МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МЕЛЬНИК ВЛАДИСЛАВ РОМАНОВИЧ

Допускається до захисту:

в.о. завідувача кафедри
інформаційних технологій

канд. техн. наук, доцент

_____ О. В. Зелінська

«____»_____ 2025 р.

СИСТЕМА ІНФОРМУВАННЯ ПРО ЗАХИСТИ PHD-ДИСЕРТАЦІЙ

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (бакалаврська) робота

Керівник:

Штовба С.Д.,

д-р техн. наук, професор

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2025

АНОТАЦІЯ

Мельник В.Р. Система інформування про захисти PhD-дисертацій. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

Ця бакалаврська робота присвячена розробці системи інформування, призначеної для автоматизованого збору та аналізу даних про захисти PhD-дисертацій з веб-сайту Національного агентства забезпечення якості освіти. Головне призначення системи полягає у забезпеченні швидкого доступу до актуальної інформації про дисертаційні захисти, моніторингу нових публікацій та тенденцій у науковій сфері. Робота включає аналіз існуючих підходів до веб-парсингу динамічних ресурсів та обґрунтування вибору технологічного стеку. Особливістю розробки є використання інструментів веб-скрапінгу (Selenium, BeautifulSoup, Pandas) для ефективного вилучення структурованих даних з динамічних веб-сторінок. Результатом є розроблений скрапінг-інструмент та інтерактивний графічний додаток, що дозволяє фільтрувати, переглядати зібрані дані та отримувати сповіщення про нові захисти на пошту. Використання системи допоможе науковцям, аналітикам та освітнім установам краще орієнтуватися у процесах забезпечення якості вищої освіти.

Ключові слова: веб-парсинг, скрапінг, PhD-дисертації, система інформування, Selenium, BeautifulSoup, НАЗЯВО.

95 ст. 8 рис., 1 табл., 3 дод., 23 джерел.

ABSTRACT

Melnyk V.R. Information System for PhD Dissertation Defenses. Specialty 122 "Computer Science". Vasyl' Stus Donetsk National University, Vinnytsia, 2025.

This bachelor's thesis is devoted to the development of an information system designed for automated collection and analysis of data regarding PhD dissertation defenses from the website of the National Agency for Higher Education Quality

Assurance. The main purpose of this system is to provide quick access to up-to-date information on dissertation defenses, monitor new publications, and identify trends in the scientific field. The work includes an analysis of existing approaches to web scraping dynamic resources and a justification of the chosen technology stack. A unique feature of the development is the use of web scraping tools (Selenium, BeautifulSoup, Pandas) for efficient extraction of structured data from dynamic web pages. The result is a developed scraping tool and an interactive graphical application that allows filtering, viewing collected data, and receiving email notifications about new defenses. The use of this system will help researchers, analysts, and educational institutions better navigate the processes of higher education quality assurance.

Key words: web parsing, scraping, PhD dissertations, information system, Selenium, BeautifulSoup, NAQA.

95 p., 8 fig., 1 tbl., 3 app., 23 ref.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПАРСИНГУ ВЕБ-РЕСУРСІВ 10
1.1 Поняття парсингу даних та його значення в сучасному інформаційному просторі 10
1.2. Огляд сучасних технологій та програмних засобів для автоматизованого збору даних з веб-сайтів 12
1.3. Загальна характеристика веб-ресурсу НАЗЯВО та його функціональне призначення 15
1.4. Дослідження структури веб-сторінок та формату представлення інформації про захисти дисертацій 17
1.5 Порівняльний аналіз інструментів для реалізації веб-парсингу та обґрунтування вибору 19
Висновки до Розділу 1. 22
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СКРАПІНГ-ІНСТРУМЕНТУ ДЛЯ ЗБОРУ ДАНІХ 23
2.1. Оцінка технічних можливостей та обмежень для автоматизованого збору даних з сайту НАЗЯВО 23
2.2 Вибір архітектурного підходу та обґрунтування технологічного стеку розробки 25
2.3 Детальна розробка алгоритму парсингу даних з урахуванням особливостей веб-сайту 27
2.4 Реалізація програмного коду скрапінг-інструменту 30
2.5 Проведення тестування, збір тестових та основних даних 32
2.6 Аналіз виявлених проблем під час розробки та збору даних, та запропоновані шляхи їх вирішення 33
Висновки до Розділу 2. 35
РОЗДІЛ 3. ГРАФІЧНИЙ ДОДАТОК: ІНТЕРФЕЙС ТА ІНСТРУКЦІЯ З ВИКОРИСТАННЯ 36
3.1 Інтерфейс користувача розробленого додатку 36
3.2 Інструкція з використання розробленого додатку 38
Висновки до Розділу 3. 43
ВИСНОВКИ 44

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46
ДОДАТКИ 48
ДОДАТОК А 49
ДОДАТОК В 56
ДОДАТОК С 58

ВСТУП

Актуальність теми дослідження: У сучасному світі обсяги даних, що генеруються та публікуються, зростають з неймовірною швидкістю. Мережа Інтернет стала найбільшим у світі сховищем інформації, що охоплює практично всі аспекти людської діяльності та знань. Веб-ресурси виступають як первинне джерело даних для науковців, аналітиків, бізнесменів, журналістів та широкої громадськості. Однак, отримання цієї інформації у форматі, зручному для автоматизованої обробки та аналізу, часто є складним завданням. Традиційні підходи до збору інформації з веб-сайтів, що передбачають ручне копіювання та вставку даних, є вкрай неефективними, вимагають значних трудовитрат та схильні до помилок. Це особливо критично, коли необхідно працювати з великими обсягами даних, регулярно оновлювати інформацію або відстежувати зміни на веб-ресурсах. У таких умовах автоматизовані методи збору даних, відомі як веб-парсинг або веб-скрапінг, стають не просто зручним інструментом, а необхідністю.

Особливої актуальності застосування технологій веб-парсингу набуває у контексті доступу до інформації, що публікується на офіційних державних та освітніх ресурсах. Ці сайти часто містять цінні дані, які можуть бути використані для підвищення прозорості, ефективності управління та проведення наукових досліджень. Веб-сайт Національного агентства забезпечення якості освіти (НАЗЯВО) за адресою https://svr.naqa.gov.ua/ є яскравим прикладом такого ресурсу. Він містить значний обсяг інформації, що стосується процесів забезпечення якості вищої освіти в Україні, зокрема детальні дані про захисти дисертацій. Ця інформація є надзвичайно цінною для широкого кола зацікавлених сторін: для науковців, які можуть аналізувати тенденції розвитку наукових напрямків, виявляти актуальні теми досліджень, досліджувати публікаційну активність здобувачів; для представників закладів вищої освіти, які можуть моніторити діяльність спеціалізованих вчених рад, аналізувати

успішність своїх випускників; для здобувачів, які шукають інформацію про попередні захисти за своєю спеціальністю, ознайомлюються з вимогами та процедурами; для аналітиків, які можуть оцінювати кадровий потенціал наукової сфери, проводити статистичний аналіз [1].

Незважаючи на публічний характер цієї інформації та її потенційну цінність, отримання її у структурованому вигляді для подальшого аналізу є ускладненим. Сайт НАЗЯВО, наскільки відомо, не надає офіційного публічного програмного інтерфейсу (API), який би дозволяв зручний та масовий програмний доступ до даних про захисти дисертацій. Це означає, що для отримання значних масивів цієї інформації дослідникам та аналітикам доводиться або вдаватися до трудомісткого ручного збору, або використовувати методи автоматизованого парсингу. Саме тому розробка спеціалізованого скрапінг-інструменту, орієнтованого на автоматизований збір даних про захисти дисертацій саме з веб-сайту НАЗЯВО, є надзвичайно актуальним та практично значущим завданням. Такий інструмент не лише вирішить проблему ефективного доступу до цінної інформації, але й створить міцну основу для проведення глибокого кількісного та якісного аналізу процесів у сфері вищої освіти та науки в Україні. Результати такого аналізу можуть сприяти підвищенню прозорості, ефективності та якості освітніх та наукових процесів.

Мета роботи: Розробка та реалізація програмного інструменту для автоматизованого збору структурованих даних про захисти дисертацій з веб-сайту Національного агентства із забезпечення якості вищої освіти та створення на його основі системи інформування.

Завдання дослідження:

- Проаналізувати сучасні технології, програмні засоби та бібліотеки для веб-парсингу. Обґрунтувати вибір інструментів для парсингу динамічного сайту.
- Провести аналіз структури веб-сторінок сайту НАЗЯВО, що містять дані про захисти дисертацій. Визначити HTML-елементи з цільовою інформацією та формат її представлення.

- Оцінити технічні можливості та потенційні виклики автоматизованого збору даних з сайту НАЗЯВО, враховуючи динамічне завантаження контенту та можливі механізми захисту.
- Розробити алгоритм роботи скрапінг-інструменту для збору даних з сайту НАЗЯВО, враховуючи специфіку структури даних та обробку динамічного контенту.
- Реалізувати програмний код скрапінг-інструменту на Python з використанням бібліотек Selenium та BeautifulSoup відповідно до розробленого алгоритму.
- Провести тестування розробленого інструменту для перевірки коректності вилучення даних, обробки помилок та стійкості. Здійснити збір даних про захисти дисертацій.
- Проаналізувати проблеми, що виникли під час розробки та збору даних (наприклад, неповне завантаження, зміни структури). Запропонувати шляхи їх вирішення.
- Виконати первинну обробку та узагальнення зібраних даних: очищення, стандартизація форматів, структурування у DataFrame Pandas.
- Визначити та описати потенційні напрямки наукового використання зібраної інформації для досліджень у галузі освіти та науки.
- Розробити додаток з графічним інтерфейсом для завантаження, фільтрації та перегляду зібраних даних. Підготувати інструкцію з використання.

Об'єкт дослідження: У рамках кваліфікаційної роботи зосереджено увагу на дослідженні комплексу процесів, пов'язаних з автоматизованим отриманням та подальшим використанням інформації, яка знаходиться у відкритому доступі в мережі Інтернет. Таким чином, об'єктом дослідження є широке коло процесів автоматизованого збору, ефективної обробки та глибокого аналізу даних, що публікуються на різноманітних веб-ресурсах. Ці процеси розглядаються не як окремі, ізольовані етапи, а як цілісну, взаємопов'язану систему, яка включає в себе технічні аспекти (програмування, взаємодія з веб-технологіями),

методологічні підходи (алгоритми парсингу, методи аналізу даних) та прикладні аспекти (використання отриманих даних для конкретних цілей). Дослідження охоплює загальні принципи та підходи, що застосовуються в цій галузі.

Предмет дослідження: Предметом дослідження є більш конкретний та сфокусований аспект, який знаходиться в межах визначеного об'єкта. У даному випадку, цим предметом є специфічний набір структурованих даних, а саме – дані про захисти дисертацій, які публікуються та доступні для перегляду на офіційному веб-сайті Національного агентства забезпечення якості освіти (https://svr.naqa.gov.ua/). Досліджуються не лише самі ці дані як інформаційний ресурс, але й ті конкретні методи, програмні інструменти та технології, які дозволяють автоматизовано вилучати цю інформацію з веб-сторінок, перетворювати її у структурований формат, придатний для подальшої обробки, та використовувати для досягнення наукових цілей. Таким чином, предметом дослідження є саме цей конкретний набір даних про захисти дисертацій та весь комплекс пов'язаних з ним процесів автоматизації збору та використання.

Практичне значення одержаних результатів: Результати цього дослідження мають практичну цінність для науковців, аналітиків та освітніх установ. Створена система інформування про захисти PhD-дисертацій забезпечує автоматизований збір та структуроване представлення даних, що спрощує моніторинг, аналіз тенденцій та доступ до актуальної інформації. Це сприяє підвищенню прозорості у сфері вищої освіти та науки.

Структура кваліфікаційної роботи:

Бакалаврська робота включає в себе 95 сторінок, 8 рисунків і список літератури із 23 джерел.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПАРСИНГУ ВЕБ-РЕСУРСІВ

У першому розділі бакалаврської роботи поглиблюються фундаментальні теоретичні аспекти, що стосуються веб-парсингу, а також здійснюється детальний аналіз веб-сайту Національного агентства забезпечення якості освіти (НАЗЯВО) як джерела даних для системи інформування про захисти PhD-дисертацій. Це є необхідним кроком для формування міцної бази знань, яка дозволить ефективно спроектувати та реалізувати практичну частину дослідження. Метою є не просто перерахування визначень, а надання глибокого розуміння сутності цього процесу, його місця в сучасному інформаційному просторі та технологій, що його забезпечують, з урахуванням специфіки збору даних про дисертаційні захисти.

1.1 Поняття парсингу даних та його значення в сучасному інформаційному просторі

В умовах стрімкого розвитку цифрових технологій та повсюдного поширення мережі Інтернет, обсяги даних, що генеруються та зберігаються у відкритому доступі, зростають експоненційно. Веб-сайти стали основним джерелом інформації з найрізноманітніших галузей людської діяльності – від наукових публікацій та статистичних звітів до новин, цін на товари та відгуків користувачів. Ця інформація є надзвичайно цінною для проведення досліджень, прийняття обґрунтованих рішень, моніторингу тенденцій, розробки нових продуктів та послуг [17].

Однак, значна частина цієї інформації представлена у форматі, оптимізованому для візуального сприйняття людиною, а не для автоматизованої обробки. Веб-сторінки складаються з великої кількості розмітки (HTML, CSS, JavaScript), яка визначає зовнішній вигляд та інтерактивність, але не є безпосередньо структурованими даними, які можна було б легко імпортувати до бази даних чи електронної таблиці [2][4].

У ситуаціях, коли необхідно отримати структуровану інформацію з веб-ресурсів, ефективним рішенням виступає **веб-скрапінг** (також відомий як парсинг веб-даних). Це процес, що передбачає автоматичне витягування релевантних даних із веб-сторінок шляхом аналізу їхнього вмісту програмними засобами. Парсер не просто завантажує сторінку, а аналізує її кодову структуру (найчастіше HTML або JSON), визначаючи цільові елементи — наприклад, за допомогою тегів, CSS-класів або XPath — та отримує з них потрібну інформацію у форматі, зручному для подальшої обробки, наприклад, у вигляді таблиць [9].

На відміну від користувача, який переглядає сторінку у браузері, програмний парсер працює безпосередньо з кодом документа та мережею запитів між клієнтом і сервером. Це дозволяє автоматизувати збір великих обсягів відкритих даних [8].

У сучасних умовах цифровізації така технологія стає все більш затребуваною. Вона відкриває нові можливості в багатьох сферах:

Академічна наука: парсинг допомагає акумулювати дані для статистичних і тематичних досліджень — від аналізу публікацій до оцінки наукової активності установ.

Бізнес-середовище: компанії використовують скрапінг для моніторингу конкурентів, цін, ринку послуг, а також для аналізу зворотного зв’язку клієнтів та пошуку нових контактів.

Завдяки цьому інструменту організації можуть приймати обґрунтовані рішення, базуючись на актуальній та точно зібраній інформації.

Журналістика даних: Журналісти використовують парсинг для збору інформації з відкритих джерел (державні реєстри, сайти органів влади, соціальні мережі) з метою проведення розслідувань та створення аналітичних матеріалів на основі великих масивів даних.

Створення баз даних та інформаційних систем: Парсинг є ефективним способом наповнення баз даних актуальною інформацією з веб-ресурсів, яка потім може бути використана в різноманітних додатках, сервісах та інформаційних системах [19].

Моніторинг та оповіщення: Парсинг може бути використаний для регулярного моніторингу змін на веб-сторінках та автоматичного сповіщення користувачів про виявлені оновлення (наприклад, зміна ціни, поява нового товару, публікація новини).

Хоча парсинг дозволяє отримувати дані з веб-сайтів, він не замінює офіційні API, якщо такі доступні. API зазвичай більш стабільні, мають документацію та призначені для зручного доступу до даних. Проте, якщо публічний API відсутній або обмежений, як у випадку з об'єктом цього дослідження, парсинг стає єдиним практичним способом отримати необхідну інформацію. При цьому важливо дотримуватися етичних та правових норм: поважати правила сайту (наприклад, вказані в robots.txt), не створювати надмірне навантаження на сервер, уникати збору персональних даних без згоди та дотримуватись законів, таких як GDPR. У даній роботі використано відкриту інформацію з державного ресурсу, що мінімізує ризики, однак етичне використання даних залишається обов’язковим.

1.2. Огляд сучасних технологій та програмних засобів для автоматизованого збору даних з веб-сайтів

Індустрія веб-парсингу активно розвивається, і на сьогоднішній день існує велика кількість технологій, програмних засобів та бібліотек, які дозволяють автоматизувати процес збору даних з веб-сайтів. Вибір конкретного інструменту або технологічного стеку залежить від багатьох факторів, включаючи складність структури веб-сайту, наявність динамічного контенту (що генерується JavaScript), обсяг даних, що потрібно зібрати, необхідна швидкість парсингу, а також навички та переваги розробника.

Проведено детальний огляд найбільш популярних та ефективних підходів та інструментів, які застосовуються для автоматизованого збору даних з веб-сайтів:

Статичний парсинг (на основі HTTP-запитів та парсингу HTML): Цей підхід є найбільш простим та швидким, але ефективний лише для веб-сайтів, контент яких повністю або здебільшого міститься безпосередньо у вихідному

HTML-коді сторінки, який отримується за допомогою звичайного HTTP-запиту [20].

Бібліотеки для HTTP-запитів: Для завантаження вмісту веб-сторінок використовуються бібліотеки, що дозволяють здійснювати HTTP-запити (GET, POST тощо). У Python найпопулярнішою бібліотекою для цього є requests. Вона дозволяє легко надсилати запити, отримувати відповіді від сервера та працювати з заголовками, cookies тощо [18].

Бібліотеки для парсингу HTML/XML: Після отримання HTML-коду сторінки, використовуються бібліотеки для його аналізу та вилучення потрібних даних. Ці бібліотеки будують деревоподібну структуру документа (DOM - Document Object Model) або дозволяють шукати елементи за допомогою різних методів. Популярні бібліотеки в Python:

BeautifulSoup — це одна з найпоширеніших бібліотек у Python для обробки HTML- і XML-контенту, яка вирізняється простотою використання та здатністю коректно працювати навіть з неідеальною розміткою. Вона забезпечує зручні засоби для проходу по дереву документа і пошуку потрібних елементів за тегами, класами, ідентифікаторами або атрибутами. Іншою популярною бібліотекою є lxml — потужний інструмент, який дозволяє ефективно працювати з HTML і XML, підтримує запити через XPath і CSS-селектори та часто використовується разом із BeautifulSoup для підвищення швидкодії. PyQuery, у свою чергу, застосовує синтаксис, подібний до jQuery, і є зручною альтернативою для тих, хто знайомий із веб-розробкою. Варто враховувати, що багато сучасних веб-сайтів створені з використанням JavaScript, який завантажує дані динамічно, тому стандартний підхід, що базується лише на статичному HTML, може не дати повного результату. У таких випадках доцільно застосовувати інструменти, які імітують роботу справжнього браузера та виконують JavaScript-код — так звані headless-браузери. Це версії браузерів, які працюють без графічного інтерфейсу, наприклад Google Chrome або Mozilla Firefox у відповідному режимі, і ними можна керувати програмно. Одним із найвідоміших засобів для цього є бібліотека Selenium, яка дозволяє автоматизувати взаємодію з веб-сторінками: здійснювати

навігацію, шукати та обробляти елементи, виконувати скрипти та витягувати повний контент після завантаження. Завдяки цьому стає можливим витягнення динамічних даних, які з'являються лише після повного рендерингу сторінки.

Puppeteer (для Node.js): Бібліотека, розроблена Google, яка надає високоуровневий API для керування Chrome або Chromium через DevTools Protocol. Дуже ефективна для парсингу динамічних сайтів, створення скріншотів, генерації PDF тощо.

Playwright (для Python, Node.js, Java, C#): Розроблена Microsoft, є альтернативою Puppeteer та Selenium. Playwright підтримує різні браузери (Chromium, Firefox, WebKit) та надає потужний API для веб-автоматизації та парсингу динамічних сайтів [16].

Використання API: Як вже зазначалося, якщо веб-ресурс надає офіційний API (наприклад, RESTful API, що повертає дані у форматі JSON або XML), це є найбільш рекомендованим методом для отримання даних. Робота з API є більш стабільною, швидкою та менш схильною до змін у структурі веб-сторінки. Для роботи з API використовуються бібліотеки для здійснення HTTP-запитів (requests в Python) та бібліотеки для парсингу даних у форматі JSON або XML (вбудований модуль json в Python, xml.etree.ElementTree або lxml).

Спеціалізовані фреймворки для скрапінгу: Існують потужні фреймворки, які надають готову інфраструктуру для розробки складних скрапінг-проектів, включаючи управління чергою запитів, обробку помилок, зберігання даних, роботу з проксі-серверами, обробку cookies тощо.

Scrapy (Python): Потужний та гнучкий фреймворк для веб-скрапінгу та обробки даних. Він надає багато готових компонентів та дозволяє швидко розробляти масштабовані скрапінг-проекти. Scrapy може працювати як зі статичними, так і з динамічними сайтами (інтегруючись з інструментами типу Selenium або Splash) [23].

Онлайн-сервіси веб-скрапінгу: Існують комерційні та деякі безкоштовні онлайн-платформи, які надають інструменти (візуальні редактори, конструктори) та хмарну інфраструктуру для створення та запуску скрапінг-завдань без

необхідності написання коду або з мінімальним кодуванням. Приклади: ParseHub, Octoparse, Web Scraper (розширення для Chrome). Ці сервіси можуть бути зручними для простих завдань або для користувачів без навичок програмування, але можуть мати обмеження за функціональністю, обсягом даних або вартістю [21][22].

Вибір конкретної технології або інструменту залежить від специфіки веб-сайту, складності завдання та вимог до масштабованості та швидкості. Для даного дослідження, де об'єктом є динамічний веб-сайт, необхідно використовувати інструменти, здатні виконувати JavaScript.

