдання.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

У процесі розробки модуля парсингу значна увага приділялась тестуванню та налагодженню, щоб забезпечити коректність його роботи та виявити можливі помилки. Оскільки проєкт складається з Python/Flask бекенду та React фронтенду, я використовував підходи, релевант для цих технологій.

На початкових етапах розробки Python-скриптів для парсингу, часто використовувалась стандартна функція print() для виведення значень змінних, HTML-фрагментів або структурованих даних на різних етапах виконання коду. Це дозволяло швидко перевіряти, чи коректно вилучаються дані, чи правильно формуються URL-адреси тощо.

Для більш систематичного відстеження подій та помилок активно використовувся вбудований у Python модуль logging. Я налаштував базову конфігурацію логування в app.py для запису інформації про вхідні запити до API, параметри цих запитів, URL-адреси, до яких зверталися парсери, проміжні результати та можливі винятки. Це було особливо корисно для аналізу проблем, що виникали при взаємодії з зовнішніми сайтами, наприклад, при отриманні неочікуваних HTTP-статусів або при виявленні CAPTCHA.

Окрім виведення інформації в консоль, дуже корисним методом налагодження при розробці веб-скраперів є збереження HTML-коду сторінок, що аналізуються, у локальні файли. Це дозволяє детально вивчити структуру документа офлайн, перевірити правильність CSS-селекторів або XPath-виразів, а також зрозуміти причини помилок парсингу, якщо такі виникають. Такий підхід також періодично застосовувався, особливо при роботі зі складними сторінками або коли потрібно було точно ідентифікувати потрібні елементи. У лістингу 3.4 наведено приклад коду, який я використовував для збереження HTML-відповіді сервера під час відладки.

Лістинг 3.4 – Фрагмент коду для збереження HTML-сторінки під час налагодження

```
         response = requests.get(search_url, headers=headers, timeout=15)
        safe_author_name = "".join(c if c.isalnum() else "_" for c in
author_name)
         with
open(f"debug_gs_author_search_page_for_{safe_author_name}.html", "w",
encoding="utf-8") as f:
             f.write(response.text)
         logging.info(f"HTML для ’{author_name}’ (GS Author Search)
збережено. Статус: {response.status_code}")
```

кінець лістингу 3.4

Збереження HTML-файлів таким чином давало змогу відкрити їх у браузері, використовувати інструменти розробника для інспектування елементів та тестування селекторів без необхідності повторно надсилати запити до цільового сервера, що також є важливим для уникнення надмірного навантаження на ресурс та можливих тимчасових блокувань. Це особливо актуально при роботі з такими сайтами, як Google Scholar, структура яких може бути складною, а також які мають механізми захисту.

На стороні React-фронтенду я використовував console.log() для виведення в консоль браузера даних про стан компонентів, отримані відповіді від API та проміжні значення змінних. Інструменти розробника в браузері, зокрема вкладки «Network» та «Console», були моїми основними інструментами для налагодження фронтенд-частини.

Хоча я не писав формальних автоматизованих тестів (наприклад, з використанням unittest або pytest для Python, чи Jest/React Testing Library для React) у рамках даної кваліфікаційної роботи, середовище розробки та структура коду дозволяють їх інтеграцію в майбутньому. На даному етапі я зосередився на ручному тестуванні.

Ручне тестування включало перевірку всіх основних сценаріїв взаємодії з системою:

– пошук за різними ПІБ;

– коректність обробки ситуацій, коли профіль ЛНТУ знайдено/не знайдено;

– коректність обробки ситуацій, коли профіль Google Scholar знайдено;

– правильність вибору одного профілю зі списку кількох знайдених профілів Google Scholar;

– адекватність відображення всіх типів даних в інтерфейсі React;

– роботу посилань на оригінальні джерела;

– функціональність кнопки «Новий пошук»;

– відображення повідомлень про помилки або інформаційних повідомлень.

Такий комплексний підхід до налагодження та ручного тестування дозволив виявити та усунути більшість проблем та забезпечити стабільну роботу розробленого модуля парсингу.

Після завершення ітеративного налагодження та ручного тестування, я провів фінальне експериментальне дослідження для кількісної оцінки точності та ефективності розробленого модуля.

Для формальної перевірки якості роботи парсера була обрана вибірку з 10 профілів НПП ЛНТУ, для яких існували профілі на сайті університету та в Google Scholar. Для кожного профілю я вручну зібрав дані безпосередньо з сайтів на момент тестування: загальна кількість цитувань, h-індекс, i10-індекс та кількість публікацій. Після цього я запускав свій програмний модуль для парсингу цих же профілів та порівнював отримані автоматично дані з еталонними. Результати порівняння наведено в таблиці 3.1.

Таблиця 3.1 – Результати тестування точності збору даних (вибірка з 10 профілів)

Показник	Еталонне значення	Отримано парсером	Відсоток точності	Примітки
Загальна кількість цитувань	5432	5432	100 %	Повна відповідність
h-індекс	98	98	100 %	Повна відповідність
i10-індекс	115	115	100 %	Повна відповідність
Кількість публікацій	488	481	98,6 %	Незначна розбіжність через групування версій статей у Google Scholar

Як видно з таблиці, точність збору ключових наукометричних індексів склала 100 %. Незначна розбіжність у кількості публікацій пов’язана з особливостями інтерфейсу Google Scholar, який іноді згортає кілька версій однієї статті в один запис. Отримані результати підтверджують високу точність та надійність розробленого модуля.

Також я провів тестування продуктивності, вимірюючи середній час, необхідний для повного збору даних для одного профілю. Вимірювання часу

виконання показали очікувану різницю у продуктивності між двома підходами до парсингу:

– середній час парсингу профілю ЛНТУ (використовуючи Requests): 1,8 секунди;

– середній час детального парсингу профілю Google Scholar (використовуючи Selenium): 17,5 секунд.

Ці результати демонструють, що використання Selenium є значно більш ресурсоємним, але необхідним для отримання повних даних з динамічних веб-сторінок.

Головним критерієм ефективності є порівняння автоматизованого процесу з ручним. За моїми оцінками, ручний збір, перевірка та структурування даних для одного профілю займає у відповідальної особи в середньому 15-20 хвилин. Розроблений мною модуль виконує те ж саме завдання в середньому за 20-25 секунд. Це свідчить про підвищення ефективності виконання задачі приблизно у 35-45 разів. Впровадження даного модуля дозволяє вивільнити значний обсяг робочого часу співробітників та значно підвищити точність та актуальність наукометричних даних, що використовуються в університеті.

3.3 Управління даними в проєкті

Для реалізації парсингу профіля НПП ЛНТУ я підійшов наступним чином, враховуючи, що я не використовував традиційну реляційну базу даних (як MySQL) для зберігання результатів парсингу на сервері.

Основним джерелом вхідних даних для ідентифікації профілів НПП ЛНТУ слугує JSON-файл lntu_faculty_urls.json. Цей файл я підготував заздалегідь, і він містить масив об’єктів, де кожен об’єкт представляє одного викладача і включає, як мінімум, його ім’я та пряме посилання на його персональну сторінку на сайті Рейтингу НПП ЛНТУ. Мій Flask-додаток читає цей файл для отримання URL-адрес, за якими потім відбувається парсинг інформації з сайту ЛНТУ. Цей файл фактично виконує роль простої файлової бази даних для початкових даних.

У лістингу 3.5 наведено приклад структури запису у файлі lntu_faculty_urls.json.

Лістинг 3.5 – Приклад структури даних у lntu_faculty_urls.json

```
        [
  {
    "name": "Прізвище Ім’я По-батькові",
    "lntu_profile_url": "https://rating2.lntu.edu.ua/staff/123",
    "lntu_photo_thumbnail":
"/sites/default/files/styles/thumbnail/public/pictures/picture-123-
12345.jpg"
  {
    "name": "Інше Прізвище Ім’я По-батькові",
    "lntu_profile_url": "https://rating2.lntu.edu.ua/staff/456",
    "lntu_photo_thumbnail":
"/sites/default/files/styles/thumbnail/public/pictures/picture-456-
67890.jpg",
    "gs_profile_url": "https://scholar.google.com/citations?user=ABCDEFG"
  } //Якщо потрібно
]
```

кінець лістингу 3.5

Дані, отримані в результаті парсингу сторінок ЛНТУ та Google Scholar моїми функціями в scholar_profiles.py, агрегуються та структуруються в Python-словники на стороні бекенду. Ці словники точно відповідають тій структурі, яку очікує мій React-фронтенд для відображення.