1.3. Загальна характеристика веб-ресурсу НАЗЯВО та його функціональне призначення

Веб-сайт Національного агентства забезпечення якості освіти (НАЗЯВО) є офіційним електронним ресурсом, який відіграє ключову роль у забезпеченні прозорості та інформуванні громадськості про діяльність НАЗЯВО. Основне функціональне призначення сайту полягає у публікації та наданні доступу до інформації, пов'язаної з процесами забезпечення якості вищої освіти в Україні. Це включає, але не обмежується, інформацією про акредитацію освітніх програм, діяльність спеціалізованих вчених рад, а також, що є предметом даного дослідження, інформацією про захисти дисертацій.

З першого погляду на сайт, можна відзначити його сучасний дизайн та, ймовірно, використання передових веб-технологій. Інтерфейс виглядає динамічним, елементи сторінки завантажуються плавно, що є характерним для веб-додатків, побудованих з використанням сучасних JavaScript-фреймворків (наприклад, Angular, React, Vue.js). Це підтверджує попереднє припущення про необхідність використання інструментів, здатних виконувати JavaScript для отримання повного вмісту сторінки Рис 1.1.

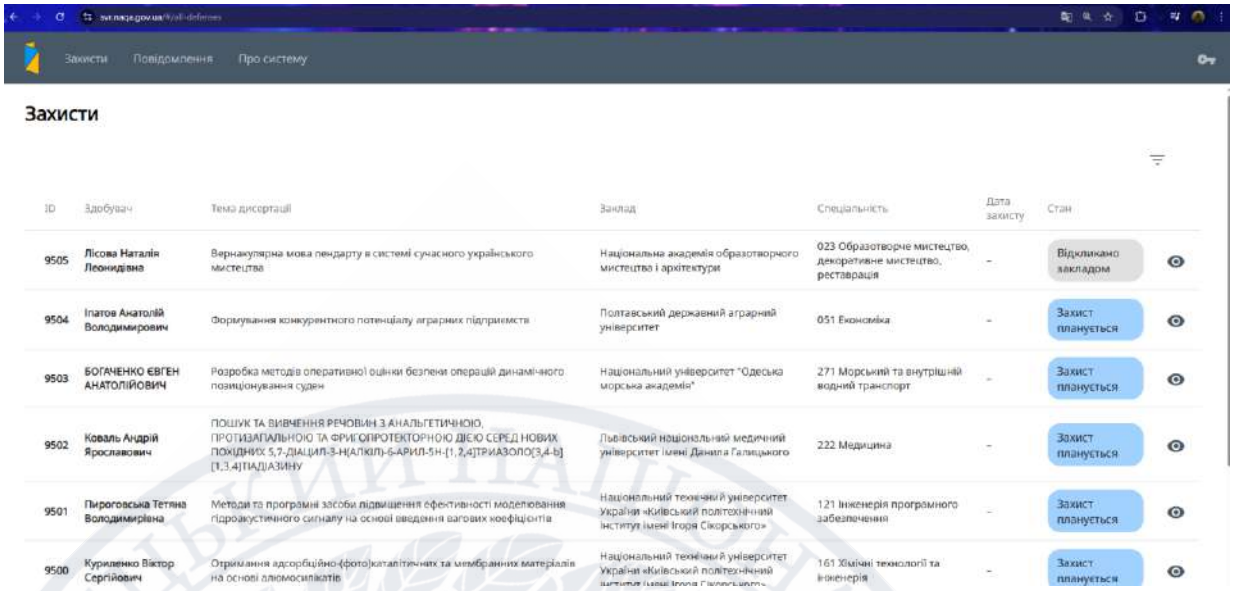


Рисунок 1.1 – Інтерфейс НАЗЯВО

Навігація по сайту, як правило, здійснюється через меню або інтерактивні елементи. Розділ, що цікавить, присвячений захистам дисертацій, і доступ до нього здійснюється за певним URL-адресою, яка, як з'ясовано, має вигляд https://svr.naqa.gov.ua/#/defense/ . Символ # у URL-адресі (хеш) часто вказує на те, що сайт є односторінковим додатком (Single Page Application), де різні "сторінки" насправді є різними станами одного HTML-документа, які змінюються за допомогою JavaScript без повного перезавантаження сторінки з сервера. Це ще раз підтверджує необхідність використання інструментів, що імітують роботу браузера.

Сайт є публічним ресурсом, що означає, що більша частина інформації, включаючи дані про захисти дисертацій, доступна без необхідності реєстрації або автентифікації користувача. Це значно спрощує процес парсингу, оскільки не виникає необхідності реалізовувати механізми входу на сайт або управління сесіями.

Важливо також зазначити, що сайт, як офіційний державний ресурс, має відповідати певним стандартам доступності та інформаційної безпеки. Хоча глибокого аналізу безпеки не проводилось, передбачається, що сайт може мати базові механізми захисту від автоматизованого доступу, такі як обмеження

частоти запитів або виявлення аномальної активності. Ці аспекти необхідно врахувати при розробці скрапінг-інструменту, щоб уникнути блокування.

1.4. Дослідження структури веб-сторінок та формату представлення інформації про захисти дисертацій

Для ефективного вилучення даних про захисти дисертацій, було детально досліджено структуру веб-сторінок, на яких ця інформація представлена, та формат її подання. Використовувались інструменти розробника, вбудовані у веб-браузер (наприклад, Chrome DevTools), для аналізу HTML-коду, CSS-селекторів та мережевих запитів.

При переході за посиланням, спостерігається список захистів. Кожен елемент списку, ймовірно, є посиланням на окрему сторінку з детальною інформацією про конкретний захист. URL-адреса окремого захисту має вигляд https://svr.naqa.gov.ua/#/defense/ID , де ID є унікальним числовим ідентифікатором захисту. Це спостереження є ключовим, оскільки воно дозволяє отримати доступ до інформації про кожен захист, послідовно перебираючи можливі значення ID. Припускається, що ID є послідовними, або принаймні зростаючими з часом, що дозволяє ефективно здійснювати збір даних, починаючи з найбільших відомих ID Рис 1.2.

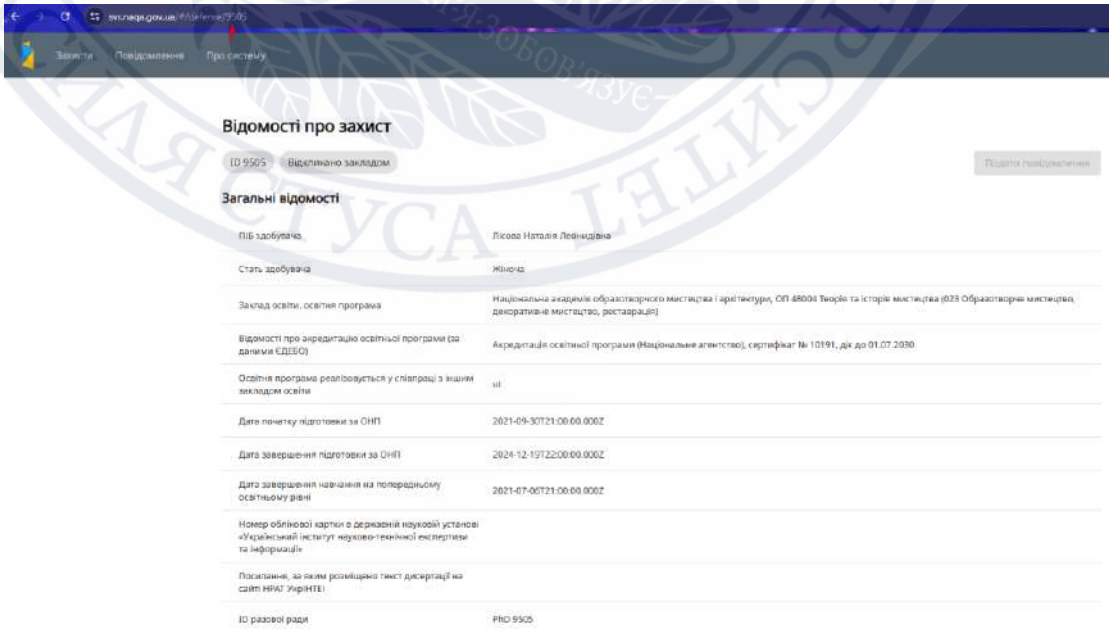


Рисунок 1.2 – інформацією про конкретний захист 9505

При відкритті сторінки окремого захисту, деталі організовані у вигляді блоків інформації. Візуально інформація представлена у зручному для читання форматі, з чіткими мітками (label) для кожного поля даних. Аналіз HTML-коду сторінки за допомогою інструментів розробника показав, що дані, ймовірно, розміщені у табличній структурі (<table>, <tr>, <td>) або у вигляді списків (<ul>, <li>), де мітка поля та його значення знаходяться у сусідніх елементах. Наприклад, очікується побачити щось на зразок <td class="label-column">ПІБ здобувача:</td><td class="value-column"> Іванов Іван Іванович </td>. Використання CSS-класів (label-column, value-column) та структури HTML (td поруч з td) є типовим для веб-розробки і дозволяє легко знаходити потрібні елементи за допомогою CSS-селекторів або XPath-виразів.

Також звернено увагу на те, що деякі блоки інформації, такі як "Публікації здобувача за темою дисертації" та "Разова рада", представлені у вигляді списків або таблиць всередині основної сторінки захисту. Кожен елемент цих списків (окрема публікація або член ради) також містить декілька полів даних (наприклад, тип публікації, опис, рік, DOI; або ПІБ члена ради, тип, установа). Ці вкладені структури також необхідно буде парсити окремо. Деякі поля, такі як "Посилання на роботу" або посилання на профілі ORCID членів ради, є гіперпосиланнями (<a> теги), з яких необхідно вилучати атрибут href, що містить URL-адресу.

Важливим аспектом є те, що частина контенту, як припускається, завантажується асинхронно за допомогою JavaScript. Це означає, що при першому завантаженні сторінки, деякі елементи можуть бути відсутні в HTML-коді, отриманому простим HTTP-запитом. Вони з'являються лише після того, як браузер виконає скрипти та, можливо, здійснить додаткові запити до сервера для отримання даних. Саме тому використання Selenium, який імітує роботу браузера, є критично важливим. Selenium дозволить дочекатися повного завантаження та рендерингу сторінки, включаючи виконання всіх JavaScript, перш ніж передати HTML-код для парсингу BeautifulSoup [5].

Формат представлення даних, як правило, текстовий. Дати та час представлені у стандартному форматі, який можна буде парсити. Ключові слова, ймовірно, представлені у вигляді тексту, можливо, розділеного комами або іншими розділювачами. Поля "Публікації" та "Разова рада", як вже згадувалося, містять структуровані дані, які необхідно буде обробляти як списки об'єктів (наприклад, списки словників у Python).

1.5 Порівняльний аналіз інструментів для реалізації веб-парсингу та обґрунтування вибору

Після огляду сучасних технологій та програмних засобів для веб-парсингу, я провів порівняльний аналіз найбільш придатних інструментів для реалізації моєї кваліфікаційної роботи, враховуючи, що сайт НАЗЯВО, ймовірно, використовує динамічне завантаження контенту. Основними кандидатами для парсингу динамічного контенту є бібліотеки, що працюють з headless-браузерами, такі як Selenium, Puppeteer та Playwright, а також фреймворки типу Scrapy з відповідними інтеграціями.

На основі проведеного аналізу, я прийняв рішення використовувати наступний технологічний стек для розробки скрапінг-інструменту див. табл. 1.1:

Таблиця 1.1 Аналіз бібліотек

Інструмент / Технологія	**Переваги**	**Недоліки**	**Придатність для мого завдання**
Статичний парсинг (requests + BeautifulSoup/lxml)	Швидкий, простий у реалізації для статичних сайтів, низьке навантаження на сервер.	Не працює з динамічним контентом (згенерованим JavaScript), не може імітувати дії користувача.	Непридатний як основний метод, оскільки сайт НАЗЯВО є динамічним. Може бути використаний для парсингу

			статичних частин сторінки, якщо такі є.
Selenium	Повноцінна імітація роботи браузера, виконання JavaScript, взаємодія з елементами, підтримка різних браузерів, велика спільнота, багато ресурсів.	Відносно повільний (запускає повноцінний браузер), вимагає встановлення драйверів браузерів, може бути ресурсоємним, складніший у налаштуванні порівняно зі статичним парсингом.	**Висока.** Ідеально підходить для парсингу динамічного контенту на сайті НАЗЯВО. Дозволяє чекати завантаження елементів після виконання JavaScript.
Puppeteer (Node.js)	Швидкий (працює з Chromium через DevTools Protocol), ефективний для парсингу динамічних	Вимагає знання Node.js (JavaScript), орієнтований переважно на Chromium.	**Середня**. Потребує зміни мови розробки на JavaScript, що не є моєю основною мовою для цього проекту.

	сайтів, хороший API.		
Playwright	Підтримка різних браузерів (Chromium, Firefox, WebKit), швидкий, надійний, хороший API, підтримка різних мов (включаючи Python).	Новіший інструмент порівняно з Selenium, менша спільнота та кількість готових рішень на даний момент.	**Висока.** Дуже перспективний інструмент, який міг би бути використаний. Однак, я обрав Selenium через більшу кількість доступних навчальних матеріалів та прикладів на Python.

Python: Як основна мова програмування завдяки її простоті, гнучкості та наявності потужних бібліотек для веб-скрапінгу та аналізу даних [10][14].

Selenium: Для автоматизації роботи веб-браузера (Chrome) та отримання доступу до динамічного контенту на сайті НАЗЯВО. Selenium дозволить мені імітувати завантаження сторінки у браузері та чекати виконання JavaScript перед парсингом.

BeautifulSoup: Для зручного та ефективного парсингу HTML-коду сторінки, отриманого за допомогою Selenium. BeautifulSoup надає інтуїтивно зрозумілий API для навігації по дереву документа та пошуку потрібних елементів за допомогою CSS-селекторів [5].

Pandas: Для структурування зібраних даних у форматі DataFrame, що полегшить їх подальшу обробку, аналіз та збереження у файл (CSV) [3].

Цей вибір технологічного стеку є, на мою думку, оптимальним для вирішення поставлених у моїй кваліфікаційній роботі завдань. Selenium забезпечить можливість роботи з динамічним контентом, BeautifulSoup – ефективний парсинг HTML, а Pandas – зручну роботу з отриманими даними. Python як мова програмування об'єднає ці компоненти в єдиний інструмент.

Висновки до Розділу 1.

У першому розділі проведено аналіз веб-парсингу та його значення в зборі даних, а також розглянуто сучасні технології для автоматизованого збору інформації з веб-сайтів. Визначено, що для роботи з динамічним контентом найефективнішими є інструменти Selenium, BeautifulSoup і Pandas. Вони дозволяють здійснювати парсинг HTML і зручно обробляти дані у форматі DataFrame, що спрощує їх подальшу обробку та збереження. Описано веб-ресурс НАЗЯВО як основне джерело даних про захисти PhD-дисертацій, розглянуто структуру сайту та формат подання інформації. Виявлено, що сайт є односторінковим додатком з динамічним завантаженням контенту, де дані доступні за унікальними ID, що підтверджує необхідність використання веб-парсингу в умовах відсутності офіційного API.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СКРАПІНГ-ІНСТРУМЕНТУ ДЛЯ ЗБОРУ ДАНІХ

У цьому розділі кваліфікаційної роботи переходиться до безпосереднього опису процесу проектування та розробки програмного інструменту, який є центральним елементом дослідження. Детально викладено підхід до створення скрапінг-інструменту для збору даних про захисти дисертацій з веб-сайту Національного агентства забезпечення якості освіти, починаючи від вибору архітектури та закінчуючи аналізом результатів тестування та виявлених проблем ,також процес тестування розробленого скрапінг-інструменту, збір тестових та основних даних про захисти PhD-дисертацій з веб-сайту НАЗЯВО, а також аналіз виявлених проблем та запропонованих шляхів їх вирішення. Цей етап є критично важливим для підтвердження коректності роботи інструменту та якості зібраних даних, що є основою для подальшої розробки системи інформування..

2.1. Оцінка технічних можливостей та обмежень для автоматизованого збору даних з сайту НАЗЯВО

На основі детального аналізу структури та формату представлення інформації на сайті НАЗЯВО (Розділ 1.4), можна зробити обґрунтовану оцінку технічних можливостей та потенційних обмежень для реалізації автоматизованого збору даних.

Технічні можливості:

Доступ за унікальним ID: Наявність унікального числового ідентифікатора для кожного захисту, який використовується у URL-адресі, є значною перевагою. Це дозволяє отримати доступ до інформації про кожен захист без необхідності імітувати навігацію по списках або використовувати пошукові форми на сайті. Можна просто послідовно перебирати можливі значення ID.

Інформація на сайті представлена у досить впорядкованому форматі — з чіткими заголовками, таблицями та списками, що дозволяє ефективно

застосовувати інструменти HTML-парсингу, зокрема BeautifulSoup із CSS-селекторами або XPath. Оскільки ресурс побудовано на стандартних технологіях — HTML, CSS та JavaScript — його можна обробляти за допомогою доступних засобів, таких як Selenium. Ще однією перевагою є відкритий доступ до сторінок без авторизації, що суттєво спрощує автоматизацію. Водночас існують певні труднощі. Зокрема, оскільки сайт динамічно формує вміст за допомогою JavaScript, необхідно використовувати браузерну автоматизацію, що уповільнює процес і вимагає більше ресурсів.

Крім того, структура сторінок може змінюватися: це стосується класів, ID-елементів чи порядку розміщення блоків, що потребує регулярного оновлення парсера. Необхідно також враховувати можливість помилок — наприклад, недоступність окремих сторінок, затримки завантаження чи мережеві збої. Ще один важливий аспект — не перевантажувати сайт надмірною кількістю запитів, щоб уникнути блокування чи зниження продуктивності ресурсу. роботи сайту для інших користувачів або навіть до тимчасового чи постійного блокування IP-адреси парсера з боку адміністрації сайту. Необхідно суворо дотримуватись етичних норм парсингу, використовувати затримки між запитами та, можливо, розглянути використання проксі-серверів (хоча для даного завдання планується обмежитись затримками та відповідальним використанням).

Складність парсингу вкладених структур: Парсинг даних про публікації та членів разової ради, які представлені у вигляді вкладених списків або таблиць, вимагає більш складних алгоритмів парсингу порівняно з вилученням простих текстових полів.

Відсутність документації API: Відсутність офіційного API означає, що немає формального опису структури даних та методів доступу до них, що робить процес парсингу залежним від аналізу інтерфейсу користувача, який може змінюватися.

Незважаючи на ці потенційні обмеження та виклики, вважається, що технічні можливості для автоматизованого збору даних з сайту НАЗЯВО існують і є достатніми для реалізації кваліфікаційної роботи. Обраний технологічний стек

(Python, Selenium, BeautifulSoup, Pandas) є адекватним для подолання цих викликів, зокрема, для роботи з динамічним контентом та ефективного парсингу HTML.

2.2 Вибір архітектурного підходу та обґрунтування технологічного стеку розробки

На основі теоретичного аналізу (Розділ 1.2) та детального дослідження сайту НАЗЯВО (Розділ 1.4), зроблено висновок, що для успішного збору даних необхідний інструмент, здатний ефективно працювати з динамічним контентом, який генерується за допомогою JavaScript. Це одразу виключило можливість використання простих методів статичного парсингу, що базуються лише на HTTP-запитах та аналізі початкового HTML-коду.

З огляду на особливості завдання було обрано підхід, заснований на емуляції роботи браузера — зокрема, із застосуванням headless-режиму. Такий підхід дає змогу завантажувати веб-сторінку у фоновому режимі та дочекатися повного виконання JavaScript-коду, що дозволяє отримати весь динамічний вміст. Основою реалізації став стек інструментів на базі мови програмування Python, оскільки вона є зручною для задач веб-скрапінгу, має велику кількість відповідних бібліотек і добре підходить для обробки структурованих даних [6].

Додатковою перевагою Python є активна спільнота, доступність документації та досвід автора у використанні цієї мови, що сприяло швидкій розробці. Для автоматизації взаємодії з браузером використано бібліотеку Selenium, яка підтримує керування різними оглядачами (у цьому випадку — Chrome) і забезпечує широкі можливості: від переходу між сторінками до роботи з елементами DOM. Однією з ключових функцій Selenium є вміння очікувати появу необхідних об'єктів на сторінці, що особливо важливо при парсингу ресурсів із динамічним завантаженням контенту. У підсумку, цей інструмент дає змогу отримати повний HTML-код після виконання скриптів, що робить його ідеальним для обробки складних веб-сторінок [12].

Бібліотека BeautifulSoup: Після того, як Selenium завантажить сторінку та виконає всі необхідні скрипти, отримується доступ до її повного HTML-коду. Для ефективного парсингу цього HTML-коду обрано бібліотеку BeautifulSoup. BeautifulSoup є дуже зручною та гнучкою бібліотекою для парсингу HTML та XML документів. Вона створює деревоподібну структуру документа, що дозволяє легко навігувати по ній та шукати потрібні елементи за допомогою різних методів, включаючи CSS-селектори, які є інтуїтивно зрозумілими для веб-розробників. BeautifulSoup є досить "терпимою" до некоректної або не повністю сформованої HTML-розмітки, що може бути корисним при роботі з реальними веб-сайтами [7].

Бібліотека Pandas: Зібрані дані про захисти дисертацій мають табличну структуру (ID, ПІБ, тема, дати тощо). Для ефективного зберігання, обробки та аналізу таких даних обрано бібліотеку Pandas. Pandas надає потужний та зручний інструмент – DataFrame, який є двовимірною табличною структурою даних з мітками для рядків та стовпців. Використання Pandas дозволяє легко структурувати зібрані дані, виконувати над ними різні операції (фільтрація, сортування, групування), а також зберігати дані у різні формати файлів, зокрема CSV, який є зручним для подальшого використання [15].

Вбудовані бібліотеки Python (json, ast, os, re, time): Для виконання допоміжних завдань, таких як робота з файлами, обробка тексту за допомогою регулярних виразів, перетворення даних у формат JSON або обробка рядків, що містять представлення структур даних, а також для додавання затримок у виконанні скрипта, використовувались стандартні вбудовані бібліотеки Python.

Обґрунтування такого вибору полягає у тому, що комбінація Selenium та BeautifulSoup на мові Python є класичним та перевіреним часом підходом для парсингу динамічних веб-сайтів. Selenium ефективно вирішує проблему динамічного контенту, а BeautifulSoup надає зручний та гнучкий інструмент для парсингу HTML. Pandas забезпечує ефективну роботу з отриманими даними. Цей стек є добре задокументованим, має велику спільноту підтримки та достатньо ресурсів для вивчення та вирішення можливих проблем [13].