Ключовим моментом є те, що ці зібрані та оброблені дані не зберігаються персистентно на сервері у якійсь базі даних після кожного запиту. Замість цього, вони формуються відразу для кожного запиту /api/profile і одразу ж серіалізуються у формат JSON та відправляються як HTTP-відповідь клієнту (React-додатку). Такий підхід спрощує архітектуру бекенду для даної задачі, оскільки не вимагає налаштування, адміністрування та підтримки окремої СУБД для результатів парсингу. Вся актуальна інформація збирається безпосередньо з першоджерел під час запиту користувача.

Це означає, що якщо структура сайтів-джерел зміниться, або дані на них оновляться, наступний запит до мого парсера потенційно отримає вже оновлену інформацію, без необхідності синхронізації з якоюсь проміжною базою даних. З

іншого боку, це також означає, що кожен запит на отримання профілю ініціює новий процес парсингу, що може бути менш ефективним при дуже частих однакових запитах, але для цілей кваліфікаційної роботи та демонстрації функціоналу парсингу такий підхід є виправданим.

Важливою частиною управління даними в моєму проєкті, окрім їх збору та передачі, є процес очищення та нормалізації. Дані, отримані шляхом веб-скрапінгу, рідко бувають ідеальними. Вони часто містять зайві пробіли, символи нового рядка, непотрібні префікси або відсутні взагалі. Ігнорування цього етапу призвело б до некоректного відображення інформації на фронтенді та потенційних помилок.

Тому в функціях парсингу scholar_profiles.py я приділив увагу очищенню даних одразу після їх вилучення з HTML-коду. Для цього я активно використовував стандартні методи роботи з рядками в Python. Наприклад, метод .strip() застосовувався для видалення зайвих пробілів на початку та в кінці рядка з іменами, назвами публікацій та іншими текстовими полями. У лістингу 3.6 показано, як при парсингу даних з сайту ЛНТУ я не лише вилучав текст, а й одразу очищував його від службових слів.

Лістинг 3.6 – Приклад очищення даних під час парсингу

```
work_info_block = soup.select_one("div.field--name-field-zagalna-
informaciya ...")
if work_info_block:
    divs = work_info_block.find_all("div", recursive=False)

    # Вилучення та очищення назви факультету
    if len(divs) > 1 and divs[1].p:
        lntu_data["lntu_faculty"] =
divs[1].p.text.strip().replace("Факультет", "").strip()

    # Вилучення та очищення назви кафедри
    if len(divs) > 2 and divs[2].p:
        lntu_data["lntu_department"] =
divs[2].p.text.strip().replace("Кафедра", "").strip()

    # Вилучення та очищення назви посади
    if len(divs) > 3 and divs[3].p:
```

```
        lntu_data["lntu_position"] =
divs[3].p.text.strip().replace("Посада", "").strip()
```

кінець лістингу 3.6

Окрім очищення, важливим етапом є нормалізація даних, тобто приведення їх до єдиного, узгодженого формату. Мій бекенд виступає в ролі нормалізатора, який отримує дані з двох абсолютно різних джерел (сайт ЛНТУ та Google Scholar) і формує з них єдину структуру для JSON-відповіді. Наприклад, поле name з Google Scholar та lntu_full_name з сайту ЛНТУ, хоч і позначають одну й ту ж сутність, можуть мати різний формат. Моя система агрегує їх і надає фронтенду в стандартизованому вигляді.

Також я обробляв випадки, коли певні дані на сторінці відсутні. Замість того щоб генерувати помилку код присвоює відповідному полю значення None (або порожній список для масивів), що потім коректно обробляється на стороні React. Це робить систему більш стійкою та запобігає падінню всього додатку через відсутність одного некритичного елемента на сторінці профілю. Цей етап очищення та нормалізації є невидимим для кінцевого користувача, але критично важливим для забезпечення якості та цілісності даних, що відображаються в інтерфейсі.