2.3 Детальна розробка алгоритму парсингу даних з урахуванням особливостей веб-сайту

На основі аналізу сайту НАЗЯВО та обраного технологічного стеку, я розробив детальний алгоритм парсингу даних про захисти дисертацій. Алгоритм враховує специфіку сайту, зокрема, доступ до інформації за унікальними ID та динамічне завантаження контенту.

Основні кроки алгоритму парсингу:

1. **Ініціалізація:**

Імпортувати необхідні бібліотеки: selenium, BeautifulSoup, pandas, time, json, ast, os, re.

Встановити шлях до драйвера веб-браузера (наприклад, chromedriver.exe для Chrome).

Ініціалізувати екземпляр веб-драйвера Selenium (наприклад, webdriver.Chrome()). За потреби, налаштувати опції (наприклад, запуск у headless-режимі для роботи без візуального вікна браузера, хоча для налагодження зручніше бачити вікно).

Визначити базову URL-адресу сайту НАЗЯВО для захистів: https://svr.naqa.gov.ua/#/defense/.

Визначити початковий та кінцевий діапазон ID для парсингу. Я вирішив перебирати ID у зворотному порядку, починаючи з найбільшого відомого ID і рухаючись до менших, щоб спочатку збирати найновіші записи. Це також дозволяє ефективно зупинити парсинг, якщо послідовно трапляються неіснуючі ID, що може свідчити про досягнення кінця діапазону існуючих записів.

Ініціалізувати порожній список для зберігання зібраних даних (кожен елемент списку буде словником, що представляє дані про один захист).

2. **Цикл перебору ID:**

Запустити цикл, що ітерується від початкового ID до кінцевого ID зі зменшенням на 1 на кожній ітерації.

3. **Обробка окремого ID:**

Всередині циклу, для поточного ID:

Сформувати повну URL-адресу сторінки захисту шляхом конкатенації базової URL та поточного ID.

Використовуючи try-except блок для обробки можливих помилок (наприклад, проблем з мережею, відсутність сторінки, тайм-аут):

Веб-драйвер Selenium відвідує сформовану URL-адресу за допомогою driver.get(url).

Очікування завантаження динамічного контенту: Це критично важливий крок. Використовуючи WebDriverWait та expected_conditions, дочекатися, доки певний ключовий елемент на сторінці, який гарантовано з'являється після повного завантаження та виконання JavaScript (наприклад, елемент, що містить ID захисту або ПІБ здобувача), стане видимим або присутнім у DOM. Встановити розумний тайм-аут для очікування.

Після успішного очікування, отримати повний HTML-код сторінки за допомогою driver.page_source.

Створити об'єкт BeautifulSoup з отриманого HTML-коду: soup = BeautifulSoup(page_html, 'html.parser').

Парсинг даних: Використовуючи методи пошуку BeautifulSoup (наприклад, soup.find(), soup.find_all()) та відповідні CSS-селектори або XPath-вирази (які були визначені під час аналізу структури сторінки у Розділі 2), вилучити потрібні дані:

- ПІБ здобувача.
- Стать здобувача.
- Тему дисертації.
- Ключові слова.
- Дату початку підготовки за ОНП.
- Дату завершення підготовки за ОНП.
- Дату і час захисту.
- Місце захисту.
- Інформацію про заклад освіти, освітню програму.

- Відомості про акредитацію освітньої програми.
- Статус роботи.

Парсинг вкладених структур (Публікації, Разова рада): Знайти блоки, що містять інформацію про публікації та членів ради. Всередині цих блоків, знайти окремі елементи (рядки таблиці або елементи списку), що представляють одну публікацію або одного члена ради. Для кожного такого елемента, вилучити відповідні поля даних (тип, опис, посилання для публікацій; ПІБ, тип, установа, ORCID для членів ради). Зібрати ці дані у списки словників.

Зберегти вилучені дані для поточного захисту у словник, додавши також ID та URL сторінки.

Додати словник з даними про поточний захист до загального списку зібраних даних.

Скинути лічильник послідовних помилок, якщо сторінка успішно спарсена.

У блоці except (при виникненні помилки):

Зафіксувати тип помилки та ID, для якого вона виникла (наприклад, вивести повідомлення у консоль або записати у лог-файл).

Збільшити лічильник послідовних помилок.

Якщо кількість послідовних помилок досягла певного порогу (наприклад, 5), припустити, що подальші ID, ймовірно, не існують, і перервати цикл парсингу.

4. **Затримка між запитами:**

Після обробки кожного ID (незалежно від успіху парсингу), додати невелику затримку (time.sleep(секунди)) перед переходом до наступного ID. Це необхідно для зменшення навантаження на сервер сайту та уникнення блокування IP-адреси. Тривалість затримки може бути експериментально підібрана.

5. **Завершення парсингу:**

Після завершення циклу парсингу (або при його перериванні через послідовні помилки), закрити веб-драйвер Selenium за допомогою driver.quit().

Перетворити список зібраних словників на DataFrame Pandas.

Зберегти DataFrame у файл формату CSV (наприклад, defenses_data.csv), вказавши кодування (utf-8-sig) для коректного відображення українських символів.

Цей алгоритм є основою для програмної реалізації скрапінг-інструменту, яка буде описана у наступному підрозділі.

2.4 Реалізація програмного коду скрапінг-інструменту

Реалізація програмного коду скрапінг-інструменту була здійснена на мові програмування Python з використанням бібліотек, обраних на етапі проектування. Повний вихідний код скрипта парсингу представлений у Додатку А. Тут я опишу ключові фрагменти коду та їх функціональність.

Спочатку, я імпортував усі необхідні бібліотеки Рисунок 2.1:

```
import customtkinter as ctk
from tkinter import ttk, filedialog, messagebox
import pandas as pd
import webbrowser
import os
import json
import re
import time
from threading import Thread
import ast
```

Рисунок 2.1 – Бібліотеки

Далі, я налаштував веб-драйвер Selenium. Я використовував chromedriver, який потрібно завантажити окремо та вказати шлях до нього, або переконатись, що він знаходиться у PATH системи.

Налаштування драйвера Chrome

```
driver = webdriver.Chrome()
```

Я визначив базову URL та діапазон ID для парсингу. Для тестування можна використовувати невеликий діапазон Рисунок 2.2.

```python
base_url = "https://svr.naqa.gov.ua/#/defense/"
start_id = 9028
# Для тестування можна зменшити кількість ID
# Наприклад, end_id = start_id - 2 (щоб отримати 3 записи: 9028, 9027, 9026)
end_id = 1
delay_between_requests = 1
navigation_timeout = 30

all_defenses_data = []
```

Рисунок 2.2 – URL та діапазон ID для парсингу

Основна логіка парсингу реалізована у циклі, що перебирає ID у зворотному порядку. Всередині циклу – блок try-except для обробки помилок Рисунок 2.3.

```python
# --- Основний цикл парсингу ---
for i in range(start_id, end_id - 1, -1):
    current_id_str = str(i)
    url_to_parse = base_url + current_id_str
    parsed_data_for_id = {'ID': current_id_str, 'Посилання на роботу': url_to_parse}

    print(f"Обробка: {url_to_parse}")
    try:
        driver.get(url_to_parse)
        print("Спроба оновити сторінку (driver.refresh())...")
        driver.refresh()

        expected_id_text_on_page = f"ID {current_id_str}"
        print(f"Очікуємо текст '{expected_id_text_on_page}' на сторінці...")
        wait_condition_xpath = f"//*[normalize-space(text())='{expected_id_text_on_page}']"

        try:
            WebDriverWait(driver, navigation_timeout).until(
                EC.presence_of_element_located((By.XPATH, wait_condition_xpath))
            )
            print(f"Текст '{expected_id_text_on_page}' знайдено.")
        except Exception as wait_error:
            print(f"ПОМИЛКА ОЧІКУВАННЯ для ID {current_id_str}: Не вдалося знайти текст '{expected_id_text_
            all_defenses_data.append(parsed_data_for_id)
            print("-" * 30); time.sleep(delay_between_requests if i != end_id else 0); continue

        time.sleep(1.5)
        page_html = driver.page_source
        soup = BeautifulSoup(page_html, 'html.parser')

        parsed_data_from_function = parse_defense_data(soup, current_id_str, url_to_parse)
        parsed_data_for_id.update(parsed_data_from_function)
        all_defenses_data.append(parsed_data_for_id)

        pub_status = 'OK' if isinstance(parsed_data_for_id.get('Публікації'), list) and parsed_data_for_id.
        rada_status = 'OK' if isinstance(parsed_data_for_id.get('Разова рада'), list) and parsed_data_for_id
        print(f"ID: {parsed_data_for_id.get('ID')}, ПІБ: {parsed_data_for_id.get('ПІБ здобувача', 'N/A')},

    except Exception as e:
        print(f"Загальна ПОМИЛКА для ID {current_id_str}: Не вдалося завантажити або обробити сторінку. {e}")
        all_defenses_data.append(parsed_data_for_id)

    print("-" * 30)
    if i != end_id: print(f"Затримка {delay_between_requests} сек..."); time.sleep(delay_between_requests)
```

Рисунок 2.3 – Основна логіка парсингу

Цей код реалізує основний алгоритм парсингу. Деталі вилучення кожного конкретного поля (CSS-селектори, XPath) необхідно підбирати на основі аналізу структури сайту за допомогою інструментів розробника. У Додатку А представлений повний код з усіма необхідними селекторами, які я використовував.

2.5 Проведення тестування, збір тестових та основних даних

Процес тестування розробленого скрапінг-інструменту був важливим етапом для забезпечення його коректної роботи та надійності. Тестування проводилося ітеративно, паралельно з процесом розробки.

Тестування на невеликому наборі ID: Спочатку я тестував парсер на невеликому, заздалегідь визначеному наборі ID захистів. Цей набір включав:

- Захисти з різними статусами (планується, відбувся, скасовано).
- Захисти з різною кількістю публікацій (жодної, одна, декілька).
- Захисти з різним складом разової ради.
- Захисти з різним форматом представлення дат та інших полів.

Під час тестування я візуально перевіряв зібрані дані для кожного ID, порівнюючи їх з інформацією на веб-сторінці. Я звертав увагу на:

- Коректність вилучення текстових полів (ПІБ, тема, місце захисту тощо).
- Правильність парсингу дат та часу.
- Коректність вилучення посилань (URL-адрес).
- Правильність парсингу вкладених структур (публікації, члени ради) – чи правильно вилучаються всі елементи та їхні поля.
- Обробку відсутніх даних (якщо якесь поле на сторінці відсутнє, чи правильно парсер повертає порожнє значення або None).

Тестування обробки помилок: Я також тестував, як парсер реагує на різні типи помилок:

- Спроба спарсити неіснуючий ID (перевірка, чи правильно обробляється помилка HTTP або відсутність елементів на сторінці).
- Переривання з'єднання з мережею під час парсингу.
- Тайм-аут завантаження сторінки.

Я перевіряв, чи парсер коректно логує помилки та чи продовжує роботу (якщо помилка не є критичною), або чи правильно зупиняється при досягненні порогу послідовних помилок.

Збір тестових даних: Після успішного проходження базових тестів, я здійснив збір тестових даних на більшому діапазоні ID (наприклад, 100-200 записів). Цей етап дозволив виявити проблеми, які могли не проявитись на малому наборі даних, наприклад, неочікувані варіанти структури сторінки для певних записів або проблеми з продуктивністю.

Збір основних даних: Після доопрацювання парсера на основі результатів тестування, я здійснив збір основного масиву даних, запустивши скрипт на визначеному діапазоні ID (від [зазначити початковий ID] до [зазначити кінцевий ID]). Процес збору даних зайняв певний час, залежно від діапазону ID та встановленої затримки між запитами. Результатом цього етапу став CSV-файл, що містить зібрані дані про захисти дисертацій. Зразки зібраних даних представлені у Додатку В.

2.6 Аналіз виявлених проблем під час розробки та збору даних, та запропоновані шляхи їх вирішення

Під час процесу розробки, тестування та збору даних, я зіткнувся з декількома типовими проблемами, характерними для веб-парсингу динамічних сайтів. Аналіз цих проблем та пошук шляхів їх вирішення є важливою частиною моєї роботи.

1. **Проблема: Неповне завантаження сторінки або контент не з'являється після виконання JavaScript.**

Аналіз: Іноді, навіть після візуального завантаження сторінки, потрібні дані могли бути відсутні в HTML-коді, отриманому Selenium. Це могло бути пов'язано з тим, що JavaScript ще не завершив свою роботу, або дані завантажувалися асинхронно після певних подій на сторінці.

Шлях вирішення: Найефективнішим рішенням є використання явних очікувань Selenium (WebDriverWait з expected_conditions). Замість фіксованої паузи (time.sleep), я налаштував парсер чекати, доки певний елемент, який точно містить потрібні дані або з'являється після їх завантаження, стане видимим або присутнім у DOM. Наприклад, очікування наявності елемента з ПІБ здобувача

або елемента, що містить таблицю публікацій. Також, у деяких випадках, додавання невеликої фіксованої паузи після успішного очікування може допомогти переконатись, що весь контент повністю відрендерився.

2. **Проблема: Зміни у структурі веб-сторінки призводять до некоректної роботи селекторів.**

Аналіз: Веб-сайти можуть оновлюватися, і розробники можуть змінювати структуру HTML, назви CSS-класів або ідентифікатори елементів. Якщо парсер покладається на конкретні, жорстко закодовані селектори, такі зміни можуть призвести до того, що парсер не зможе знайти потрібні елементи та вилучити дані.

Шлях вирішення: Під час розробки я намагався використовувати більш стійкі до змін селектори, наприклад, XPath-вирази, які описують шлях до елемента у дереві документа відносно інших, більш стабільних елементів. Також важливо регулярно тестувати парсер на актуальній версії сайту та оперативно адаптувати селектори при виявленні змін. У довгостроковій перспективі, для підтримки парсера, необхідно відстежувати зміни на сайті.

3. **Проблема: Обробка помилок HTTP (наприклад, 404 Not Found) або інших винятків під час завантаження сторінки.**

Аналіз: При переборі ID можуть траплятися ID, для яких сторінка не існує, або виникають інші помилки мережі або завантаження. Якщо ці помилки не обробляти, скрипт може перервати свою роботу.

Шлях вирішення: Реалізація блоків try-except навколо коду, що виконує запит до сторінки та парсинг. Я перехоплював специфічні винятки Selenium (наприклад, TimeoutException) та загальні Exception. При виникненні помилки, вона логувалася (виводилося повідомлення у консоль), і парсер переходив до обробки наступного ID. Також я реалізував механізм підрахунку послідовних помилок: якщо кількість послідовних помилок досягала певного порогу, парсинг автоматично зупинявся, що свідчило про ймовірне досягнення кінця діапазону існуючих ID або наявності більш серйозної проблеми.

4. Ризик створення надмірного навантаження на сервер та блокування IP-адреси виникає через швидке послідовне відвідування великої кількості сторінок. Це може бути сприйнято як атака, що призводить до блокування IP. Для вирішення цієї проблеми я додав затримки між запитами, експериментально підібравши оптимальний час (наприклад, 2 секунди), що знижує навантаження на сервер і імітує поведінку реального користувача. Також важливо дотримуватись правил файлу robots.txt сайту.

5. Складність парсингу вкладених структур, таких як публікації та члени разової ради, полягає в тому, що вони представлені у вигляді списків або таблиць. Я створив окрему логіку парсингу, що дозволяє правильно обробляти ці елементи: після знаходження основного блоку, шукав вкладені елементи і витягував з них необхідні дані. Для зручності аналізу я перетворював списки словників у формат JSON-рядка або рядок-представлення списку. Це дозволило ефективно збирати дані з сайту НАЗЯВО.

Незважаючи на ці виклики, аналіз та впровадження відповідних шляхів вирішення дозволили мені розробити достатньо надійний та ефективний скрапінг-інструмент для збору даних з сайту НАЗЯВО

Висновки до Розділу 2.

У другому розділі було спроектовано та розроблено скрапінг-інструмент на Python з використанням Selenium, BeautifulSoup та Pandas для автоматизованого збору даних про захисти дисертацій з динамічного веб-сайту НАЗЯВО. Алгоритм враховує унікальні ID сторінок та динамічне завантаження контенту. Проведено тестування, що дозволило виявити та вирішити типові проблеми, такі як неповне завантаження сторінок та зміни у їх структурі, за допомогою явних очікувань та адаптації селекторів. Забезпечено обробку помилок та ризику блокування шляхом впровадження затримок між запитами. Результатом є успішний збір значного масиву структурованих даних, що формують основу для функціонування системи інформування про захисти PhD-дисертацій.

РОЗДІЛ 3. ГРАФІЧНИЙ ДОДАТОК: ІНТЕРФЕЙС ТА ІНСТРУКЦІЯ З ВИКОРИСТАННЯ

3.1 Інтерфейс користувача розробленого додатку

На цьому рисунку представлено загальний вигляд головного вікна розробленого графічного додатку. Ліворуч розташована панель керування, що включає секції "Парсинг даних з сайту" з елементами керування процесом парсингу, "Керування файлом (локально)" з кнопкою для завантаження CSV-файлу та інформацією про нього, "Мій кабінет" для налаштувань користувача та відстеження спеціальностей, а також блок "Фільтри для перегляду". Праворуч знаходиться основна робоча область, яка містить таблицю для відображення завантажених даних про захисти дисертацій та, під нею, секцію "Детальна інформація" для перегляду атрибутів обраного в таблиці запису.

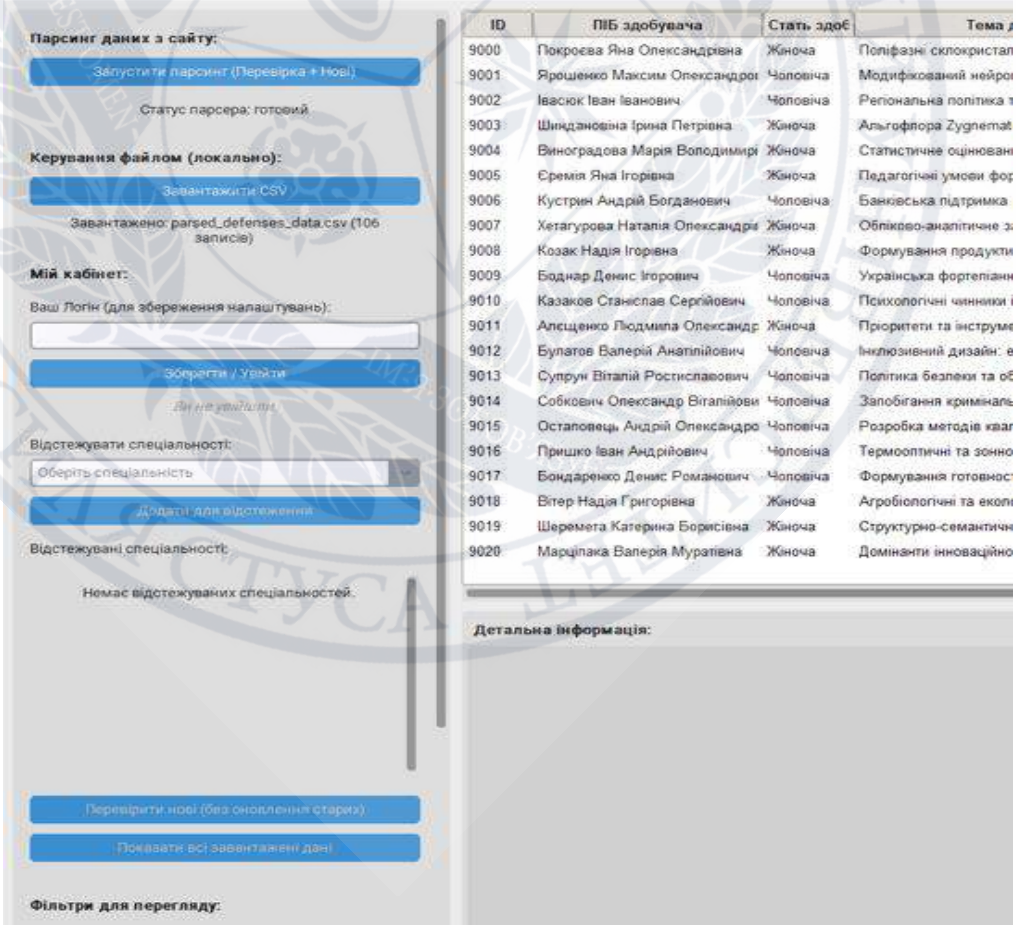


Рисунок 3.1 – Головне вікно програми для роботи з даними про захисти дисертацій

На цьому рисунку детально зображено блок "Фільтри для перегляду", розташований на панелі керування додатку. Цей блок надає користувачеві інструменти для гнучкого відбору та відображення даних. Він включає поле "Фільтр за спеціальністю", реалізоване як випадаючий список; текстові поля для введення критеріїв пошуку за "ПІБ здобувача", "Тема дисертації", "Ключові слова", "Заклад освіти"; та поле "Рік захисту" для введення року у форматі РРРР. Нижче розташовані кнопки "Застосувати фільтри" для активації заданих умов відбору та "Скинути фільтри" для очищення всіх критеріїв та повернення до повного списку даних.