3.4 Захист інформаційно-комп’ютерної системи

При розробці мого модуля парсингу, хоча він переважно призначений для локального використання або використання у внутрішньому контурі університету, я враховував деякі базові аспекти безпеки, особливо стосовно взаємодії з зовнішніми веб-ресурсами та обробки даних.

Безпека взаємодії з зовнішніми ресурсами:

– при надсиланні HTTP-запитів до сайту ЛНТУ та Google Scholar я встановлюю реалістичний заголовок User-Agent, щоб ідентифікувати мій парсер як стандартний веб-браузер. Це допомагає уникнути негайного блокування

запитів, які можуть розцінюватися як автоматизовані, якщо User-Agent не вказано або він є типовим для ботів;

– між послідовними запитами до одного й того ж домену (особливо до Google Scholar) я програмно вставив випадкові затримки (time.sleep(random.uniform(...))). Це зменшує навантаження на цільові сервери та знижує ризик спрацювання автоматичних систем захисту від надто частих запитів з однієї IP-адреси;

– у коді є базова перевірка на наявність ознак CAPTCHA або повідомлень про «незвичний трафік» у відповіді від Google Scholar. Якщо такі ознаки виявлено, парсер припиняє роботу для даного запиту та логує цю подію, замість того щоб намагатися обійти CAPTCHA, що було б значно складніше та виходило за рамки моєї задачі. Також у мене є жорстко прописані cookies у коді app.py для доступу до Google Scholar, з попередженням про необхідність їх оновлення, що є тимчасовим рішенням для обходу деяких обмежень під час розробки, але для продуктивного використання потребувало б більш надійного механізму автентифікації або управління сесіями, якщо це можливо для Google Scholar API (якого офіційно немає).

Безпека API та веб-додатку:

– хоча детальна валідація всіх вхідних параметрів в /api/profile не була глибоко реалізована, я перевіряю наявність необхідних параметрів;

– оскільки мій Flask API повертає JSON і не генерує HTML безпосередньо на основі неконтрольованого вводу, ризик XSS на стороні API мінімальний. React, який я використовую на фронтенді, також має вбудовані механізми захисту від XSS, екрануючи дані за замовчуванням при рендерингу;

– для будь-якого розгортання у відкритій мережі критично важливим є використання HTTPS для шифрування трафіку між клієнтом та сервером. Це забезпечується встановленням SSL/TLS сертифікату на сервері, де буде розміщено додаток.

Конфіденційність даних:

– логування лише необхідної для діагностики інформації, уникаючи запису надто чутливих даних у відкритому вигляді у лог-файли;

– додаток намагається коректно обробляти винятки та повертати інформативні, але не надто детальні повідомлення про помилки на фронтенд, щоб не розкривати внутрішню структуру системи або потенційні вразливості.

3.5 Реалізація обробки складних випадків та виняткових ситуацій

При розробці модуля парсингу я розумів, що надійний програмний продукт повинен коректно працювати не лише в ідеальних умовах, але й вміти обробляти складні та виняткові ситуації. У контексті веб-скрапінгу це особливо актуально через неоднозначність даних та наявність механізмів захисту на сайтах-джерелах. Тому значну увагу я приділив реалізації логіки для таких випадків.

Однією з головних проблем при автоматизованому пошуку є наявність тезок – науковців з однаковими або схожими іменами. Мій модуль вирішує цю проблему шляхом делегування фінального рішення користувачеві. Якщо пошук за ПІБ в Google Scholar повертає декілька потенційних профілів, мій Python-бекенд не намагається вгадати правильний, а збирає короткі дані про кожного кандидата (ПІБ, афіліація, фото) і відправляє цей список на фронтенд (рис. 3.1).