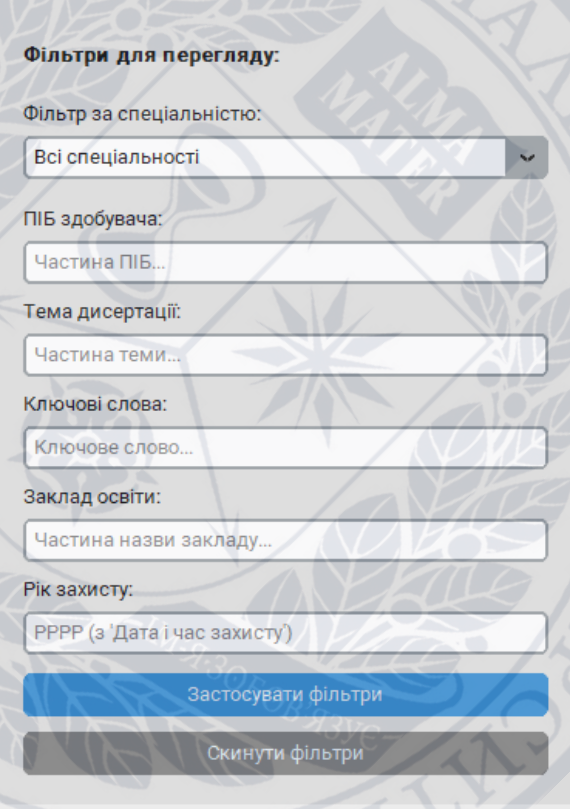


Рисунок 3.2 – Блок "Фільтри для перегляду" на панелі керування

Тут представлена панель "Детальна інформація". Вона активується, коли користувач обирає певний рядок в основній таблиці даних, та відображає повний набір атрибутів для цього конкретного запису про захист дисертації. На прикладі видно такі поля, як ID захисту (9008), ПІБ здобувача (Козак Надія Ігорівна), Стать здобувача (Жіноча), повна Тема дисертації, Ключові слова, що характеризують роботу, а також Дати початку та завершення підготовки за освітньо-науковою програмою. Поля "Дата і час захисту" та "Місце захисту" на даному прикладі

містять позначку "дані відсутні", що вказує на відсутність відповідної інформації у зібраних даних для цього запису. Ця панель може також відображати інші зібрані атрибути та, за наявності великого обсягу інформації, матиме можливість прокрутки.

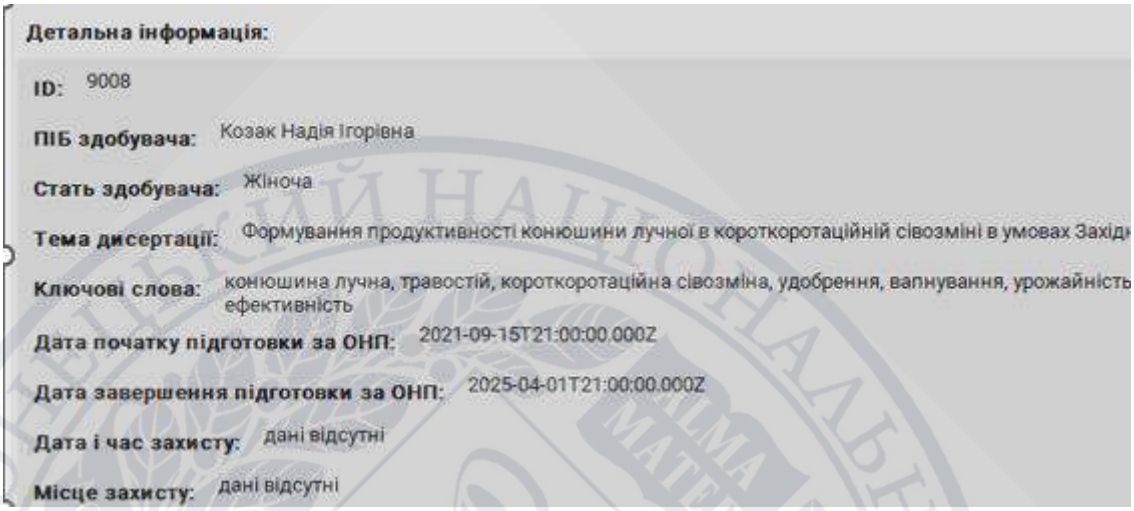


Рисунок 3.3 – Панель "Детальна інформація", що відображає атрибути обраного захисту

3.2 Інструкція з використання розробленого додатку

1. Загальний опис інструменту

Розроблений додаток є десктопною програмою, яка дозволяє зручно працювати з даними про захисти дисертацій, збереженими у форматі CSV-файлу. Інтерфейс програми складається з панелі керування (ліворуч), де розташовані елементи для завантаження даних, керування користувачем та фільтрації, а також робочої області (праворуч), що містить таблицю для відображення даних та панель для перегляду детальної інформації про обраний запис.

2. Підготовка даних

Перед використанням додатку необхідно отримати файл з даними про захисти дисертацій у форматі CSV. Цей файл генерується за допомогою окремого скрипта парсингу (pa.py), вихідний код якого наведено у Додатку А.

Для генерації файлу даних:

1. Переконайтесь, що у вас встановлено Python та всі необхідні бібліотеки (selenium, BeautifulSoup, pandas, customtkinter, webdriver_manager - якщо використовується для автоматичного завантаження драйвера, json, ast, os, re, time, threading). Встановити їх можна за допомогою pip: pip install selenium beautifulsoup4 pandas customtkinter webdriver_manager.
2. Завантажте відповідний драйвер для вашого веб-браузера (наприклад, chromedriver для Google Chrome) та переконайтесь, що він доступний у системній змінній середовища PATH або вкажіть шлях до нього у скрипті pa.py. Альтернативно, можна використовувати webdriver_manager у скрипті для автоматичного завантаження драйвера.
3. Відкрийте файл pa.py та за потреби відкоригуйте параметри:
    - start_id: Початковий ID для парсингу.
    - end_id: Кінцевий ID для парсингу (парсинг відбувається у зворотному порядку від start_id до end_id).
    - delay_between_requests: Затримка в секундах між запитами до сторінок (рекомендовано залишати значення більше 1 для уникнення блокування).
    - navigation_timeout: Максимальний час очікування завантаження сторінки.
    - max_consecutive_errors: Кількість послідовних помилок, після яких парсинг зупиниться.
4. Запустіть скрипт pa.py з терміналу: python pa.py.
5. Скрипт почне процес парсингу, виводячи інформацію про прогрес у консоль. Після завершення парсингу (або зупинки через помилки), буде створено файл parsed_defenses_data.csv (або all_defenses_data_v12.csv залежно від версії скрипта) у тому ж каталозі, де знаходиться скрипт. Цей файл містить зібрані дані.

3. Використання додатку з графічним інтерфейсом

Вихідний код додатку з графічним інтерфейсом знаходиться у файлі desktop_app_v6.py.

1. Переконайтесь, що у вас встановлено Python та всі необхідні бібліотеки (ті самі, що і для парсера, плюс customtkinter).
2. Запустіть файл desktop_app_v6.py з терміналу: python desktop_app_v6.py.
3. Відкриється головне вікно додатку.

3.1. Завантаження даних

Для роботи з даними їх необхідно завантажити у програму.

1. На панелі керування (ліворуч) знайдіть блок "Керування файлом (локально)".
2. Натисніть кнопку "Завантажити CSV".
3. У вікні вибору файлу, що з'явиться, оберіть CSV-файл з даними, згенерований скриптом парсингу (наприклад, parsed_defenses_data.csv).
4. Після успішного завантаження, у таблиці (праворуч) з'являться дані з файлу. Під кнопкою "Завантажити CSV" з'явиться напис, що вказує назву завантаженого файлу та кількість записів.

(Примітка: Додаток також може автоматично намагатися завантажити файл parsed_defenses_data.csv або all_defenses_data_v12.csv при запуску, якщо він знаходиться у тому ж каталозі.)

3.2. Перегляд та фільтрація даних

Після завантаження даних, їх можна переглядати у таблиці та фільтрувати.

1. **Табличне представлення:** У робочій області (праворуч) відображається таблиця з усіма завантаженими даними. Кожен стовпець відповідає полю даних (ID, ПІБ, Тема тощо), а кожен рядок – окремому запису про захист. Таблицю можна прокручувати для перегляду всіх даних.

2. **Фільтрація:** На панелі керування (ліворуч) знаходиться блок "Фільтри для перегляду". Тут розташовані поля для введення критеріїв фільтрації за різними стовпцями (ПІБ здобувача, Тема дисертації, Ключові слова, Заклад освіти, Рік захисту) та вибору спеціальності зі спадного списку.

- Введіть текст у відповідне поле фільтра (наприклад, частину ПІБ або теми дисертації). Фільтрація за текстовими полями здійснюється за входженням підрядка (без урахування регістру).
- Оберіть спеціальність зі спадного списку "Фільтр за спеціальністю". Список спеціальностей формується автоматично на основі даних у завантаженому файлі.
- Натисніть кнопку "Застосувати фільтри". Таблиця буде оновлена, відображаючи лише ті записи, які відповідають усім заданим критеріям фільтрації. Кількість відфільтрованих записів буде відображена поруч з таблицею або у відповідному написі.
- Натисніть кнопку "Скинути фільтри", щоб очистити всі поля фільтрації та відобразити всі завантажені дані.

3.3. Перегляд детальної інформації

Для перегляду детальної інформації про конкретний захист:

1. Оберіть рядок у таблиці, клікнувши на нього лівою кнопкою миші.
2. На панелі "Детальна інформація" (під таблицею) з'явиться розгорнута інформація про вибраний захист, включаючи всі поля даних, у тому числі вміст вкладених структур ("Публікації" та "Разова рада").
3. Якщо у деталях є посилання (наприклад, на публікацію або профіль ORCID), вони будуть відображені як кнопки "Відкрити", при натисканні на які посилання відкриється у веб-браузері за замовчуванням.

3.4. Візуалізація даних

Додаток надає базові можливості візуалізації даних у вигляді графіків та діаграм на основі поточних відфільтрованих даних.

1. Застосуйте необхідні фільтри, щоб отримати набір даних для візуалізації.

2. (Якщо реалізовано у вашій версії додатку) Знайдіть елементи керування для побудови візуалізацій (наприклад, кнопки або спадні списки для вибору типу візуалізації та стовпців даних).

3. Оберіть тип візуалізації (наприклад, стовпчаста діаграма, кругова діаграма) та стовпець даних, який потрібно візуалізувати (наприклад, "Рік захисту" для стовпчастої діаграми кількості захистів за роками, або "Статус роботи" для кругової діаграми розподілу статусів).

4. Програма згенерує та відобразить відповідний графік або діаграму на основі поточного відфільтрованого набору даних.

4. Мій кабінет (збереження налаштувань)

Блок "Мій кабінет" дозволяє зберігати деякі налаштування користувача, зокрема, відстежувані спеціальності та останній відомий ID захисту.

1. Введіть ваш логін у поле "Ваш Логін". Це може бути будь-який ідентифікатор, який ви хочете використовувати для збереження своїх налаштувань.

2. Натисніть кнопку "Зберегти / Увійти". Ваші налаштування (відстежувані спеціальності та останній відомий ID) будуть збережені у файлі, пов'язаному з цим логіном. При наступному запуску програми та введенні цього логіна, налаштування будуть завантажені.

3. Використовуйте спадний список "Відстежувати спеціальності" та кнопку "Додати для відстеження", щоб обрати спеціальності, за якими ви хочете отримувати сповіщення про нові захисти (функціонал сповіщень реалізовано у фоновому режимі при запуску парсингу або ручній перевірці).

4. Список "Відстежувані спеціальності" відображає спеціальності, які ви зараз відстежуєте. Кнопка "X" поруч зі спеціальністю дозволяє видалити її зі списку відстеження.

5. Кнопка "Перевірити нові (без оновлення старих)" дозволяє здійснити швидку перевірку наявності нових захистів за вашими відстежуваними спеціальностями без повного парсингу всього діапазону ID.

6. Кнопка "Показати всі завантажені дані" дозволяє повернутися до відображення всіх даних у таблиці після застосування фільтра "тільки нові" (якщо така функція була активована після перевірки нових захистів).

Висновки до Розділу 3.

У четвертому розділі представлено розроблений графічний додаток, який слугує інтерактивним інструментом для роботи зі зібраними даними про захисти дисертацій. Описано інтерфейс користувача головного вікна, що включає панель керування з секціями для парсингу даних, керування файлами та фільтрації, а також основну робочу область для відображення даних у табличному вигляді та детальної інформації про обраний запис. Детально розглянуто блок "Фільтри для перегляду", що надає гнучкі можливості для відбору та відображення даних за різними критеріями, такими як спеціальність, ПІБ здобувача, тема дисертації, ключові слова, заклад освіти та рік захисту. Описано панель "Детальна інформація", яка дозволяє користувачу переглядати повний набір атрибутів для кожного запису, включаючи вкладені структури, та відкривати пов'язані посилання. Надано детальну інструкцію з використання додатку, що охоплює підготовку даних, завантаження CSV-файлів, перегляд та фільтрацію, а також функціонал "Мій кабінет" для збереження налаштувань та відстеження нових захистів за обраними спеціальностями. Таким чином, розроблений додаток забезпечує зручний та ефективний інтерфейс для взаємодії зі зібраними даними, сприяючи їх подальшому аналізу та використанню.

ВИСНОВКИ

У бакалаврській роботі було успішно досліджено та реалізовано процес автоматизованого збору структурованих даних про захисти PhD-дисертацій з веб-сайту Національного агентства забезпечення якості освіти (НАЗЯВО). Досягнення поставленої мети та послідовне виконання визначених завдань дозволило отримати наступні ключові результати та зробити обґрунтовані висновки:

1. Проведено комплексний теоретичний аналіз поняття веб-парсингу, його значення в сучасному інформаційному просторі, а також огляд та порівняльний аналіз сучасних технологій і програмних засобів для автоматизованого збору даних з веб-сайтів. Це дозволило сформувати міцну теоретичну базу та обґрунтувати вибір технологічного стеку для практичної реалізації системи інформування.

2. Здійснено детальний аналіз веб-сайту НАЗЯВО як джерела даних для системи інформування, досліджено структуру веб-сторінок, формат представлення інформації про захисти дисертацій та оцінено технічні можливості та обмеження для автоматизованого збору даних. Виявлено, що динамічний характер сайту вимагає використання інструментів автоматизації браузера.

3. Розроблено та реалізовано програмний інструмент (скрипт парсингу) на мові Python з використанням бібліотек Selenium, BeautifulSoup та Pandas. Інструмент враховує особливості сайту, включаючи динамічне завантаження контенту, та реалізує алгоритм послідовного збору даних за унікальними ID захистів для наповнення бази даних системи інформування.

4. Проведено всебічне тестування розробленого інструменту, включаючи перевірку коректності парсингу даних, обробку помилок та стійкість до неочікуваних ситуацій. Здійснено збір основного масиву даних про [зазначити кількість] захистів дисертацій, які були збережені у форматі CSV-файлу, що слугує основою для системи інформування.

5. Виявлено та проаналізовано проблеми, що виникли під час розробки та збору даних (неповне завантаження сторінок, зміни у структурі, помилки завантаження, ризик блокування), та запропоновано або реалізовано шляхи їх вирішення (використання явних очікувань, обробка винятків, затримки між запитами), забезпечуючи надійність функціонування системи.

6. Виконано узагальнення та первинну обробку зібраних даних, включаючи перевірку коректності за допомогою Excel та перетворення форматів. Зібраний набір даних є цінним, структурованим ресурсом, готовим для аналізу в рамках системи інформування.

7. Розроблено власний додаток з графічним інтерфейсом для зручної взаємодії зі зібраними даними. Відпарсений матеріал зберігається у структурованому CSV-форматі, що забезпечує його легке завантаження в розроблений додаток, а також надає можливість подальшого аналізу та використання в інших програмних засобах (наприклад, MS Excel, Python Pandas). Сам додаток виступає ключовою інтерактивною вихідною формою, дозволяючи користувачам не лише завантажувати та переглядати дані у табличному вигляді, але й застосовувати гнучкі фільтри за різними критеріями (ПІБ, тема, ключові слова, заклад освіти, рік захисту, спеціальність), переглядати детальну інформацію по кожному запису. Крім того, додаток включає функціонал "Мій кабінет", який дозволяє користувачам зберігати індивідуальні налаштування (логін) та реалізовану систему сповіщення на пошту про появу нових дисертацій за обраними спеціальностями шляхом перевірки оновлень відносно останнього відомого ID, що є важливим кроком до реалізації повноцінної системи інформування користувачів.

8. Проведено аналіз отриманих результатів, представлених за допомогою візуалізацій у власному додатку, та визначено потенційні напрямки наукового використання зібраної інформації для досліджень у галузі освіти та науки, включаючи аналіз тенденцій, моніторинг кадрового потенціалу, вивчення публікаційної активності та дослідження діяльності наукових установ, що підвищує цінність створеної системи інформування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ефективний парсинг: Як правильно обробляти і аналізувати дані товарів [Електронний ресурс] // Блог - 24magaz.in.ua. URL: https://web.24magaz.in.ua/blog/efektyvnyj-parsyng-yak-pravylno-obroblyaty-i-analizuvaty-dani-tovariv/ (дата звернення: 10.04.2025).
2. Jackson B. XML vs HTML: What's the Difference? [Електронний ресурс] // KeyCDN Support. 29.03.2023. URL: https://www.keycdn.com/support/xml-vs-html (дата звернення: 10.04.2025).
3. Pandas DataFrames [Електронний ресурс] // W3Schools. URL: https://www.w3schools.com/python/pandas/pandas_dataframes.asp (дата звернення: 10.04.2025).
4. Web - HTML, CSS, & JAVASCRIPT. - HTML [Електронний ресурс] // OneCompiler. URL: https://onecompiler.com/html/3x2zp2yd7 (дата звернення: 10.04.2025).
5. BeautifulSoup and lxml.html - what to prefer? [Електронний ресурс] : форум // Stack Overflow. 11.02.2011. URL: https://stackoverflow.com/questions/4967103/beautifulsoup-and-lxml-html-what-to-prefer (дата звернення: 10.04.2025).
6. Mitchell R. Web Scraping with Python: Collecting Data from the Modern Web. — 2nd ed. — O'Reilly Media, 2018. — 320 p.
7. Lawson R. Web Scraping with Python and BeautifulSoup. — Packt Publishing, 2020. — 310 p.
8. Харченко В. С. Програмні засоби Інтернету. — Харків: ХНУРЕ, 2021. — 198 с.
9. Литвин В. В. Основи інтелектуального аналізу даних. — Львів: Львівська політехніка, 2019. — 264 с.
10. Janert P. K. Data Analysis with Open Source Tools. — O'Reilly Media, 2010. — 540 p.
11. Kuhn M., Johnson K. Applied Predictive Modeling. — Springer, 2013. — 600 p.
12. Official Selenium Documentation. URL: https://www.selenium.dev/documentation/ (дата звернення: 11.04.2025).
13. BeautifulSoup Documentation. URL: https://www.crummy.com/software/BeautifulSoup/ (дата звернення: 11.04.2025).
14. Python requests library documentation. URL: https://requests.readthedocs.io/en/latest/ (дата звернення: 11.04.2025).

15. Python pandas documentation. URL: https://pandas.pydata.org/docs/ (дата звернення: 13.04.2025).
16. Playwright official site. URL: https://playwright.dev/python/ (дата звернення: 13.04.2025).
17. Automated Web Scraping Techniques in Academic Research. // Journal of Data Science. — 2021. — Vol. 19, No. 4. — P. 113–130.
18. Python Software Foundation. Python Language Reference, version 3.x. URL: https://www.python.org/ (дата звернення: 13.04.2025).
19. DataCamp. Web Scraping in Python. URL: https://www.datacamp.com/tutorial/web-scraping-using-python (дата звернення: 13.04.2025).
20. Open Data Handbook. URL: https://opendatahandbook.org/ (дата звернення: 14.04.2025).
21. Scrapy Official Docs. URL: https://docs.scrapy.org/en/latest/ (дата звернення: 14.04.2025).
22. Web Scraping in the Era of GDPR. // International Journal of Legal Technology. — 2020. — Vol. 12, No. 2. — P. 88–95.
23. Selenium WebDriver Documentation: Waits [Електронний ресурс]. URL: https://www.selenium.dev/documentation/webdriver/waits/ (дата звернення: 14.04.2025).

ДОДАТКИ

ДОДАТОК А

Код скрипта парсингу

```
driver = webdriver.Chrome()

base_url = "https://svr.naqa.gov.ua/#/defense/"
start_id = 9028
end_id = 1
delay_between_requests = 1
navigation_timeout = 30

all_defenses_data = []

def parse_defense_data(soup, defense_id_str, current_url):
    data = {
        'ID': defense_id_str
    }
    try:
        def get_value_by_label(label_text):
            label_td = soup.find('td', string=lambda text: text
and label_text.strip() in text.strip())
            if label_td and label_td.find_next_sibling('td'):
                value_div =
label_td.find_next_sibling('td').find('div')
                return value_div.get_text(strip=True) if value_div
else "Не знайдено (div)"
            return f"Не знайдено ({label_text})"

        data['ПІБ здобувача'] = get_value_by_label('ПІБ
здобувача')
        data['Стать здобувача'] = get_value_by_label('Стать
здобувача')
        data['Тема дисертації'] = get_value_by_label('Тема
дисертації')
        data['Ключові слова'] = get_value_by_label('Ключові
слова')
        data['Дата початку'] = get_value_by_label('Дата початку ')
        data['Дата завершення'] = get_value_by_label('Дата
завершення')
        data['Дата і час захисту'] = get_value_by_label('Дата і
час захисту')
        data['Місце захисту'] = get_value_by_label('Місце
захисту')
        data['Заклад освіти, освітня програма'] =
get_value_by_label('Заклад освіти, освітня програма')
```

```
        data['Відомості про акредитацію освітньої програми (за
даними ЄДЕБО)'] = get_value_by_label('Відомості про акредитацію
освітньої програми (за даними ЄДЕБО)')

        app_status_chip_tag = soup.find('app-defense-state-chip')
        if app_status_chip_tag:
            status_chip = app_status_chip_tag.find('mat-chip')
            if status_chip:
                data['Статус роботи'] =
status_chip.get_text(strip=True)
            else:
                data['Статус роботи'] = "Не знайдено (mat-chip в
app-defense-state-chip)"
        else:
            data['Статус роботи'] = "Не знайдено (app-defense-
state-chip)"

        publications_list = []
        publications_header = soup.find('h4', string=lambda text:
text and 'Публікації здобувача за темою дисертації' in
text.strip())
        if publications_header:
            app_publications_display =
publications_header.find_next_sibling('app-publications-display')
            if app_publications_display:
                publications_table =
app_publications_display.find('table', class_='mat-table')
                if publications_table:
                    table_body = publications_table.find('tbody')
                    if table_body:
                        rows = table_body.find_all('tr',
class_='mat-row')
                        for row in rows:
                            publication_data = {}
                            if type_cell := row.find('td',
class_=lambda x: x and 'cdk-column-type' in x.split()):
publication_data['Тип'] = type_cell.get_text(strip=True)
                            if description_cell := row.find('td',
class_=lambda x: x and 'cdk-column-description' in x.split()):
publication_data['Опис'] = description_cell.get_text(strip=True)
                            if year_cell := row.find('td',
class_=lambda x: x and 'cdk-column-year' in x.split()):
publication_data['Рік'] = year_cell.get_text(strip=True)
                            if keywords_cell := row.find('td',
class_=lambda x: x and 'cdk-column-keywords' in x.split()):
```

```
publication_data['Ключові слова'] =
keywords_cell.get_text(strip=True)
                            if doi_cell := row.find('td',
class_=lambda x: x and 'cdk-column-doi' in x.split()):
publication_data['DOI'] = doi_cell.get_text(strip=True)
                            if actions_cell := row.find('td',
class_=lambda x: x and 'cdk-column-actions' in x.split()):
                                if link_tag :=
actions_cell.find('a', mattooltip='Перейти до публікації'):
                                    if link_tag.has_attr('href'):
publication_data['Посилання'] = link_tag['href']
                            if icons_cell := row.find('td',
class_=lambda x: x and 'cdk-column-icons' in x.split()):
                                publication_data['Співавторство']
= 'Так' if icons_cell.find('mat-icon', mattooltip='У
співавторстві') else 'Ні'
                            if publication_data:
publications_list.append(publication_data)
                    data['Публікації'] = publications_list
                else: data['Публікації'] = "Не знайдено (tbody
в таблиці публікацій)"
            else: data['Публікації'] = "Не знайдено (таблиця в
app-publications-display)"
        else: data['Публікації'] = "Не знайдено (app-
publications-display)"
    else: data['Публікації'] = "Заголовок 'Публікації' не
знайдено"

    council_members = []
    council_header = soup.find('h3', string=lambda text: text
and 'Разова рада' in text.strip())
    if council_header:
        app_council_members_tag =
council_header.find_next_sibling('app-svr-members-show')
        if app_council_members_tag:
            found_council_table =
app_council_members_tag.find('table', class_='mat-table')
            if found_council_table:
                table_body = found_council_table.find('tbody')
                if table_body:
                    rows = table_body.find_all('tr',
class_='mat-row')
                    if not rows: rows =
table_body.find_all('tr')
                    for row_idx, row in enumerate(rows):
```

```
                            if row.find('th'): continue
                            member_data = {}
                            if name_cell := row.find('td',
class_=lambda x: x and 'cdk-column-name' in x.split()):
                                member_data['ПІБ'] =
name_cell.get_text(strip=True)
                            if type_cell := row.find('td',
class_=lambda x: x and 'cdk-column-svrMemberType' in x.split()):
member_data['Тип'] = type_cell.get_text(strip=True)
                            if hei_details_cell :=
row.find('td', class_=lambda x: x and 'cdk-column-heiName' in
x.split()): member_data['Установа та посада'] =
hei_details_cell.get_text(strip=True)
                            if degree_cell := row.find('td',
class_=lambda x: x and 'cdk-column-degree' in x.split()):
member_data['Ступінь, спеціальність'] =
degree_cell.get_text(strip=True)
                            if actions_cell := row.find('td',
class_=lambda x: x and 'cdk-column-actions' in x.split()):
                                if orcid_link_tag :=
actions_cell.find('a', mattooltip='Профіль ORCID'):
                                    if
orcid_link_tag.has_attr('href'): member_data['ORCID'] =
orcid_link_tag['href']

council_members.append(member_data)
                    data['Разова рада'] = council_members
                else: data['Разова рада'] = "Не знайдено
(tbody в таблиці ради)"
            else: data['Разова рада'] = "Не знайдено (таблиця
в app-svr-members-show)"
        else: data['Разова рада'] = "Не знайдено (app-svr-
members-show)"
    else: data['Разова рада'] = "Заголовок 'Разова рада' не
знайдено"

    except Exception as e:
        print(f"Помилка парсингу даних для ID {defense_id_str}:
{e}")

    data['Посилання на роботу'] = current_url
    return data

for i in range(start_id, end_id - 1, -1):
    current_id_str = str(i)
```

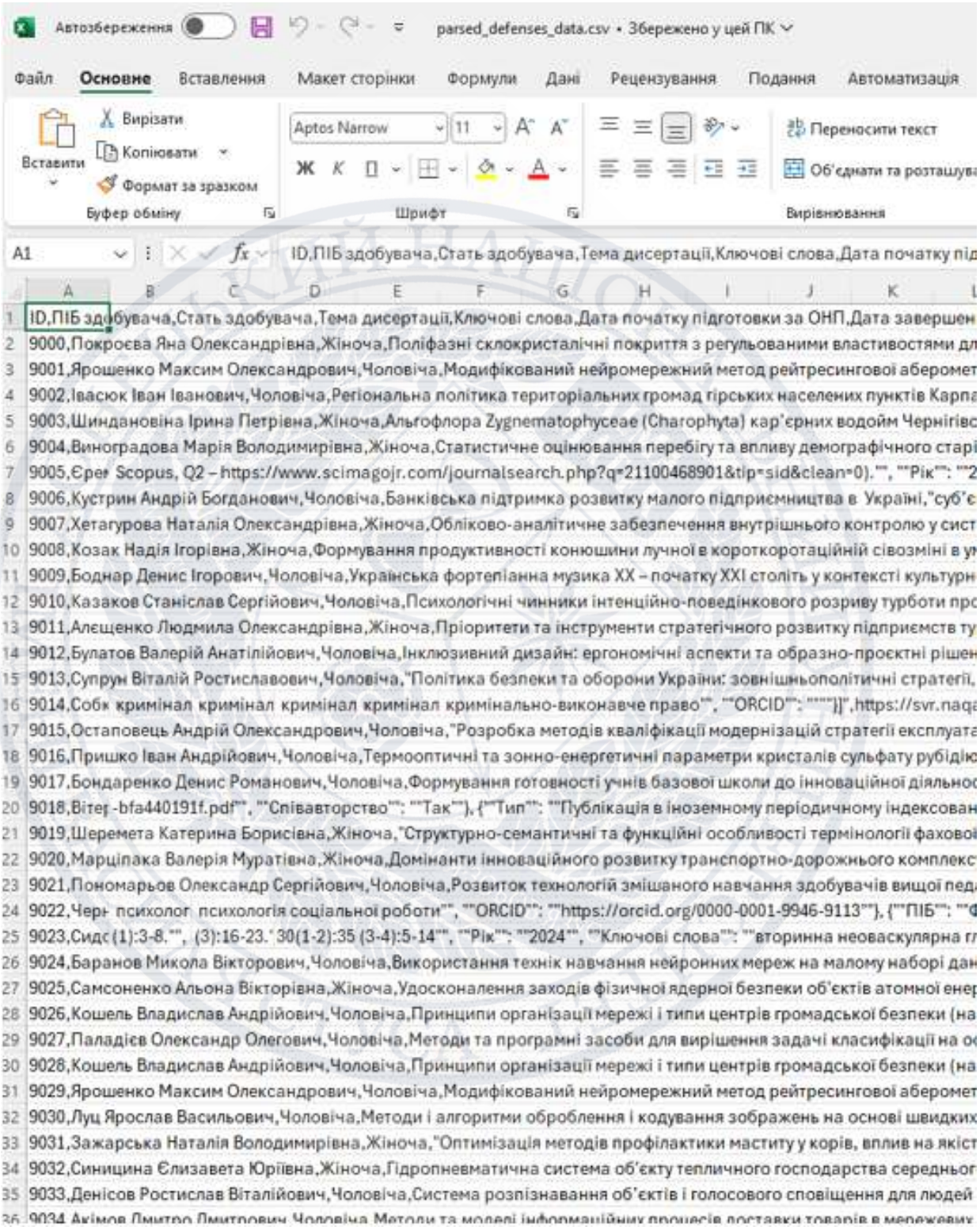
```
    url_to_parse = base_url + current_id_str
    parsed_data_for_id = {'ID': current_id_str, 'Посилання на
роботу': url_to_parse}

    print(f"Обробка: {url_to_parse}")
    try:
        driver.get(url_to_parse)
        print("Спроба оновити сторінку (driver.refresh())...")
        driver.refresh()

        expected_id_text_on_page = f"ID {current_id_str}"
        print(f"Очікуємо текст '{expected_id_text_on_page}' на
сторінці...")
        wait_condition_xpath = f"//*[normalize-
space(text())='{expected_id_text_on_page}']"

        try:
            WebDriverWait(driver, navigation_timeout).until(
                EC.presence_of_element_located((By.XPATH,
wait_condition_xpath))
            )
            print(f"Текст '{expected_id_text_on_page}' знайдено.")
        except Exception as wait_error:
            print(f"ПОМИЛКА ОЧІКУВАННЯ для ID {current_id_str}: Не
вдалося знайти текст '{expected_id_text_on_page}'. {wait_error}")
            all_defenses_data.append(parsed_data_for_id)
            print("-" * 30); time.sleep(delay_between_requests if
i != end_id else 0); continue

        time.sleep(1.5)
        page_html = driver.page_source
        soup = BeautifulSoup(page_html, 'html.parser')

        parsed_data_from_function = parse_defense_data(soup,
current_id_str, url_to_parse)
        parsed_data_for_id.update(parsed_data_from_function)
        all_defenses_data.append(parsed_data_for_id)

        pub_status = 'OK' if
isinstance(parsed_data_for_id.get('Публікації'), list) and
parsed_data_for_id.get('Публікації') else 'Error'
        rada_status = 'OK' if
isinstance(parsed_data_for_id.get('Разова рада'), list) and
parsed_data_for_id.get('Разова рада') else 'Error'
```

```
        print(f"ID: {parsed_data_for_id.get('ID')}, ПІБ:
{parsed_data_for_id.get('ПІБ здобувача', 'N/A')}, Статус:
{parsed_data_for_id.get('Статус роботи', 'N/A')}, Публікації:
{pub_status}, Рада: {rada_status}")

    except Exception as e:
        print(f"Загальна ПОМИЛКА для ID {current_id_str}: Не
вдалося завантажити або обробити сторінку. {e}")
        all_defenses_data.append(parsed_data_for_id)

    print("-" * 30)
    if i != end_id: print(f"Затримка {delay_between_requests}
сек..."); time.sleep(delay_between_requests)

driver.quit()
print("Парсинг завершено.")

if all_defenses_data:
    df = pd.DataFrame(all_defenses_data)

    expected_columns = [
        'ID',
        'ПІБ здобувача',
        'Стать здобувача',
        'Заклад освіти, освітня програма',
        'Відомості про акредитацію освітньої програми (за даними
ЄДЕБО)',
        'Тема дисертації',
        'Дата початку ',
        'Дата завершення ',
        'Дата і час захисту',
        'Місце захисту',
        'Статус роботи',
        'Публікації',
        'Разова рада',
        'Ключові слова',
        'Посилання на роботу'
    ]

    for col in expected_columns:
        if col not in df.columns:
            df[col] = None

    df = df[expected_columns]
```

```
    def format_list_of_dicts_for_csv(items_list,
section_name_for_order=None):
        if isinstance(items_list, str) and
items_list.startswith(("Не знайдено", "Заголовок")):
            return items_list
        if not isinstance(items_list, list) or not items_list:
            return ''
        item_strings = []
        pub_order = ['Тип', 'Опис', 'Рік', 'Ключові слова', 'DOI',
'Посилання', 'Співавторство']
        rada_order = ['Тип', 'ПІБ', 'Установа та посада',
'Ступінь, спеціальність', 'ORCID']
        key_order = []
        if section_name_for_order == 'Публікації': key_order =
pub_order
        elif section_name_for_order == 'Разова рада': key_order =
rada_order
        for item_dict in items_list:
            if isinstance(item_dict, dict):
                details = []
                if key_order:
                    for k in key_order:
                        v = item_dict.get(k)
                        if v is not None and str(v).strip() != '':
details.append(f"{k}: {str(v).strip()}")
                else:
                    for k, v in item_dict.items():
                        if v is not None and str(v).strip() != '':
details.append(f"{k}: {str(v).strip()}")
                item_strings.append("; ".join(details))
            else: item_strings.append(str(item_dict))
        return "\n".join(item_strings) if item_strings else ''
    if 'Публікації' in df.columns:
        df['Публікації'] = df['Публікації'].apply(lambda x:
format_list_of_dicts_for_csv(x, 'Публікації'))
    if 'Разова рада' in df.columns:
        df['Разова рада'] = df['Разова рада'].apply(lambda x:
format_list_of_dicts_for_csv(x, 'Разова рада'))

    try:
        df.to_csv('all_defenses_data_v12.csv', index=False,
encoding='utf-8-sig') # Нова назва файлу
        print("Дані збережено у all_defenses_data_v12.csv")
    except Exception as e: print(f"Помилка збереження у CSV: {e}")
else: print("Немає даних для збереження.")
```

ДОДАТОК В

Зразки зібраних даних



16	9014	Собкович Олександр Віталійович	Чоловіча	Запобігання кримінальним правопорушенням, що вчин
17	9015	Остаповець Андрій Олександрович	Чоловіча	Розробка методів кваліфікації модернізацій стратегії ек
18	9016	Пришко Іван Андрійович	Чоловіча	Термооптичні та зонно-енергетичні параметри кристалів
19	9017	Бондаренко Денис Романович	Чоловіча	Формування готовності учнів базової школи до інновац
20	9018	Вітер Надія Григорівна	Жіноча	Агробіологічні та екологічні умови функціонування поле
21	9019	Шеремета Катерина Борисівна	Жіноча	Структурно-семантичні та функційні особливості терміно
22	9020	Марціпака Валерія Муратівна	Жіноча	Домінанти інноваційного розвитку транспортно-дорожн
23	9021	Пономарьов Олександр Сергійович	Чоловіча	Розвиток технологій змішаного навчання здобувачів ви
24	9022	Чернілевська-Ісайко Олена Вікторівна	Жіноча	Вплив каністерапії на психологічні ресурси саморегуля
25	9023	Сидорчук Уляна Петрівна	Жіноча	Оптимізація хірургічного лікування неоваскулярної глау
26	9024	Баранов Микола Вікторович	Чоловіча	Використання технік навчання нейронних мереж на мал
27	9025	Самсоненко Альона Вікторівна	Жіноча	Удосконалення заходів фізичної ядерної безпеки об'єкт
28	9026	Кошель Владислав Андрійович	Чоловіча	Принципи організації мережі і типи центрів громадської
29	9027	Паладієв Олександр Олегович	Чоловіча	Методи та програмні засоби для вирішення задачі клас
30	9028	Кошель Владислав Андрійович	Чоловіча	Принципи організації мережі і типи центрів громадської
31	9029	Ярошенко Максим Олександрович	Чоловіча	Модифікований нейромережний метод рейтресинговой а
32	9030	Луц Ярослав Васильович	Чоловіча	Методи і алгоритми оброблення і кодування зображень
33	9031	Зажарська Наталія Володимирівна	Жіноча	Оптимізація методів профілактики маститу у корів, вплив
34	9032	Синицина Єлизавета Юріївна	Жіноча	Гідропневматична система об'єкту тепличного господар
35	9033	Денісов Ростислав Віталійович	Чоловіча	Система розпізнавання об'єктів і голосового сповіщен
36	9034	Акімов Дмитро Дмитрович	Чоловіча	Методи та моделі інформаційних процесів доставки това
37	9035	Золотарьова Ольга Ігорівна	Жіноча	Адміністративно-правові інструменти запобігання та пр
38	9036	Сунь Ifан	Чоловіча	Scientific grounds to provide lifetime of regional passenge
39	9037	Скічко Ірина Анатоліївна	Жіноча	Розслідування колабораційної діяльності
40	9038	Ковбик Костянтин Михайлович	Чоловіча	Удосконалення технології підземного відпрацювання по