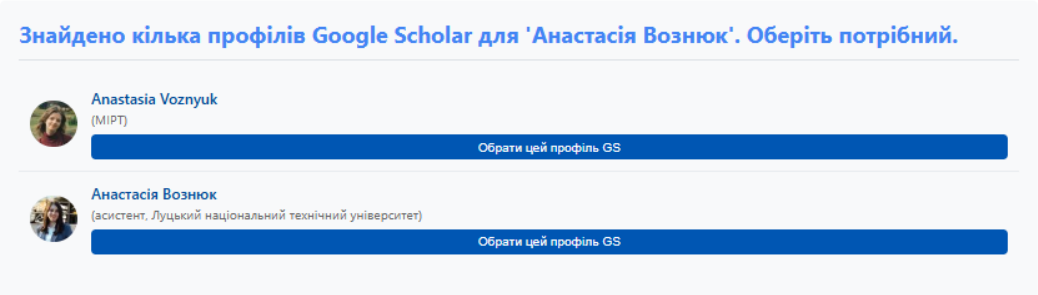


Рисунок 3.1 – Вигляд обробки кількох знайдених профілів

У лістингу 3.7 показано фрагмент коду з app.py, що реалізує цю логіку.

Лістинг 3.7 – Код на Python для обробки кількох знайдених профілів

```
gs_profiles_list = get_author_profiles_from_search(name_for_gs_lookup,
..., max_results_to_return=5)

if gs_profiles_list and len(gs_profiles_list) > 1:
    gs_data_collected = {
        'author_profiles': gs_profiles_list,
        'name_searched_gs': name_for_gs_lookup,
        'message_gs': f"Знайдено кілька профілів Google Scholar для
'{name_for_gs_lookup}'. Оберіть потрібний."
    }
```

кінець лістингу 3.7

Після отримання такого списку, React-фронтенд динамічно відображає його користувачеві, надаючи можливість зробити вибір. У лістингу 3.8 показано фрагмент JSX-коду, що відповідає за рендеринг цього списку.

Лістинг 3.8 – Код на Python для обробки кількох знайдених профілів

```
{profileData.author_profiles && profileData.author_profiles.length > 0 &&
(
  <div className="gs-profile-content profile-sub-section">
    <h3>{profileData.message_gs}</h3>
    <ul className="author-profiles-list">
      {profileData.author_profiles.map((p) => (
        <li key={p.profile_url} className="author-profile-item">
          {p.thumbnail && <img src={p.thumbnail} alt={p.name}
className="author-thumbnail" />}
          <div className="author-info">
            <span className="author-name">{p.name}</span>
            {p.affiliation && <span className="author-
affiliation">({p.affiliation})</span>}
            <button onClick={() => handleSelectGSProfileFromList(p)}
className="select-profile-button">
              Обрати цей профіль GS
            </button>
          </div>
        </li>
      ))}
    </ul>
  </div>
)}
```

кінець лістингу 3.8

Натискання кнопки «Обрати цей профіль GS» ініціює новий запит до API, але вже з точним gs_profile_url, що дозволяє отримати детальну інформацію про конкретного науковця.

Якщо для введеного ПІБ не вдається знайти жодного профілю в Google Scholar, моя система не повертає порожній результат. Замість цього вона виконує загальний пошук публікацій за цим ПІБ. Це дозволяє надати користувачеві хоча б якусь релевантну інформацію про наукову діяльність особи. Ця логіка реалізована в else блоці у функції api_profile, де викликається функція get_scholar_results.

Зрозумівши що під час взаємодії з зовнішніми сайтами можуть виникати різноманітні помилки: відсутність сторінки (помилка 404), проблеми з сервером або спрацювання захисних механізмів, як-от CAPTCHA. Для забезпечення стабільності, я обгорнув основні блоки коду, що виконують запити та парсинг, у конструкції try...except.

У лістингу 3.9 наведено приклад обробки винятків та перевірки на CAPTCHA у функції get_scholar_results.

Лістинг 3.9 – Код на Python для обробки кількох знайдених профілів

```
def get_scholar_results(name, headers, max_results=10):
    results = []
    try:
        response = requests.get(url, headers=headers, timeout=20)

        if response.status_code != 200:
            logging.error(f"HTTP {response.status_code} в
get_scholar_results для ’{name}’")
            return []
        page_content_lower = response.text.lower()
        if "captcha" in page_content_lower or "unusual traffic" in
page_content_lower:
            logging.error(f"CAPTCHA/незвичний трафік для ’{name}’ у
get_scholar_results.")
            return []
```

кінець лістингу 3.9

Такий підхід дозволяє моєму модулю завершувати роботу при виникненні проблем, не «падаючи» і не повертаючи непередбачуваний результат. На фронтенді також реалізоване відображення повідомлень про помилки, щоб користувач розумів, що пішло не так.