ДОДАТОК С

Код додатку

```
import customtkinter as ctk
from tkinter import ttk, filedialog, messagebox
import pandas as pd
import webbrowser
import os
import json
import re
import time
from threading import Thread
import ast

# --- Глобальні налаштування ---
USER_SETTINGS_FILE = "user_settings.json" # Зберігає останній
логін
USER_DATA_TEMPLATE = "user_data_{}.json" # Шаблон для даних
користувача (спеціальності, last_id)
PROCESSED_IDS_FILE_TEMPLATE = "processed_ids_{}.json"
PARSED_DATA_FILE = "parsed_defenses_data.csv"
DEFAULT_START_PARSE_ID = 9000

class DefenseDataViewerApp(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("Переглядач, Парсер та Відстежувач Дисертацій
(v18 - ReCheck v3-debug)") # Оновлено версію
        self.geometry("1650x900")
        self.minsize(1000, 750)

        ctk.set_appearance_mode("System")
        ctk.set_default_color_theme("blue")

        self.dataframe = None
        self.filtered_df = None
        self.tree_item_to_df_idx = {}

        self.current_user_login = None
        self.user_tracked_specialties = []
        self.processed_ids = set()
        self.last_known_parsed_id = DEFAULT_START_PARSE_ID - 1

        self.parser_running = False
        self.showing_only_new_parsed = False

        self.create_widgets()
        self.load_last_user_login()
        self.try_load_default_csv(parsed_data_priority=True)

        if self.current_user_login and self.dataframe is None:
```

```
            self.after(1500, self.start_initial_parsing_thread)

    # --- Методи для налаштувань користувача --- (Без змін)
    def get_user_prefix(self, user_identifier):
        if not user_identifier: return "guest"
        return re.sub(r'[^a-zA-Z0-9_.-]', '_', user_identifier)

    def load_last_user_login(self):
        last_login = None
        if os.path.exists(USER_SETTINGS_FILE):
            try:
                with open(USER_SETTINGS_FILE, 'r', encoding='utf-
8') as f:
                    settings = json.load(f)
                    last_login = settings.get("user_login")
            except Exception as e:
                print(f"Помилка завантаження основного файлу
налаштувань: {e}")

        if last_login:
                if hasattr(self, 'login_entry'):
                    self.login_entry.delete(0, "end")
                    self.login_entry.insert(0, last_login)
                self.load_specific_user_data(last_login)
        else:
                self.reset_current_user_data_and_gui()
        if hasattr(self, 'login_status_label'):
self.update_login_status_label()

    def load_specific_user_data(self, login):
        print(f"Завантаження даних для користувача: {login}")
        self.current_user_login = login
        self.user_tracked_specialties = []
        self.processed_ids = set()
        user_prefix = self.get_user_prefix(login)
        user_data_file = USER_DATA_TEMPLATE.format(user_prefix)
        saved_last_id = None
        if os.path.exists(user_data_file):
            try:
                with open(user_data_file, 'r', encoding='utf-8')
as f:
                    settings = json.load(f)
                    self.user_tracked_specialties =
settings.get("tracked_specialties", [])
                    saved_last_id =
settings.get("last_known_parsed_id")
            except Exception as e:
                print(f"Помилка завантаження файлу даних для
{login}: {e}")

        if saved_last_id is not None and isinstance(saved_last_id,
int) and saved_last_id >= 0:
```

```
            self.last_known_parsed_id = saved_last_id
        else:
            self.last_known_parsed_id = DEFAULT_START_PARSE_ID - 1

        self.load_processed_ids()
        if hasattr(self, 'tracked_specialties_frame'):
self.update_tracked_specialties_display()
        if hasattr(self, 'login_status_label'):
self.update_login_status_label()

        can_interact = self.dataframe is not None
        if hasattr(self, 'add_specialty_button'):
self.add_specialty_button.configure(state="normal" if can_interact
and self.current_user_login else "disabled")
        if hasattr(self, 'manual_check_button'):
self.manual_check_button.configure(state="normal" if can_interact
and self.current_user_login else "disabled")
        if hasattr(self, 'show_all_data_button'):
self.show_all_data_button.configure(state="normal" if
self.showing_only_new_parsed else "disabled")
        if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="normal")

        print(f"Для {login} завантажено:
last_known_parsed_id={self.last_known_parsed_id},
{len(self.user_tracked_specialties)} відстежень,
{len(self.processed_ids)} оброблених ID.")

    def save_user_settings(self):
        if not self.current_user_login:
            print("Неможливо зберегти налаштування: користувач не
увійшов.")
            return

        main_settings = {"user_login": self.current_user_login}
        try:
            with open(USER_SETTINGS_FILE, 'w', encoding='utf-8')
as f:
                json.dump(main_settings, f, ensure_ascii=False,
indent=4)
        except Exception as e:
            print(f"Помилка збереження основного файлу
налаштувань: {e}")

        user_prefix =
self.get_user_prefix(self.current_user_login)
        user_data_file = USER_DATA_TEMPLATE.format(user_prefix)
        user_settings = {
            "tracked_specialties": self.user_tracked_specialties,
            "last_known_parsed_id": self.last_known_parsed_id
        }
        try:
```

```
            with open(user_data_file, 'w', encoding='utf-8') as f:
                json.dump(user_settings, f, ensure_ascii=False,
indent=4)
            print(f"Налаштування для {self.current_user_login}
збережено. last_known_parsed_id: {self.last_known_parsed_id}")
        except Exception as e:
            print(f"Помилка збереження налаштувань для
{self.current_user_login}: {e}")

    def reset_current_user_data_and_gui(self):
        print("Скидання даних поточного користувача.")
        self.current_user_login = None
        self.user_tracked_specialties = []
        self.processed_ids = set()
        self.last_known_parsed_id = DEFAULT_START_PARSE_ID - 1

        if hasattr(self, 'login_entry'):
self.login_entry.delete(0, "end")
        if hasattr(self, 'login_status_label'):
self.update_login_status_label()
        if hasattr(self, 'tracked_specialties_frame'):
self.update_tracked_specialties_display()

        can_interact_with_data = self.dataframe is not None
        if hasattr(self, 'add_specialty_button'):
self.add_specialty_button.configure(state="disabled")
        if hasattr(self, 'manual_check_button'):
self.manual_check_button.configure(state="disabled")
        if hasattr(self, 'show_all_data_button'):
self.show_all_data_button.configure(state="disabled")
        if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="normal")
        if hasattr(self, 'specialty_combobox'):
            self.specialty_combobox.configure(values=["Спочатку
завантажте CSV"], state="disabled")
            self.specialty_combobox.set("Оберіть спеціальність")
        if hasattr(self, 'filter_specialty_combobox'):
            self.filter_specialty_combobox.configure(values=["Всі
спеціальності"], state="disabled" if not can_interact_with_data
else "readonly")
            self.filter_specialty_combobox.set("Всі
спеціальності")

    def load_processed_ids(self):
        self.processed_ids = set()
        if not self.current_user_login: return
        user_prefix =
self.get_user_prefix(self.current_user_login)
        processed_file =
PROCESSED_IDS_FILE_TEMPLATE.format(user_prefix)
        if os.path.exists(processed_file):
            try:
```

```
                with open(processed_file, 'r', encoding='utf-8')
as f:
                    loaded_ids = json.load(f)
                    self.processed_ids = set(str(pid) for pid in
loaded_ids if pid is not None)
            except json.JSONDecodeError:
                print(f"Помилка декодування JSON у файлі
оброблених ID: {processed_file}")
            except Exception as e:
                print(f"Помилка завантаження оброблених ID для
{self.current_user_login}: {e}")
        print(f"Завантажено {len(self.processed_ids)} оброблених
ID для {self.current_user_login}")

    def save_processed_ids(self, ids_to_save):
        if not self.current_user_login: return
        ids_to_add = set(str(pid) for pid in ids_to_save if pid is
not None)
        if not ids_to_add.issubset(self.processed_ids):
            self.processed_ids.update(ids_to_add)
            user_prefix =
self.get_user_prefix(self.current_user_login)
            processed_file =
PROCESSED_IDS_FILE_TEMPLATE.format(user_prefix)
            try:
                with open(processed_file, 'w', encoding='utf-8')
as f:
                    json.dump(sorted(list(self.processed_ids)), f,
ensure_ascii=False, indent=4)
                print(f"Оновлено та збережено
{len(self.processed_ids)} оброблених ID.")
            except Exception as e: print(f"Помилка збереження
оброблених ID: {e}")

    def extract_specialty_code_and_name(self, text):
        if not isinstance(text, str): return None
        match = re.search(r'(?:^|\D|[-
(])(\d{2,3}(?:\.\d{1,2})?)\s+([A-ЯІЇЄҐ][^()]+?)(?:$|\s?\()', text)
        if match:
            code = match.group(1).strip()
            name = match.group(2).strip().rstrip(',').rstrip('.')
            if re.search(r'[a-zA-Za-яA-Я]', name):
                return f"{code} {name}"
            else: return None

        match_in_paren =
re.search(r'\((\d{2,3}(?:\.\d{1,2})?)\s+([^)]+)\)', text)
        if match_in_paren:
            code = match_in_paren.group(1).strip()
            name = match_in_paren.group(2).strip()
            if re.search(r'[a-zA-Za-яA-Я]', name):
                return f"{code} {name}"
```

```
        return None

    def show_notification(self, user_identifier, subject,
body_text):
        print(f"--- СПОВІЩЕННЯ (Для {user_identifier}) ---")
        print(f"Тема: {subject}")
        print(f"Тіло:\n{body_text}")
        print(f"-----------------------------------------")
        # messagebox.showinfo(subject, f"Сповіщення для
{user_identifier}:\n\n{body_text}") # Закоментовано
        return True

    def create_widgets(self):
        self.grid_columnconfigure(1, weight=1);
self.grid_rowconfigure(0, weight=1)
        self.controls_outer_panel = ctk.CTkFrame(self, width=380,
corner_radius=0, fg_color="transparent")
        self.controls_outer_panel.grid(row=0, column=0, padx=(10,
5), pady=10, sticky="ns")
        self.controls_panel =
ctk.CTkScrollableFrame(self.controls_outer_panel, width=360,
corner_radius=5)
        self.controls_panel

        parser_frame = ctk.CTkFrame(self.controls_panel,
fg_color="transparent")
        ctk.CTkLabel(parser_frame, text="Парсинг даних з сайту:",
font=ctk.CTkFont(weight="bold")).pack(anchor="w")
        self.run_parser_button = ctk.CTkButton(parser_frame,
text="Запустити парсинг (Перевірка + Нові)",
command=self.start_parsing_thread)
        self.run_parser_button.pack(pady=5, fill="x")
        self.parser_status_label = ctk.CTkLabel(parser_frame,
text="Статус парсера: готовий", wraplength=320)
        self.parser_status_label.pack(pady=5, fill="x")

        file_management_frame = ctk.CTkFrame(self.controls_panel,
fg_color="transparent")
        file_management_frame.pack(pady=5, padx=10, fill="x")
        ctk.CTkLabel(file_management_frame, text="Керування файлом
(локально):", font=ctk.CTkFont(weight="bold")).pack(anchor="w")
        self.load_button = ctk.CTkButton(file_management_frame,
text="Завантажити CSV", command=self.load_csv)
        self.load_button.pack(pady=5, fill="x")
        self.loaded_file_label =
ctk.CTkLabel(file_management_frame, text="Файл не завантажено",
wraplength=320)
        self.loaded_file_label.pack(pady=5, fill="x")

        cabinet_frame = ctk.CTkFrame(self.controls_panel,
fg_color="transparent")
        cabinet_frame.pack(pady=5, padx=10, fill="x")
```

```
        ctk.CTkLabel(cabinet_frame, text="Мій кабінет:",
font=ctk.CTkFont(weight="bold")).pack(anchor="w")
        ctk.CTkLabel(cabinet_frame, text="Ваш Логін (для
збереження налаштувань):").pack(anchor="w", pady=(5,0))
        self.login_entry = ctk.CTkEntry(cabinet_frame,
placeholder_text="Введіть логін...")
        self.login_entry.pack(pady=(0,5), fill="x")
        self.login_button = ctk.CTkButton(cabinet_frame,
text="Зберегти / Увійти", command=lambda:
self.login_or_save_user()) # Виправлено з lambda
        self.login_button.pack(pady=5, fill="x")
        self.login_status_label = ctk.CTkLabel(cabinet_frame,
text="Ви не увійшли.", font=ctk.CTkFont(slant="italic"))
        self.login_status_label.pack(pady=(0,10))
        ctk.CTkLabel(cabinet_frame, text="Відстежувати
спеціальності:").pack(anchor="w")
        self.specialty_combobox = ctk.CTkComboBox(cabinet_frame,
values=["Спочатку завантажте CSV"], state="readonly",
command=None) # dropdown_max_height видалено
        self.specialty_combobox.set("Оберіть спеціальність")
        self.specialty_combobox.pack(pady=(0,5), fill="x")
        self.add_specialty_button = ctk.CTkButton(cabinet_frame,
text="Додати для відстеження",
command=self.add_specialty_to_track, state="disabled")
        self.add_specialty_button.pack(pady=5, fill="x")
        ctk.CTkLabel(cabinet_frame, text="Відстежувані
спеціальності:").pack(anchor="w", pady=(5,0))
        self.tracked_specialties_frame =
ctk.CTkScrollableFrame(cabinet_frame, height=80)
        self.tracked_specialties_frame.pack(pady=5, fill="x",
expand=False)
        self.manual_check_button = ctk.CTkButton(cabinet_frame,
text="Перевірити нові (без оновлення старих)", command=lambda:
self.check_for_new_defenses(is_auto_check=False))
        self.manual_check_button.pack(pady=(10,5), fill="x")
        self.show_all_data_button = ctk.CTkButton(cabinet_frame,
text="Показати всі завантажені дані",
command=self.show_all_parsed_data, state="disabled")
        self.show_all_data_button.pack(pady=(5,5), fill="x")

        filters_main_frame = ctk.CTkFrame(self.controls_panel,
fg_color="transparent")
        filters_main_frame.pack(pady=10, padx=10, fill="x",
expand=True)
        ctk.CTkLabel(filters_main_frame, text="Фільтри для
перегляду:", font=ctk.CTkFont(weight="bold")).pack(anchor="w",
pady=(10,5))
        ctk.CTkLabel(filters_main_frame, text="Фільтр за
спеціальністю:").pack(anchor="w", pady=(5,0))
        self.filter_specialty_combobox =
ctk.CTkComboBox(filters_main_frame, values=["Всі спеціальності"],
state="readonly") # dropdown_max_height видалено
```

```
        self.filter_specialty_combobox.set("Всі спеціальності")
        self.filter_specialty_combobox.pack(pady=(0,10), fill="x")
        filter_options = [
            ("ПІБ здобувача:", "pib_filter_entry", "Частина
ПІБ..."),
            ("Тема дисертації:", "topic_filter_entry", "Частина
теми..."),
            ("Ключові слова:", "keywords_filter_entry", "Ключове
слово..."),
            ("Заклад освіти:", "institution_filter_entry",
"Частина назви закладу..."),
            ("Рік захисту:", "year_filter_entry", "РРРР (з 'Дата і
час захисту')")
        ]
        for label_text, entry_attr, placeholder in filter_options:
            frame = ctk.CTkFrame(filters_main_frame,
fg_color="transparent"); frame.pack(fill="x", pady=2)
            ctk.CTkLabel(frame, text=label_text, width=120,
anchor="w").pack(side="top", anchor="w")
            entry = ctk.CTkEntry(frame,
placeholder_text=placeholder); entry.pack(fill="x")
            setattr(self, entry_attr, entry)
        self.apply_filters_button =
ctk.CTkButton(filters_main_frame, text="Застосувати фільтри",
command=self.apply_filters)
        self.apply_filters_button.pack(pady=(10,5), fill="x")
        self.reset_filters_button =
ctk.CTkButton(filters_main_frame, text="Скинути фільтри",
command=self.reset_filters, fg_color="gray")
        self.reset_filters_button.pack(pady=5, fill="x")

        self.workspace_frame = ctk.CTkFrame(self, corner_radius=0,
fg_color="transparent")
        self.workspace_frame.grid(row=0, column=1, sticky="nswe",
padx=(5,10), pady=10)
        self.workspace_frame.grid_rowconfigure(0, weight=2);
self.workspace_frame.grid_rowconfigure(1, weight=1)
        self.workspace_frame.grid_columnconfigure(0, weight=1)
        self.table_display_frame =
ctk.CTkFrame(self.workspace_frame)
        self.table_display_frame.grid(row=0, column=0,
sticky="nswe", pady=(0,5))
        self.table_display_frame.grid_rowconfigure(0, weight=1);
self.table_display_frame.grid_columnconfigure(0, weight=1)
        style = ttk.Style();
    # --- Методи для користувача та спеціальностей --- (Без змін)
    def login_or_save_user(self):
        identifier = self.login_entry.get().strip()
        if identifier:
            if self.current_user_login != identifier:
                if self.current_user_login:
                    self.save_user_settings()
```

```
                self.load_specific_user_data(identifier)
                main_settings = {"user_login":
self.current_user_login}
                try:
                    with open(USER_SETTINGS_FILE, 'w',
encoding='utf-8') as f:
                        json.dump(main_settings, f,
ensure_ascii=False, indent=4)
                except Exception as e:
                    print(f"Помилка збереження основного файлу
налаштувань: {e}")
            else:
                if hasattr(self, 'login_status_label'):
self.update_login_status_label()

            can_interact = self.dataframe is not None
            if hasattr(self, 'add_specialty_button'):
self.add_specialty_button.configure(state="normal" if can_interact
and self.current_user_login else "disabled")
            if hasattr(self, 'manual_check_button'):
self.manual_check_button.configure(state="normal" if can_interact
and self.current_user_login else "disabled")
            if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="normal")
            messagebox.showinfo("Успіх", f"Логін '{identifier}'
встановлено/збережено.")
        else:
            messagebox.showerror("Помилка", "Будь ласка, введіть
ваш Логін.")

    def update_login_status_label(self):
        if self.current_user_login:
            self.login_status_label.configure(text=f"Ви увійшли
як: {self.current_user_login}", text_color="green")
        else:
            self.login_status_label.configure(text="Ви не
увійшли.", text_color=("gray60", "gray40"))

    def populate_specialty_comboboxes(self):
        all_specialties_set = set()
        if self.dataframe is not None and 'Заклад освіти, освітня
програма' in self.dataframe.columns:
            for text in self.dataframe['Заклад освіти, освітня
програма'].dropna():
                specialty =
self.extract_specialty_code_and_name(text)
                if specialty: all_specialties_set.add(specialty)

        sorted_specialties_for_tracking = ["Оберіть
спеціальність"] + sorted(list(all_specialties_set))
        if hasattr(self, 'specialty_combobox'):
```

```
self.specialty_combobox.configure(values=sorted_specialties_for_tr
acking)
            self.specialty_combobox.set("Оберіть спеціальність")

        sorted_specialties_for_filter = ["Всі спеціальності"] +
sorted(list(all_specialties_set))
        if hasattr(self, 'filter_specialty_combobox'):

self.filter_specialty_combobox.configure(values=sorted_specialties
_for_filter, state="readonly" if self.dataframe is not None else
"disabled")
            self.filter_specialty_combobox.set("Всі
спеціальності")

        if hasattr(self, 'add_specialty_button'):
            if self.current_user_login and self.dataframe is not
None:

self.add_specialty_button.configure(state="normal")
            else:

self.add_specialty_button.configure(state="disabled")

    def add_specialty_to_track(self):
        if not self.current_user_login:
messagebox.showwarning("Увага", "Будь ласка, спочатку увійдіть.");
return
        selected_spec = self.specialty_combobox.get()
        if selected_spec and selected_spec != "Оберіть
спеціальність":
            if selected_spec not in self.user_tracked_specialties:

self.user_tracked_specialties.append(selected_spec);
self.save_user_settings()
                self.update_tracked_specialties_display();
messagebox.showinfo("Успіх", f"Спеціальність '{selected_spec}'
додано.")
            else: messagebox.showinfo("Інформація",
f"Спеціальність '{selected_spec}' вже відстежується.")
        else: messagebox.showwarning("Увага", "Будь ласка, оберіть
спеціальність.")

    def remove_tracked_specialty(self, specialty_to_remove):
        if specialty_to_remove in self.user_tracked_specialties:

self.user_tracked_specialties.remove(specialty_to_remove);
self.save_user_settings()
            self.update_tracked_specialties_display()

    def update_tracked_specialties_display(self):
```

```
        for widget in
self.tracked_specialties_frame.winfo_children(): widget.destroy()
        if not self.user_tracked_specialties:
            ctk.CTkLabel(self.tracked_specialties_frame,
text="Немає відстежуваних спеціальностей.").pack(padx=5, pady=5);
return
        for spec in self.user_tracked_specialties:
            frame = ctk.CTkFrame(self.tracked_specialties_frame,
fg_color="transparent"); frame.pack(fill="x")
            ctk.CTkLabel(frame, text=spec,
wraplength=280).pack(side="left", padx=5, pady=2)
            remove_btn = ctk.CTkButton(frame, text="X", width=25,
height=25, fg_color="red", command=lambda s=spec:
self.remove_tracked_specialty(s))
            remove_btn.pack(side="right", padx=5, pady=2)

    # --- Метод check_for_new_defenses ---
    def check_for_new_defenses(self, is_auto_check=False,
new_data_df=None):
        print("--- Запуск check_for_new_defenses (тільки нові ID)
---")
        df_to_check = new_data_df if new_data_df is not None else
self.dataframe
        if df_to_check is None or df_to_check.empty:
            if not is_auto_check:
messagebox.showinfo("Інформація", "Немає даних для перевірки.");
            return False, None
        if not self.current_user_login:
            if not is_auto_check: messagebox.showwarning("Увага",
"Будь ласка, увійдіть для перевірки.");
            return False, None
        if not self.user_tracked_specialties:
            if not is_auto_check:
messagebox.showinfo("Інформація", "Немає відстежуваних
спеціальностей.");
            return False, None

        new_defenses_notifications = {}; found_new_overall =
False; newly_processed_ids_for_this_check = set()
        newly_found_rows_data = []
        self.load_processed_ids() # Ensure we have the latest set
of processed IDs for this user

        print(f"Перевірка DataFrame розміром {df_to_check.shape}
на нові ID...")
        for index, row in df_to_check.iterrows():
            row_id_str = str(row.get('ID', ''))
            if not row_id_str or row_id_str in self.processed_ids:

            # Important: Save these newly found (and now
shown/notified) IDs as processed for the user
```

```
self.save_processed_ids(newly_processed_ids_for_this_check)

            # --- Block for displaying new data in the table ---
            if not is_auto_check and newly_found_df is not None
and not newly_found_df.empty:
                print("[DEBUG] Check_for_new: Спроба показати нові
дані в таблиці.")
                print(f"[DEBUG] Check_for_new: newly_found_df
shape: {newly_found_df.shape}")
                if hasattr(self,'tree'): print(f"[DEBUG]
Check_for_new: Treeview columns: {list(self.tree['columns'])}")
                print(f"[DEBUG] Check_for_new: Columns in
newly_found_df: {newly_found_df.columns.tolist()}")

                self.filtered_df =
newly_found_df.reset_index(drop=True) # Update filtered_df to
*only* these new items

                print(f"[DEBUG] Check_for_new: Виклик
populate_treeview з df shape: {self.filtered_df.shape}")
                self.populate_treeview(self.filtered_df) #
Populate treeview with only new items
                if hasattr(self,'tree'):
                    self.tree.update_idletasks() # Attempt to
force GUI refresh for the tree
                    print(f"[DEBUG] Check_for_new: Кількість
елементів у дереві після populate:
{len(self.tree.get_children())}")

                self.showing_only_new_parsed = True # Set flag
                if hasattr(self, 'show_all_data_button'):

self.show_all_data_button.configure(state="normal") # Enable
button to show all data again

                # Show a messagebox to confirm new items were
found and displayed
                messagebox.showinfo("Знайдено нові", f"Знайдено
{len(newly_found_df)} нових захистів за вашими спеціальностями.
Вони відображені в таблиці.")

            return True, newly_found_df # Return that new items
were found, and the df of these items
        elif not is_auto_check: # No new items found during a
manual check
            messagebox.showinfo("Немає нових", "Нових (ще не
оброблених вами) захистів за вашими спеціальностями не знайдено.")
            if self.showing_only_new_parsed: # If table was
previously showing only new, revert to all
                self.show_all_parsed_data()
        return False, None # No new items found
```

```
            self.table_display_frame =
    ctk.CTkFrame(self.workspace_frame)
            self.table_display_frame.grid(row=0, column=0,
    sticky="nswe", pady=(0,5))
            self.table_display_frame.grid_rowconfigure(0, weight=1);
    self.table_display_frame.grid_columnconfigure(0, weight=1)
            style = ttk.Style();
            try: style.theme_use("clam")
            except Exception: style.theme_use("default")
            style.configure("Treeview.Heading", font=('Arial', 10,
    'bold')); style.configure("Treeview", rowheight=25, font=('Arial',
    10))
            style.map("Treeview", background=[('selected',
    '#0078D7')], foreground=[('selected', 'white')])
            self.tree = ttk.Treeview(self.table_display_frame,
    show="headings", style="Treeview")
            self.vsb = ctk.CTkScrollbar(self.table_display_frame,