Реалізація цих механізмів обробки складних та виняткових ситуацій значно підвищила надійність та практичну корисність розробленого мною програмного модуля.

Висновки до розділу 3

У цьому розділі детально розглянуто процес реалізації мого програмного модуля для парсингу наукометричних даних НПП ЛНТУ. Основою для бекенд-частини послужили мова програмування Python та мікрофреймворк Flask, що дозволило створити гнучкий API для взаємодії з клієнтською частиною. Для безпосереднього збору даних з веб-сторінок використана комбінацію бібліотек: Requests та BeautifulSoup4 для роботи зі статичним HTML-контентом сайту ЛНТУ та деяких сторінок Google Scholar, а також Selenium для парсингу динамічно завантажуваних профілів Google Scholar.

Я описав структуру мого проєкту, ключові файли (app.py, scholar_profiles.py) та модулі, а також логіку роботи основних ендпоінтів API. Початкові дані про викладачів завантажуються з JSON-файлу, тоді як результати парсингу формуються відразу та передаються на фронтенд також у форматі JSON, без використання традиційної бази даних для їх зберігання на сервері.

Фронтенд-частина реалізована на React, створивши користувацький інтерфейс для пошуку науковців, відображення агрегованої інформації з сайту ЛНТУ та Google Scholar, а також для навігації між різними розділами даних.

Окрема увага приділена питанням тестування та налагодження системи, описавши використані підходи, такі як логування, аналіз HTML-коду сторінок, використання інструментів розробника в браузері та ручне тестування основних

сценаріїв. Також я розглянув базові аспекти інформаційної безпеки мого модуля, зокрема, при взаємодії з зовнішніми ресурсами та обробці даних.

Реалізований модуль демонструє успішне застосування обраного стеку технологій для вирішення поставленого завдання – створення інструменту для автоматизованого збору та представлення наукометричних даних. Хоча формальні UML-діаграми не розроблялися, розуміння архітектури та потоків даних було ключовим для реалізації. Проєкт готовий до демонстрації, має потенціал для подальшого розвитку, наприклад, шляхом інтеграції з іншими університетськими системами або розширення функціоналу аналізу зібраних даних

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було успішно вирішено актуальне науково-практичне завдання – розроблено програмний модуль для автоматизованого збору, обробки та представлення наукометричних даних науково-педагогічних працівників Луцького національного технічного університету з офіційного сайту університету та платформи Google Scholar.

У ході роботи було досягнуто поставленої мети та виконано всі визначені завдання.

Було проведено аналіз існуючих рішень та методів парсингу наукових профілів, включаючи спеціалізовані бібліотеки (наприклад, scholarly), комерційні API для скрапінгу та офіційні API інших наукометричних платформ. Аналіз показав, що попри наявність деяких універсальних інструментів, вони мають суттєві обмеження, пов'язані з точністю ідентифікації профілів та нестабільністю роботи.

На основі проведеного аналізу було сформовано детальний перелік вимог до програмного забезпечення. Функціональні вимоги охопили весь процес від автоматичного збору базової інформації та ключових наукометричних показників (h-індекс, цитування, публікації) до обробки виняткових ситуацій, таких як неоднозначність імен та помилки парсингу. Нефункціональні вимоги акцентували увагу на надійності системи, її продуктивності, простоті API для інтеграції та використанні інструментів з відкритим кодом.

Було спроєктовано багатокомпонентну архітектуру системи. Я обрав клієнт-серверний підхід з трьома основними компонентами: бекенд на Python з мікрофреймворком Flask, проксі-сервер на Node.js та клієнтський веб-інтерфейс на React. Такий поділ дозволив чітко розмежувати відповідальність та забезпечити гнучкість системи.