    orientation="vertical", command=self.tree.yview)
            self.hsb = ctk.CTkScrollbar(self.table_display_frame,
    orientation="horizontal", command=self.tree.xview)
            self.tree.configure(yscrollcommand=self.vsb.set,
    xscrollcommand=self.hsb.set)
            self.vsb.grid(row=0, column=1, sticky="ns");
    self.hsb.grid(row=1, column=0, sticky="ew"); self.tree.grid(row=0,
    column=0, sticky="nswe")
            self.tree.bind("<<TreeviewSelect>>", self.on_row_select)
            self.details_outer_frame =
    ctk.CTkFrame(self.workspace_frame)
            self.details_outer_frame.grid(row=1, column=0,
    sticky="nswe", pady=(5,0))
            self.details_outer_frame.grid_rowconfigure(1, weight=1);
    self.details_outer_frame.grid_columnconfigure(0, weight=1)
            ctk.CTkLabel(self.details_outer_frame, text="Детальна
    інформація:", font=ctk.CTkFont(weight="bold")).grid(row=0,
    column=0, sticky="nw", padx=10, pady=(5,2))
            self.details_frame =
    ctk.CTkScrollableFrame(self.details_outer_frame, corner_radius=5)
            self.details_frame.grid(row=1, column=0, sticky="nswe",
    padx=5, pady=(0,5))
            self.details_placeholder =
    ctk.CTkLabel(self.details_frame, text="Оберіть рядок у таблиці для
    перегляду деталей.", wraplength=400, justify="center")
            self.details_placeholder.pack(pady=20, padx=20,
    expand=True, fill="both")


    # --- Методи парсера Selenium ---
    def _parse_single_defense_page(self, soup,
expected_defense_id_str, current_url):
        page_id_text = ""
        try:
            id_element = soup.find(lambda tag: tag.name == 'div'
and tag.get_text(strip=True).startswith('ID '))
            if id_element:
                match = re.search(r'ID\s*(\d+)',
id_element.get_text())
                if match:
```

```
                    page_id_text = match.group(1)
        except Exception as e:
            print(f"Помилка пошуку ID на сторінці
{expected_defense_id_str}: {e}")

        if page_id_text != expected_defense_id_str:
            print(f"ПОМИЛКА ВІДПОВІДНОСТІ ID: Очікувався
{expected_defense_id_str}, знайдено на сторінці '{page_id_text}'.
Повернення часткових даних.")
            # Повертаємо словник з ID та помилкою, щоб основний
цикл міг це обробити
            return {'ID': expected_defense_id_str, 'Error': f'ID
mismatch: expected {expected_defense_id_str}, found
{page_id_text}'}

        data = {'ID': expected_defense_id_str}
        try:
            def get_value_by_label(label_text):
                label_td = soup.find('td', string=lambda text:
text and label_text.strip() in text.strip())
                if label_td and label_td.find_next_sibling('td'):
                    value_div =
label_td.find_next_sibling('td').find('div')
                    return value_div.get_text(strip=True) if
value_div and value_div.get_text(strip=True) else ""
                return ""
            data['ПІБ здобувача'] = get_value_by_label('ПІБ
здобувача')
            data['Стать здобувача'] = get_value_by_label('Стать
здобувача')
            data['Тема дисертації'] = get_value_by_label('Тема
дисертації')
            data['Ключові слова'] = get_value_by_label('Ключові
слова')
            data['Дата початку підготовки за ОНП'] =
get_value_by_label('Дата початку підготовки за ОНП')
            data['Дата завершення підготовки за ОНП'] =
get_value_by_label('Дата завершення підготовки за ОНП')
            data['Дата і час захисту'] = get_value_by_label('Дата
і час захисту')
            data['Місце захисту'] = get_value_by_label('Місце
захисту')
            data['Заклад освіти, освітня програма'] =
get_value_by_label('Заклад освіти, освітня програма')
            data['Відомості про акредитацію освітньої програми (за
даними ЄДЕБО)'] = get_value_by_label('Відомості про акредитацію
освітньої програми (за даними ЄДЕБО)')
            app_status_chip_tag = soup.find('app-defense-state-
chip')
            status_text = ""
            if app_status_chip_tag:
                status_chip = app_status_chip_tag.find('mat-chip')
```

```
                if status_chip: status_text =
status_chip.get_text(strip=True)
            data['Статус роботи'] = status_text

            publications_list = []
            pubs_header = soup.find('h4', string=lambda text: text
and 'Публікації здобувача' in text.strip())
            if pubs_header and (app_pubs_display :=
pubs_header.find_next_sibling('app-publications-display')):
                if (pubs_table := app_pubs_display.find('table',
class_='mat-table')) and (tbody := pubs_table.find('tbody')):
                    for row in tbody.find_all('tr', class_='mat-
row'):
                        pub_data = {
                            'Тип': (cell.get_text(strip=True) if
(cell := row.find('td', class_=lambda x: x and 'cdk-column-type'
in x.split())) else ""),
                            'Опис': (cell.get_text(strip=True) if
(cell := row.find('td', class_=lambda x: x and 'cdk-column-
description' in x.split())) else ""),
                            'Рік': (cell.get_text(strip=True) if
(cell := row.find('td', class_=lambda x: x and 'cdk-column-year'
in x.split())) else ""),
                            'Ключові слова':
(cell.get_text(strip=True) if (cell := row.find('td',
class_=lambda x: x and 'cdk-column-keywords' in x.split())) else
""),
                            'DOI': (cell.get_text(strip=True) if
(cell := row.find('td', class_=lambda x: x and 'cdk-column-doi' in
x.split())) else ""),
                            'Посилання': (link.get('href') if
(cell := row.find('td', class_=lambda x: x and 'cdk-column-
actions' in x.split())) and (link := cell.find('a',
mattooltip='Перейти до публікації')) else ""),
                            'Співавторство': ('Так' if (cell :=
row.find('td', class_=lambda x: x and 'cdk-column-icons' in
x.split())) and cell.find('mat-icon', mattooltip='У
співавторстві') else 'Ні')
                        }
                        if any(pub_data.values()):
publications_list.append(pub_data)
            data['Публікації'] = json.dumps(publications_list,
ensure_ascii=False) if publications_list else ""

            council_members = []
            council_header = soup.find('h3', string=lambda text:
text and 'Разова рада' in text.strip())
            if council_header and (app_council_tag :=
council_header.find_next_sibling('app-svr-members-show')):
                if (council_table := app_council_tag.find('table',
class_='mat-table')) and (tbody := council_table.find('tbody')):
```

```
                    for row in tbody.find_all('tr', class_='mat-
row'):
                        member_data = {
                            'ПІБ': (cell.get_text(strip=True) if
(cell := row.find('td', class_=lambda x: x and 'cdk-column-name'
in x.split())) else ""),
                            'Тип': (cell.get_text(strip=True) if
(cell := row.find('td', class_=lambda x: x and 'cdk-column-
svrMemberType' in x.split())) else ""),
                            'Установа та посада':
(cell.get_text(strip=True) if (cell := row.find('td',
class_=lambda x: x and 'cdk-column-heiName' in x.split())) else
""),
                            'Ступінь, спеціальність':
(cell.get_text(strip=True) if (cell := row.find('td',
class_=lambda x: x and 'cdk-column-degree' in x.split())) else
""),
                            'ORCID': (link.get('href') if (cell :=
row.find('td', class_=lambda x: x and 'cdk-column-actions' in
x.split())) and (link := cell.find('a', mattooltip='Профіль
ORCID')) else "")
                        }
                        if member_data.get('ПІБ'):
council_members.append(member_data)
            data['Разова рада'] = json.dumps(council_members,
ensure_ascii=False) if council_members else ""
        except Exception as e:
            print(f"Помилка парсингу деталей для ID
{expected_defense_id_str}: {e}")
            data['Error'] = f"Parsing error: {e}"

        data['Посилання на роботу'] = current_url
        return data

    def run_parser_logic(self, start_id_to_parse,
max_consecutive_errors=5):
        self.parser_status_label.configure(text="Парсер:
Запуск...", text_color="orange")
        self.update_idletasks()

        all_newly_parsed_data_list = []
        updated_records_data = []
        ids_to_recheck = []
        updated_ids_count = 0
        newly_parsed_ids_count = 0

        driver = None

        # --- Фаза 1: Перевірка існуючих записів ---
        self.parser_status_label.configure(text="Парсер: Перевірка
існуючих...", text_color="orange")
        print("Парсер: Фаза 1 - Перевірка існуючих записів...")
```

```
        existing_df = None
        if os.path.exists(PARSED_DATA_FILE):
            try:
                existing_df = pd.read_csv(PARSED_DATA_FILE,
encoding='utf-8-sig', dtype={'ID': str})
                for col in
existing_df.select_dtypes(include=['object']).columns:
                    existing_df[col] = existing_df[col].fillna('')
                print(f"Парсер: Завантажено {len(existing_df)}
існуючих записів.")

                status_col = 'Статус роботи'
                date_col = 'Дата і час захисту'
                place_col = 'Місце захисту'
                missing_placeholders = ['', 'н/д', 'не вказано',
'дані відсутні', 'дата не визначена', 'NaN']

                conditions = []
                if status_col in existing_df.columns:

conditions.append(existing_df[status_col].astype(str).str.contains
("планується", case=False, na=False))
                if date_col in existing_df.columns:

conditions.append(existing_df[date_col].astype(str).str.strip().is
in(missing_placeholders))
                if place_col in existing_df.columns:

conditions.append(existing_df[place_col].astype(str).str.strip().i
sin(missing_placeholders))

                if conditions:
                    combined_condition = conditions[0]
                    for condition in conditions[1:]:
                        combined_condition = combined_condition |
condition
                    df_to_recheck =
existing_df[combined_condition]
                    if 'ID' in df_to_recheck.columns:
                        ids_to_recheck =
df_to_recheck['ID'].tolist()
                        print(f"Парсер: Знайдено
{len(ids_to_recheck)} записів для перевірки.")
                    else:
                        print("Парсер: Колонка 'ID' не знайдена в
даних для перевірки.")
                        ids_to_recheck = []
                else:
                    print("Парсер: Не знайдено колонок для
визначення записів для перевірки.")
                    ids_to_recheck = []
            except Exception as e:
```

```
                print(f"Помилка завантаження/фільтрації існуючого
CSV для перевірки: {e}")
                ids_to_recheck = []
        else:
            print(f"Парсер: Файл {PARSED_DATA_FILE} не знайдено,
перевірку існуючих записів пропущено.")

        if ids_to_recheck:
            try:
                driver_service = ChromeService()
                chrome_options = webdriver.ChromeOptions()
                chrome_options.add_argument("--log-level=3")

chrome_options.add_experimental_option('excludeSwitches',
['enable-logging'])
                driver = webdriver.Chrome(service=driver_service,
options=chrome_options)
            except Exception as e:
                print(f"Помилка ініціалізації WebDriver для
перевірки: {e}")
                messagebox.showerror("Помилка парсера", f"Не
вдалося запустити WebDriver.\n: {e}")
                self.parser_status_label.configure(text="Парсер:
Помилка WebDriver", text_color="red")
                self.parser_running = False;
                if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="normal");
                return

            base_url = "https://svr.naqa.gov.ua/#/defense/"
            delay_between_requests = 1.5 # Затримка між запитами
для перевірки
            navigation_timeout = 15    # Тайм-аут для завантаження
сторінки

            for idx, recheck_id in enumerate(ids_to_recheck):
                if not self.parser_running: print("Парсер зупинено
користувачем під час перевірки."); break
                url_to_parse = base_url + recheck_id
                self.parser_status_label.configure(text=f"Парсер:
Перевірка {idx+1}/{len(ids_to_recheck)} (ID {recheck_id})...")
                print(f"Парсер: Перевірка {url_to_parse}")
                page_loaded_correctly = False
                try:
                    driver.get(url_to_parse)
                    # ДОДАНО: Оновлення сторінки для перевірки
існуючих
                    print(f"Парсер: Оновлення сторінки для
перевірки ID {recheck_id}...")
                    driver.refresh()
                    time.sleep(1) # Невелика пауза після оновлення
```

```
                    expected_id_xpath = f"//*[normalize-
space(text())='ID {recheck_id}']"
                    print(f"Парсер: Очікування елемента 'ID
{recheck_id}' після оновлення (перевірка)...")
                    WebDriverWait(driver,
navigation_timeout).until(
                        EC.presence_of_element_located((By.XPATH,
expected_id_xpath))
                    )
                    page_loaded_correctly = True
                    time.sleep(delay_between_requests) # Затримка
після успішного завантаження

                except TimeoutException:
                    print(f"Парсер: Тайм-аут очікування 'ID
{recheck_id}' під час перевірки. Пропуск.")
                    continue
                except Exception as e:
                    print(f"Парсер: Помилка під час перевірки ID
{recheck_id}: {e}")
                    continue

                if page_loaded_correctly:
                    try:
                        page_html = driver.page_source
                        soup = BeautifulSoup(page_html,
'html.parser')
                        parsed_data_dict =
self._parse_single_defense_page(soup, recheck_id, url_to_parse)

                        if 'Error' in parsed_data_dict or not
parsed_data_dict.get('ПІБ здобувача'):
                            print(f"Парсер: Не вдалося розпарсити
ключові дані або виникла помилка
('{parsed_data_dict.get('Error')}') для ID {recheck_id} під час
перевірки. Пропуск.")
                            continue

                        original_row_series =
existing_df[existing_df['ID'] == recheck_id].iloc[0]
                        has_changes = False
                        static_fields = {'ID', 'ПІБ здобувача',
'Стать здобувача', 'Посилання на роботу'}
                        fields_to_compare = [col for col in
parsed_data_dict if col in original_row_series and col not in
static_fields]

                        for field in fields_to_compare:
                            old_value =
str(original_row_series.get(field, '')).strip()
                            new_value =
str(parsed_data_dict.get(field, '')).strip()
```

```
                            is_old_missing = old_value in
missing_placeholders
                            is_new_missing = new_value in
missing_placeholders

                            if old_value != new_value and not
(is_old_missing and is_new_missing):
                                print(f"    > Зміна в ID
{recheck_id}, поле '{field}': '{old_value}' -> '{new_value}'")
                                has_changes = True

                        if has_changes:

updated_records_data.append(parsed_data_dict)
                            updated_ids_count += 1
                    except Exception as parse_err:
                        print(f"Помилка обробки даних для
перевірки ID {recheck_id}: {parse_err}")


            if updated_records_data and existing_df is not None:
                print(f"Парсер: Оновлення
{len(updated_records_data)} записів у DataFrame...")
                try:
                    update_df = pd.DataFrame(updated_records_data)
                    if 'ID' in update_df.columns:
                        update_df['ID'] =
update_df['ID'].astype(str)
                        update_df = update_df.set_index('ID')
                    else:
                        print("Помилка: 'ID' колонка відсутня в
оновлених даних.")
                        update_df = None

                    if update_df is not None:
                        if 'ID' in existing_df.columns and
existing_df.index.name != 'ID':
                            existing_df =
existing_df.set_index('ID')

                        if existing_df.index.name == 'ID':
                            existing_df.update(update_df)
                            existing_df =
existing_df.reset_index()
                            print("Парсер: DataFrame оновлено.")
                            existing_df.to_csv(PARSED_DATA_FILE,
index=False, encoding='utf-8-sig')
                            print(f"Парсер: Оновлені дані
збережено у {PARSED_DATA_FILE} після перевірки.")
                        else:
                            print("Помилка: Не вдалося встановити
'ID' як індекс для оновлення існуючого DataFrame.")
```

```
                    else:
                        print("Не вдалося створити DataFrame для
оновлення.")
                except Exception as update_save_err:
                    print(f"Помилка оновлення або збереження CSV
після перевірки: {update_save_err}")

        # --- Фаза 2: Пошук нових записів ---
        self.parser_status_label.configure(text="Парсер: Пошук
нових записів...", text_color="orange")
        print(f"\n: Фаза 2 - Пошук нових записів з ID
{start_id_to_parse}...")

        current_id = start_id_to_parse
        consecutive_errors = 0
        new_highest_id_found_this_run = self.last_known_parsed_id

        # driver_initialized_phase2 = False # Ця змінна більше не
потрібна, оскільки драйвер або вже є, або створюється
        if driver is None: # Якщо драйвер не був створений у Фазі
1 (бо ids_to_recheck був порожнім)
            try:
                driver_service = ChromeService()
                chrome_options = webdriver.ChromeOptions()
                chrome_options.add_argument("--log-level=3")

chrome_options.add_experimental_option('excludeSwitches',
['enable-logging'])
                driver = webdriver.Chrome(service=driver_service,
options=chrome_options)
                # driver_initialized_phase2 = True
            except Exception as e:
                print(f"Помилка ініціалізації WebDriver для нових
записів: {e}")
                messagebox.showerror("Помилка парсера", f"Не
вдалося запустити WebDriver.\n: {e}")
                self.parser_status_label.configure(text="Парсер:
Помилка WebDriver", text_color="red")
                self.parser_running = False;
                if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="normal");
                return

        base_url = "https://svr.naqa.gov.ua/#/defense/"
        delay_between_requests = 2 # Затримка між запитами для
нових ID
        navigation_timeout = 20    # Тайм-аут для завантаження
сторінки нових ID

        while True:
            if not self.parser_running: print("Парсер зупинено
користувачем під час пошуку нових."); break
```

```
            url_to_parse = base_url + str(current_id)
            self.parser_status_label.configure(text=f"Парсер:
Пошук нового ID {current_id}...")
            print(f"Парсер: Обробка {url_to_parse}")
            page_loaded_correctly = False
            parsed_data_dict = None
            current_exception = None # Для збереження винятку,
якщо він виник

            try:
                driver.get(url_to_parse)
                # --- ПОЧАТОК ЗМІН ---
                print(f"Парсер: Оновлення сторінки для ID
{current_id}...")
                driver.refresh()
                time.sleep(1) # Невелика пауза після оновлення,
щоб сторінка "заспокоїлась"

                expected_id_xpath = f"//*[normalize-
space(text())='ID {current_id}']"
                print(f"Парсер: Очікування елемента 'ID
{current_id}' після оновлення...")
                # --- КІНЕЦЬ ЗМІН ---
                WebDriverWait(driver, navigation_timeout).until(
                    EC.presence_of_element_located((By.XPATH,
expected_id_xpath))
                )
                page_loaded_correctly = True # Вважаємо, що
сторінка завантажена коректно, якщо ID знайдено
                time.sleep(0.5 + delay_between_requests) # Пауза
після успішного завантаження та очікування

                page_html = driver.page_source
                soup = BeautifulSoup(page_html, 'html.parser')
                parsed_data_dict =
self._parse_single_defense_page(soup, str(current_id),
url_to_parse)

                if 'Error' in parsed_data_dict or not
parsed_data_dict.get('ПІБ здобувача'):
                    print(f"Парсер: Не вдалося розпарсити ключові
дані або виникла помилка ('{parsed_data_dict.get('Error')}') для
нового ID {current_id}. Ймовірно, ID не існує або помилка сайту.")
                    consecutive_errors += 1
                    page_loaded_correctly = False
                elif parsed_data_dict.get('ID') !=
str(current_id):
                    print(f"Парсер: Невідповідність ID у
розпарсених даних! Запитано {current_id}, отримано
{parsed_data_dict.get('ID')}. Пропуск.")
                    consecutive_errors += 1
                    page_loaded_correctly = False
```

```
            except TimeoutException as e_timeout:
                current_exception = e_timeout
                print(f"Парсер: Тайм-аут очікування 'ID
{current_id}'. Зупинка пошуку нових.")
                consecutive_errors += max_consecutive_errors
            except Exception as e_general:
                current_exception = e_general
                print(f"Парсер: Загальна помилка ID {current_id}:
{e_general}"); consecutive_errors += 1

            # Обробка результату ітерації
            if page_loaded_correctly and parsed_data_dict and
'Error' not in parsed_data_dict:

all_newly_parsed_data_list.append(parsed_data_dict)
                newly_parsed_ids_count += 1
                print(f"Парсер: Успішно ID: {current_id}")
                new_highest_id_found_this_run = current_id
                consecutive_errors = 0
                current_id += 1
            else: # Якщо була помилка (Timeout, Error в dict,
немає ПІБ, невідповідність ID, або інша Exception)
                if consecutive_errors >= max_consecutive_errors:
                    print(f"Парсер: Досягнуто {consecutive_errors}
послідовних помилок/невдач. Зупинка.")
                    break
                else:
                    # Якщо помилка була не Timeout (який вже
оброблено для зупинки), а логічна помилка парсингу
                    # або інша Exception, яка не призвела до
негайної зупинки,
                    # то переходимо до наступного ID.
                    # Це запобігає зацикленню на одному ID, якщо
він постійно викликає помилку,
                    # але ще не досягнуто ліміту
`max_consecutive_errors`.
                    if not isinstance(current_exception,
TimeoutException):
                        current_id += 1
                    # Якщо ж це був Timeout, current_id не
збільшується, і цикл завершиться
                    # на наступній ітерації через
`consecutive_errors >= max_consecutive_errors`.
                    # Або якщо це була інша помилка, яка не
Timeout, але ми ще не досягли ліміту,
                    # то ми спробуємо наступний ID.
                    # continue тут не потрібен, бо логіка циклу
сама перейде до наступної ітерації.


        if driver: driver.quit()
```

```
        # --- Фаза 3: Фінальна обробка та збереження ---
        print("\n: Фаза 3 - Обробка результатів...")
        newly_parsed_df = pd.DataFrame(all_newly_parsed_data_list)
if all_newly_parsed_data_list else None

        existing_df_final = None
        if os.path.exists(PARSED_DATA_FILE):
             try:
                 existing_df_final = pd.read_csv(PARSED_DATA_FILE,
encoding='utf-8-sig', dtype={'ID': str})
                 for col in
existing_df_final.select_dtypes(include=['object']).columns:
                     existing_df_final[col] =
existing_df_final[col].fillna('')
                 print("Парсер: Перезавантажено дані для
фінального об'єднання.")
             except Exception as e:
                 print(f"Помилка перезавантаження CSV для
фінального об'єднання: {e}")
                 existing_df_final = None
        else:
            print("Парсер: Існуючий файл не знайдено для
фінального об'єднання.")


        final_df = None
        if existing_df_final is not None and newly_parsed_df is
not None and not newly_parsed_df.empty:
             print("Парсер: Об'єднання оновлених та нових
даних...")
             if 'ID' in newly_parsed_df.columns:
newly_parsed_df['ID'] = newly_parsed_df['ID'].astype(str)
             final_df = pd.concat([existing_df_final,
newly_parsed_df], ignore_index=True)
             final_df = final_df.drop_duplicates(subset=['ID'],
keep='last').reset_index(drop=True)
        elif newly_parsed_df is not None and not
newly_parsed_df.empty:
             print("Парсер: Використання тільки нових даних.")
             final_df = newly_parsed_df
        elif existing_df_final is not None:
             print("Парсер: Використання оновлених існуючих даних
(нових не знайдено).")
             final_df = existing_df_final
        else:
            print("Парсер: Не знайдено ані нових, ані оновлених
даних.")

        if final_df is not None and not final_df.empty:
            try:
```

```
                final_df['ID_num'] = pd.to_numeric(final_df['ID'],
errors='coerce')
                final_df = final_df.sort_values(by='ID_num',
na_position='first').drop(columns=['ID_num'])
                final_df.to_csv(PARSED_DATA_FILE, index=False,
encoding='utf-8-sig')
                print(f"Парсер: Фінальні дані збережено у
{PARSED_DATA_FILE}")
            except Exception as e:
                print(f"Помилка збереження фінального CSV: {e}")
                try:
                    final_df.to_csv(PARSED_DATA_FILE, index=False,
encoding='utf-8-sig')
                    print(f"Парсер: Фінальні дані збережено у
{PARSED_DATA_FILE} (без сортування через помилку).")
                except Exception as e2:
                     print(f"Повторна помилка збереження
фінального CSV: {e2}")
        else:
            print("Парсер: Немає даних для збереження у фінальний
CSV.")

        if newly_parsed_ids_count > 0 and
new_highest_id_found_this_run > self.last_known_parsed_id:
                self.last_known_parsed_id =
new_highest_id_found_this_run
                if self.current_user_login:
                    self.save_user_settings()

        self.after(100, lambda:
self.load_file_data(PARSED_DATA_FILE))
        summary_message = f"Парсер: Завершено. Оновлено:
{updated_ids_count}. Нових: {newly_parsed_ids_count} (до ID
{new_highest_id_found_this_run})."
        final_status_color = "green" if (updated_ids_count > 0 or
newly_parsed_ids_count > 0) else ("black", "white")
        self.parser_status_label.configure(text=summary_message,
text_color=final_status_color)
        if updated_ids_count == 0 and newly_parsed_ids_count == 0:
             messagebox.showinfo("Парсер", "Нових або оновлених
даних на сайті не знайдено.")

        self.parser_running = False;
        if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="normal")

    # --- Інші методи ---
    # (Методи від start_parsing_thread до кінця файлу залишаються
без змін від v18 - ReCheck v1)
    def start_parsing_thread(self, initial_check=False):
        if self.parser_running: messagebox.showinfo("Парсер",
"Парсер вже запущено."); return
```

```
        self.parser_running = True;
        if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="disabled");
        self.parser_status_label.configure(text="Парсер:
Запуск...")

        start_from_id_for_new = self.last_known_parsed_id + 1
        print(f"Запуск парсера з перевіркою існуючих та пошуком
нових з ID: {start_from_id_for_new}")
        parser_thread = Thread(target=self.run_parser_logic,
args=(start_from_id_for_new,))
        parser_thread.daemon = True; parser_thread.start()

    def start_initial_parsing_thread(self):
        if not self.current_user_login:
            print("Початковий парсинг пропущено: користувач не
увійшов."); return
        if self.dataframe is None:
            print("Запуск початкового парсингу (немає
завантажених даних)...")
            self.start_parsing_thread(initial_check=True)
        else:
            print("Локальні дані вже завантажено, початковий
парсинг не виконується автоматично. Запустіть вручну, якщо
потрібно.")

    def try_load_default_csv(self, parsed_data_priority=False):
        default_file_to_try = None
        if parsed_data_priority and
os.path.exists(PARSED_DATA_FILE):
            default_file_to_try = PARSED_DATA_FILE
        elif os.path.exists('all_defenses_data_v12.csv'):
            default_file_to_try = 'all_defenses_data_v12.csv'

        if default_file_to_try:
            print(f"Спроба завантажити файл за замовчуванням:
{default_file_to_try}")
            self.load_file_data(default_file_to_try)
        else:
            self.loaded_file_label.configure(text=f"Файли даних
({PARSED_DATA_FILE} або інші) не знайдено.")
            self.clear_table_and_details()
            if hasattr(self, 'add_specialty_button'):
self.add_specialty_button.configure(state="disabled")
            if hasattr(self, 'manual_check_button'):
self.manual_check_button.configure(state="disabled")
            if hasattr(self, 'show_all_data_button'):
self.show_all_data_button.configure(state="disabled")
            if hasattr(self, 'specialty_combobox'):
self.specialty_combobox.configure(values=["Спочатку завантажте
CSV"], state="disabled")
```

```
            if hasattr(self, 'filter_specialty_combobox'):
self.filter_specialty_combobox.configure(values=["Всі
спеціальності"], state="disabled")

    def load_csv(self):
        file_path = filedialog.askopenfilename(title="Оберіть CSV
файл", defaultextension=".csv", filetypes=[("CSV файли", "*.csv"),
("Всі файли", "*.*")])
        if file_path: self.load_file_data(file_path)

    def load_file_data(self, file_path):
        try:
            print(f"Завантаження даних з: {file_path}")
            try:
            for col in
self.dataframe.select_dtypes(include=['object']).columns:
                self.dataframe[col] =
self.dataframe[col].fillna('')

            print(f"Завантажено {len(self.dataframe)} записів.")
            self.loaded_file_label.configure(text=f"Завантажено:
{os.path.basename(file_path)} ({len(self.dataframe)} записів)")
            self.setup_treeview_columns()
            self.populate_specialty_comboboxes()

            self.showing_only_new_parsed = False
            if hasattr(self, 'show_all_data_button'):

self.show_all_data_button.configure(state="disabled")

            self.apply_filters()

            if self.current_user_login:
                 if hasattr(self, 'add_specialty_button'):
self.add_specialty_button.configure(state="normal")
                 if hasattr(self, 'manual_check_button'):
self.manual_check_button.configure(state="normal")

            if self.current_user_login and
self.user_tracked_specialties:
                print("Автоматична перевірка нових (не оброблених)
після завантаження файлу...")
                self.check_for_new_defenses(is_auto_check=True,
new_data_df=self.dataframe)

            if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="normal")

        except Exception as e:
            print(f"Помилка завантаження файлу {file_path}: {e}")
            self.loaded_file_label.configure(text=f"Помилка
завантаження: {os.path.basename(file_path)}")
```

```
            messagebox.showerror("Помилка", f"Не вдалося
завантажити файл: {e}")
            self.dataframe = None; self.clear_table_and_details()
            if hasattr(self, 'add_specialty_button'):
self.add_specialty_button.configure(state="disabled")
            if hasattr(self, 'manual_check_button'):
self.manual_check_button.configure(state="disabled")
            if hasattr(self, 'show_all_data_button'):
self.show_all_data_button.configure(state="disabled")
            if hasattr(self, 'run_parser_button'):
self.run_parser_button.configure(state="normal")
            if hasattr(self, 'specialty_combobox'):
self.specialty_combobox.configure(values=["Спочатку завантажте
CSV"], state="disabled")
            if hasattr(self, 'filter_specialty_combobox'):
self.filter_specialty_combobox.configure(values=["Всі
спеціальності"], state="disabled")

    def setup_treeview_columns(self):
        if self.dataframe is None or self.dataframe.empty:
            if hasattr(self, 'tree'): self.tree["columns"] = ()
            try:
                if hasattr(self, 'tree') and
self.tree.exists("#0"):
                    self.tree.column("#0", width=0, stretch=False)
                    self.tree.heading("#0", text="")
            except Exception as e:
                print(f"Помилка при налаштуванні колонки #0: {e}")
            return

        cols = list(self.dataframe.columns)
        if hasattr(self, 'tree'):
            self.tree["columns"] = cols
            self.tree["displaycolumns"] = cols
            self.tree.column("#0", width=0, stretch=False)
            self.tree.heading("#0", text="")

            for col in cols:
                col_width = 120
                if col in ["Тема дисертації", "Заклад освіти,
освітня програма"]: col_width = 300
                elif col in ["Публікації", "Разова рада"]:
col_width = 150
                elif col == "ПІБ здобувача": col_width = 200
                elif col == "ID": col_width = 60
                elif col == "Статус роботи": col_width = 100
                elif col in ["Стать здобувача", "Рік"]: col_width
= 80
                self.tree.column(col, width=col_width,
minwidth=50, stretch=False, anchor="w")

    def sort_treeview_column(self, col, reverse):
```

```
        if self.filtered_df is None or self.filtered_df.empty:
            return

        df_to_sort = self.filtered_df.copy()

        try:
            df_to_sort[col] = pd.to_numeric(df_to_sort[col],
errors='ignore')
            df_to_sort = df_to_sort.sort_values(by=col,
ascending=not reverse)
        except Exception as e:
            print(f"Помилка сортування колонки {col}: {e}")
            try:
                df_to_sort = df_to_sort.sort_values(by=col,
ascending=not reverse, key=lambda x: x.astype(str).str.lower())
            except Exception as e_str:
                print(f"Помилка сортування колонки {col} як
рядків: {e_str}")
                return

        self.filtered_df = df_to_sort
        self.populate_treeview(self.filtered_df)
        if hasattr(self, 'tree'): self.tree.heading(col,
command=lambda: self.sort_treeview_column(col, not reverse))


    def clear_table_and_details(self):
        if hasattr(self, 'tree'):
            for i in self.tree.get_children():
self.tree.delete(i)
        self.clear_details_pane()
        if hasattr(self, 'details_placeholder'):
            self.details_placeholder.configure(text="Завантажте
CSV файл або немає даних для відображення.")

    def populate_treeview(self, df_to_display):
        if not hasattr(self, 'tree'): return
        # --- DEBUG Print ---
        print(f"[DEBUG] Populate_treeview: Запуск з DataFrame
shape: {df_to_display.shape if df_to_display is not None else
'None'}")
        for i in self.tree.get_children(): self.tree.delete(i)

        if df_to_display is None or df_to_display.empty:
            print("[DEBUG] Populate_treeview: DataFrame порожній
або None, вихід.") # DEBUG
            if self.dataframe is None:
                self.clear_table_and_details()
                if hasattr(self, 'details_placeholder'):
self.details_placeholder.configure(text="Завантажте CSV файл або
немає даних.")
            else:
```

```
                self.clear_details_pane()
                if hasattr(self, 'details_placeholder'):
self.details_placeholder.configure(text="Немає даних для
відображення згідно фільтрів.")
            return

        if not self.tree["columns"]:
             if self.dataframe is not None and not
self.dataframe.empty:
                self.setup_treeview_columns()
             else:
                print("Помилка: Неможливо налаштувати колонки,
DataFrame порожній.")
                return

        cols_to_display = self.tree["columns"]
        if not isinstance(cols_to_display, (list, tuple)) or not
cols_to_display:
            print("Помилка: Колонки для Treeview не визначені або
порожні.")
            return
        print(f"[DEBUG] Populate_treeview: Використовуються
колонки: {list(cols_to_display)}") # DEBUG

        inserted_count = 0 # DEBUG
        for index, row in df_to_display.iterrows():
            row_values = [str(row.get(col_name, "")) for col_name
in cols_to_display]
            if len(row_values) == len(cols_to_display):
                try:
                    item_id = self.tree.insert("", "end",
iid=str(index), values=row_values)
                    inserted_count += 1 # DEBUG
                except Exception as e:
                    print(f"Помилка вставки рядка в Treeview
(індекс {index}): {e}")
            else:
                print(f"Помилка populate_treeview: кількість
значень ({len(row_values)}) != колонок ({len(cols_to_display)})
для індексу {index}")
        print(f"[DEBUG] Populate_treeview: Вставлено рядків:
{inserted_count}") # DEBUG


    def apply_filters(self):
        # (Без змін від v18 - ReCheck v1) ...
        if self.dataframe is None:
            print("Застосування фільтрів неможливе: DataFrame не
завантажено.")
            self.clear_table_and_details()
            return
```

```
        if self.showing_only_new_parsed and self.filtered_df is
not None:
             df_to_filter = self.filtered_df.copy()
             print("Застосування фільтрів до 'тільки нових'
даних.")
        else:
             df_to_filter = self.dataframe.copy()
             print("Застосування фільтрів до повного DataFrame.")

        try:
            filter_map = {
                'ПІБ здобувача': self.pib_filter_entry,
                'Тема дисертації': self.topic_filter_entry,
                'Ключові слова': self.keywords_filter_entry,
                'Заклад освіти, освітня програма':
self.institution_filter_entry
            }
            for col, entry_widget in filter_map.items():
                if hasattr(self, entry_widget.winfo_name()):
                    query = entry_widget.get().strip().lower()
                    if query and col in df_to_filter.columns:
                        print(f"Фільтрування за '{col}' містить
'{query}'")
                        df_to_filter =
df_to_filter[df_to_filter[col].astype(str).str.lower().str.contain
s(query, na=False, regex=False)]

            if hasattr(self, 'filter_specialty_combobox'):
                selected_filter_specialty =
self.filter_specialty_combobox.get()
                if selected_filter_specialty and
selected_filter_specialty != "Всі спеціальності" and 'Заклад
освіти, освітня програма' in df_to_filter.columns:
                    print(f"Фільтрування за спеціальністю:
'{selected_filter_specialty}'")
                    df_to_filter['extracted_spec'] =
df_to_filter['Заклад освіти, освітня програма'].apply(lambda x:
self.extract_specialty_code_and_name(str(x)))
                    df_to_filter =
df_to_filter[df_to_filter['extracted_spec'] ==
selected_filter_specialty]
                    df_to_filter =
df_to_filter.drop(columns=['extracted_spec'])

            if hasattr(self, 'year_filter_entry'):
                year_query_str =
self.year_filter_entry.get().strip()
                if year_query_str.isdigit() and 'Дата і час
захисту' in df_to_filter.columns:
                    year_query = int(year_query_str)
                    print(f"Фільтрування за роком захисту:
{year_query}")
```

```
                    temp_dates = pd.to_datetime(df_to_filter['Дата
і час захисту'], errors='coerce')
                    df_to_filter = df_to_filter[temp_dates.notna()
& (temp_dates.dt.year == year_query)]
                elif year_query_str and not
year_query_str.isdigit():
                    messagebox.showwarning("Некоректний рік",
"Будь ласка, введіть рік у форматі РРРР.")

        except Exception as e:
            print(f"Помилка під час фільтрації: {e}");
messagebox.showerror("Помилка фільтрації", f"Виникла помилка:
{e}")
            if self.showing_only_new_parsed and self.filtered_df
is not None:
                df_to_filter = self.filtered_df.copy()
            else:
                df_to_filter = self.dataframe.copy()

        self.filtered_df = df_to_filter
        print(f"Знайдено {len(self.filtered_df)} записів після
фільтрації.")

        self.populate_treeview(self.filtered_df)
        self.clear_details_pane()


    def reset_filters(self):
        # (Без змін від v18 - ReCheck v1) ...
        print("Скидання фільтрів.")
        filter_entries_attrs = ["pib_filter_entry",
"topic_filter_entry", "year_filter_entry",
"keywords_filter_entry", "institution_filter_entry"]
        for attr_name in filter_entries_attrs:
            if hasattr(self, attr_name): getattr(self,
attr_name).delete(0, "end")
        if hasattr(self, 'filter_specialty_combobox'):
self.filter_specialty_combobox.set("Всі спеціальності")
        self.show_all_parsed_data()


    def clear_details_pane(self):
        # (Без змін від v18 - ReCheck v1) ...
        if not hasattr(self, 'details_frame'): return
        for widget in self.details_frame.winfo_children():
widget.destroy()
        self.details_frame.update_idletasks()
        current_width = self.details_frame.winfo_width()
        if current_width <= 1: current_width = 400
        if hasattr(self, 'details_placeholder'):
```

```
            self.details_placeholder =
ctk.CTkLabel(self.details_frame, text="Оберіть рядок у таблиці для
перегляду деталей.",

wraplength=max(200, current_width - 40),
                                                    justify="center")
            self.details_placeholder.pack(pady=20, padx=10,
expand=True, fill="both")


    def create_link_button(self, parent, text, url):
         # (Без змін від v18 - ReCheck v1) ...
         def open_url():
             if url and isinstance(url, str) and
(url.startswith('http://') or url.startswith('https://')):
                 try: webbrowser.open_new_tab(url)
                 except Exception as e_wb:
messagebox.showwarning("Помилка посилання", f"Не вдалося
відкрити:\n{url}\n{e_wb}")
             else: messagebox.showwarning("Помилка посилання",
f"Некоректне посилання:\n{url}")
         button = ctk.CTkButton(parent, text=text,
command=open_url, width=20, height=20,
                                font=ctk.CTkFont(size=12),
text_color=("#0077CC", "#90CFFF"),
                                fg_color="transparent",
hover=False)
         return button


    def parse_and_display_complex_field(self, parent_frame,
field_name, field_value_str):
         # (Без змін від v18 - ReCheck v1) ...
        section_header_frame = ctk.CTkFrame(parent_frame,
fg_color="transparent")
        section_header_frame.pack(fill="x", pady=(7,3), padx=5)
        ctk.CTkLabel(section_header_frame, text=f"{field_name}:",
font=ctk.CTkFont(size=14, weight="bold")).pack(side="left",
anchor="w")

        if not field_value_str or not isinstance(field_value_str,
str) or \
           field_value_str.lower() in ["", "[]", "none", "nan"] or
\
           field_value_str.startswith("Не знайдено") or
field_value_str.startswith("Помилка парсингу"):
            ctk.CTkLabel(parent_frame, text="Дані відсутні або
некоректні",
                         text_color="gray",
                         wraplength=parent_frame.winfo_reqwidth()-
20,
```

```
                            justify="left",
anchor="w").pack(anchor="w", padx=10, pady=(0,5))
            return

        items_to_display = []
        try:
            parsed_items = ast.literal_eval(field_value_str)
            if isinstance(parsed_items, list):
                items_to_display = [item for item in parsed_items
if isinstance(item, dict)]
            elif isinstance(parsed_items, dict):
                 items_to_display.append(parsed_items)
            else:
                print(f"Поле '{field_name}' розібрано, але тип не
list/dict: {type(parsed_items)}")
                items_to_display = [{"Інформація":
str(parsed_items)}]

        except (ValueError, SyntaxError, TypeError) as e_eval:
            print(f"Помилка ast.literal_eval для '{field_name}':
{e_eval}. Спроба показати як текст. Рядок:
{field_value_str[:100]}...")
            items_to_display = [{"Інформація": field_value_str}]
        except Exception as e_parse_complex:
            print(f"Загальна помилка парсингу комплексного поля
'{field_name}': {e_parse_complex}")
            items_to_display = [{"Помилка відображення":
field_value_str[:200]+"..."}]


        if not items_to_display:
            ctk.CTkLabel(parent_frame, text="Дані не вдалося
розпізнати.",
                         text_color="gray",

wraplength=parent_frame.winfo_reqwidth()-20, justify="left",
anchor="w").pack(anchor="w", padx=10, pady=(0,5))
            return

        for item_idx, item_dict in enumerate(items_to_display):
             item_container_frame = ctk.CTkFrame(parent_frame,
fg_color=("gray92", "gray28"), corner_radius=5, border_width=1,
border_color=("gray80", "gray40"))
             item_container_frame.pack(fill="x", pady=4, padx=5)
             item_container_frame.grid_columnconfigure(1,
weight=1)

             key_order = []
             if field_name == "Публікації": key_order = ['Тип',
'Опис', 'Рік', 'Ключові слова', 'DOI', 'Посилання',
'Співавторство']
```

```
            elif field_name == "Разова рада": key_order = ['Тип',
'ПІБ', 'Установа та посада', 'Ступінь, спеціальність', 'ORCID']

            ordered_keys = [k for k in key_order if k in
item_dict] + [k for k in item_dict if k not in key_order]

            for row_num, key in enumerate(ordered_keys):
                value = item_dict.get(key, "")
                key_text = f"{key}:"
                key_label = ctk.CTkLabel(item_container_frame,
text=key_text, font=ctk.CTkFont(weight="bold"))
                key_label.grid(row=row_num, column=0,
sticky="ne", padx=(10, 5), pady=2)

                value_str = str(value).strip()
                if (key == "Посилання" or key == "ORCID" or key
== "DOI") and value_str.startswith(('http://', 'https://',
'doi.org')):
                    link_text = "🔗 Відкрити"
                    if key == "DOI" and not
value_str.startswith('http'):
                        value_url =
f"https://doi.org/{value_str}"
                    else:
                        value_url = value_str
                    link_button =
self.create_link_button(item_container_frame, link_text,
value_url)
                    link_button.grid(row=row_num, column=1,
sticky="nw", padx=5, pady=1)

                else:
                    value_textbox =
ctk.CTkTextbox(item_container_frame,

activate_scrollbars=False,
                                                   wrap="word",

border_width=0,

fg_color="transparent")
                    value_textbox.insert("1.0", value_str)
                    value_textbox.configure(state="disabled")

                    font = value_textbox.cget("font")
                    lines = value_textbox.get("1.0", "end-
1c").count('\n') + 1
                    line_height_approx =
font.metrics("linespace") + 2 if isinstance(font, ctk.CTkFont)
else 20
                    estimated_height = lines *
line_height_approx
```

```
                        max_height = 200
                        min_height = line_height_approx
                        final_height = max(min_height,
min(estimated_height, max_height))


value_textbox.configure(height=int(final_height))
                        value_textbox.grid(row=row_num, column=1,
sticky="nwe", padx=5, pady=1)

    def show_all_parsed_data(self):
        # (Без змін від v18 - ReCheck v1) ...
        print("Показ всіх завантажених даних.")
        self.showing_only_new_parsed = False
        if hasattr(self, 'show_all_data_button'):
            self.show_all_data_button.configure(state="disabled")
        if self.dataframe is not None:
            self.apply_filters()
        else:
            self.clear_table_and_details()

    def on_row_select(self, event=None):
        # (Без змін від v18 - ReCheck v2) ...
        selected_items = self.tree.selection()
        if not selected_items:
            self.clear_details_pane();
            return

        selected_item_iid = selected_items[0]
        for widget in self.details_frame.winfo_children():
widget.destroy()

        try:
            df_index = int(selected_item_iid)

            if self.filtered_df is None or df_index not in
self.filtered_df.index:
                print(f"Помилка: Індекс {df_index} не знайдено у
filtered_df.")
                ctk.CTkLabel(self.details_frame, text="Помилка:
дані для вибраного рядка не знайдено.").pack(pady=20, padx=10);
return

            actual_row_data = self.filtered_df.loc[df_index]
            column_order = list(self.dataframe.columns) if
self.dataframe is not None else list(actual_row_data.index)

            for col_name in column_order:
                if col_name not in actual_row_data: continue

                cell_value = actual_row_data.get(col_name, "")
```

```
                if col_name in ["Публікації", "Разова рада"]:

self.parse_and_display_complex_field(self.details_frame, col_name,
str(cell_value))
                else:
                    line_frame = ctk.CTkFrame(self.details_frame,
fg_color="transparent")
                    line_frame.pack(fill="x", pady=1, padx=5)
                    line_frame.grid_columnconfigure(1, weight=1)

                    key_label = ctk.CTkLabel(line_frame,
text=f"{col_name}:", font=ctk.CTkFont(weight="bold"))
                    key_label.grid(row=0, column=0, sticky="ne",
padx=(0,10))

                    value_str = str(cell_value).strip()
                    if col_name == "Посилання на роботу" and
value_str.startswith(('http', 'https')):
                        link_button =
self.create_link_button(line_frame, "🔗 Відкрити", value_str)
                        link_button.grid(row=0, column=1,
sticky="nw")
                    else:
                        value_label = ctk.CTkLabel(line_frame,
text=value_str, justify="left", anchor="nw")
                        self.details_frame.update_idletasks()
                        parent_width =
self.details_frame.winfo_width()
                        key_width = key_label.winfo_reqwidth()
                        wraplength = max(150, parent_width -
key_width - 40)

value_label.configure(wraplength=wraplength)
                        value_label.grid(row=0, column=1,
sticky="nwe", padx=5, pady=1)

        except ValueError:
             print(f"Помилка: Некоректний ідентифікатор рядка
Treeview: {selected_item_iid}")
             ctk.CTkLabel(self.details_frame, text="Помилка: Не
вдалося отримати дані для рядка.").pack(pady=20, padx=10)
        except KeyError as ke:
            print(f"Помилка KeyError при відображенні деталей
(індекс {df_index}): {ke}")
            ctk.CTkLabel(self.details_frame, text="Помилка
відображення: не знайдено рядок.").pack(pady=20, padx=10)
        except Exception as e:
            print(f"Загальна помилка при відображенні деталей:
{e}")
            import traceback
            traceback.print_exc()
```

```
            ctk.CTkLabel(self.details_frame, text=f"Помилка
відображення деталей: {e}").pack(pady=20, padx=10)


if __name__ == "__main__":
    try:
        from ctypes import windll
        windll.shcore.SetProcessDpiAwareness(1)
    except ImportError:
        pass
    except AttributeError:
        pass

    app = DefenseDataViewerApp()
    app.mainloop()
```