Було практично реалізовано програмний модуль. Я розробив бекенд-логіку на Python для збору даних, застосувавши комбінований підхід: бібліотеки Requests та BeautifulSoup для парсингу статичних сторінок сайту ЛНТУ та

Selenium для обробки динамічного контенту профілів Google Scholar. Також було створено інтерактивний веб-інтерфейс на React, який взаємодіє з бекендом через API та візуалізує отримані дані для користувача. Було забезпечено обробку помилок та виняткових ситуацій, таких як неоднозначність імен, відсутність профілів та виявлення CAPTCHA, що підвищило стабільність роботи модуля.

Для забезпечення зручної та сучасної взаємодії з користувачем було створено інтерактивний веб-інтерфейс на базі бібліотеки React та успішно інтегровано його з бекендом. Ключовою особливістю інтерфейсу є його здатність динамічно візуалізувати складні та агреговані дані, він коректно відображає загальну інформацію профілю, обробляє сценарій вибору при неоднозначності імен, а також структурує списки публікацій та наукометричні показники у чіткі, розділені блоки з можливістю перемикання між ними, що значно покращує досвід користувача.

Для забезпечення стабільної та передбачуваної роботи модуля, було приділено особливу увагу забезпеченню обробки помилок та виняткових ситуацій. Так, для вирішення проблеми неоднозначності імен, коли для одного запиту знаходиться декілька профілів Google Scholar, система не намагається вгадати правильний, а делегує рішення користувачеві, відображаючи на веб-інтерфейсі список кандидатів для вибору. У випадку повної відсутності профілю науковця в Google Scholar, було реалізовано механізм відкату, за яким система автоматично переходить до пошуку загальних публікацій за ПІБ, надаючи користувачеві хоча б часткову інформацію замість порожнього результату.

Тестування розробленого модуля було проведено та оцінено його точність і ефективність. Експериментальне дослідження підтвердило високу точність збору ключових наукометричних індексів (100 %) та загальну точність на рівні 98,6 %. Аналіз ефективності показав, що автоматизований збір даних відбувається у 35-45 разів швидше порівняно з ручним методом.

Таким чином, усі поставлені завдання виконано в повному обсязі, а головну мету роботи – розробку функціонального модуля парсингу – досягнуто. Результатом роботи є готовий до використання програмний прототип, який

демонструє високу ефективність та точність. Розроблений модуль має практичну цінність для Луцького національного технічного університету, оскільки дозволяє автоматизувати та значно прискорити процес моніторингу наукової діяльності, що є важливим для звітності, акредитації та підвищення рейтингів університету. Перспективи подальшого розвитку проєкту включають інтеграцію з внутрішніми інформаційними системами університету, впровадження механізмів кешування для підвищення продуктивності та розширення аналітичних можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Scholar. URL: https://scholar.google.com (дата звернення: 01.04.2025).

2. Google Scholar profile image. Google Scholar. URL: https:// scholar.google.com/citations?hl=uk&user=3ks5LOcAAAAJ (дата звернення: 10.05.2025).

3. Tutorial Parsing. Mate academy. URL: https://mate.academy/blog/python /python-web-parser / (дата звернення: 05.04.2025).

4. ЛНТУ. URL: https://lntu.edu.ua/uk (дата звернення: 03.04.2025).

5. НПП ЛНТУ. ЛНТУ. URL: https://rating2.lntu.edu.ua/searchuser (дата звернення: 14.05.2025).

6. ORCID. URL: https://orcid.org (date of access: 14.05.2025).

7. Scopus. URL: https://www.scopus.com/home.uri (date of access: 14.05.2025).

8. Web Of Science. Clarivate. URL: https://www.webofscience.com/wos/ (date of access: 14.05.2025).

9. Requests. Pypi. URL: https://pypi.org/project/requests/ (date of access: 08.05.2025).

10. BeautifulSoup4. Pypi. URL: https://pypi.org/project/beautifulsoup4/ (date of access: 08.05.2025).

11. Python. URL: https://www.python.org (date of access: 03.04.2025).

12. Selenium. URL: https://www.selenium.dev (date of access: 08.04.2025).

13. Node.js. URL: https://nodejs.org/en (date of access: 08.05.2025).

14. Visual Studio Code. URL: https://code.visualstudio.com (date of access: 02.04.2025).

15. React. URL: https://react.dev (date of access: 14.05.2025).

16. Flask. URL: https: https://flask.palletsprojects.com/en/stable/ (date of access: 08.04.2025).