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

До захисту допущено:
Завідувач кафедри
_____ Олександр КОВАЛЬ
«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

спеціальності 122 «Комп’ютерні науки»

на тему: «Система автоматизованого збору наукометричних даних»

Виконав:
студент IV курсу, групи ТР-71
Попченко Вадим Михайлович _____

Керівник:
Старший викладач,
Дацюк Оксана Антонівна

Консультант:
_____ _____

Рецензент:
_____ _____

Київ — 2021 року

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

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 122 Комп’ютерні науки

Спеціалізація: Комп'ютерне геометричне моделювання процесів та систем

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

ЗАВДАННЯ
на дипломну роботу студенту
Попченко Вадиму Михайловичу
(прізвище, ім’я, по батькові)

1. Тема роботи _Система автоматизованого збору наукометричних даних_____
керівник роботи_ Дацюк Оксана Антонівна, старший викладач._
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

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

2. Строк подання студентом роботи _____.06.2021р.
3. Вихідні дані до роботи _мова програмування C#, фреймворк ASP.NET, середовище розробки Visual Studio 2019.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розробити веб-застосунок для збору наукометричних даних; проаналізувати існуючі методи класифікації та фільтрації; розробити метод для парсингу, аналізу та класифікації даних та збереження до реферативної бази даних.

5. Перелік ілюстративного матеріалу

Актуальність, Постановка задачі, Стилі керування, Ідентифікація парсера, Архітектура системи, Алгоритм виявлення та ідентифікації наукометричних даних, Діаграма класів, Діаграма розгортання, Модель бази даних, Діаграма прецедентів, Сторінки авторизації, Інтерфейс головної сторінки, Збір та аналіз даних, Сторінка завантаження, Висновки.

6. Консультанти розділів роботи

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

7. Дата видачі завдання ”_15_”_____березня_____ 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	15.03.2021	
2.	Вивчення та аналіз задачі	15.03.2021-12.04.2021	
3.	Розробка архітектури та загальної структури системи	13.04.2021-18.04.2021	
4.	Розробка структур окремих підсистем	19.04.2021-02.05.2021	
5.	Програмна реалізація системи	03.05.2021-12.05.2021	
6.	Оформлення пояснювальної записки	13.05.2021-23.05.2021	
7.	Захист програмного продукту	13.05.2021	
8.	Передзахист	27.05.2021	
9.	Захист	16.06.2021	

Студент _____ _____Попченко В.М._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Дацюк О.А._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

У роботі були розглянуті теоретичні та практичні аспекти розробки програмного продукту для автоматизованого збору наукометричних даних.

Метою роботи є створення автоматизованої системи для збору даних та реферативної бази даних.

Для досягнення мети було розроблено веб-застосунок на основі трьох рівненої структури, реалізовано алгоритм збору, аналізу та класифікації даних, розроблено метод розширення системи у майбутньому.

Пояснювальна записка складається зі вступу, п’яти розділів, висновку, списку використаних джерел; містить 50 сторінок, 34 рисунки та 3 додатки. Список використаних джерел включає 11 бібліографічних найменувань.

Ключові слова: веб-застосунок, наукометричні дані, реферативна база даних, парсинг, синтаксичний аналізатор.

ABSTRACT

The paper considers theoretical and practical aspects of software product development for automated collection of scientometric data.

The purpose of this work is to create an automated system for data collection and an abstract database.

To achieve this goal, a web application based on three equal structures was developed, an algorithm for data collection, analysis and classification was implemented, and a method for expanding the system in the future was developed.

The explanatory note consists of an introduction, five sections, a conclusion, a list of sources used; contains 50 pages, 34 figures and 3 applications. The list of used sources includes 8 bibliographic items.

Keywords: web application, scientometric data, abstract database, parsing, parser.

ЗМІСТ

Перелік умовних позначень, скорочень і термінів 8
Вступ 10
1. Завдання розробки 11
2. Опис предметної області 12
2.1 Можливості та використання Scopus 13
2.2 Використання Google Scholar, як основа для збору даних 14
2.3 Опис та аналіз предметної області 16
2.4 Аналіз функціональних особливостей системи 16
2.5 Висновок до розділу 17
3. Засоби розробки 18
3.1 Мова програмування C# 18
3.2 Середовище розробки Visual Studio 2019 19
3.3 СКБД Microsoft SQL Server Management Studio 2019 20
3.4 Framework ASP.NET Core для розробки веб-застосунків 21
3.5 Використання патерну MVC 23
3.6 Технологія Entity Framework Core 25
3.7 Технологія використання парсингу 26
3.8 Мова програмування JavaScript, HTML та CSS 28
3.9 Система контролю версій Git 29
3.10 Висновок до розділу 30
4. Опис програмної реалізації 31
4.1 Реалізація парсера на основі бібліотеки AngleSharp 31
4.2 Реалізація доступу до бази даних 32
4.3 Структура проекту 34
4.4 Діаграма прецедентів 35
4.5 Архітектура класів 37
4.6 Опис структури бази даних 39
4.7 Висновок до розділу 40

5. Методика роботи користувача з програмною системою 41
5.1 Системні вимоги та інсталяція 41
5.2 Використання розробленої системи 41
5.3 Висновок до розділу 50
Висновки 51
Список використаних джерел 52
Додаток А. Специфікація 53
Додаток Б. Текст програми 55
Додаток В. Опис програмного модулю 62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AngleSharp — це бібліотека, яка дозволяє працювати з об'єктними моделями HTML, полегшує розробку програмних додатків при роботі з браузерними сторінками.

ASP.NET CORE — це новітня технологія для розробки веб-застосунків і різних веб-сервісів. Була розроблена компанією Майкрософт. Технологія заснована на платформі .NET.

C# — це об'єктно-орієнтована мова програмування, розроблена Андерсом Хейлсбером та Скоттом Вільтаумотом в 1998 — 2001 роках. Мова набула величезної популярності через свій лаконічний та зрозумілий синтаксис.

Entity Framework Core — це фреймворк, який є структурою об'єктно-реляційного представлення з усіма можливостями більш старого ADO.NET. Ця технологія полегшує розробку моделей баз даних при створенні програмних додатків.

Google Scholar — це безкоштовна пошукова система, яка займається пошуком наукових публікацій, дисциплін всіх форматів. Працює з більшістю ліцензованих онлайн-журналів Америки та Європи.

MVC — це архітектурний шаблон, який передбачає поділ розроблюваної системи на три частини. Зазвичай використовується для розмежування даних від користувацького інтерфейсу. Таким чином, щоб зміни в інтерфейсі мінімально або взагалі не змінювали на роботу з даними, а зміни в розробленій моделі даних могли здійснюватися без видозміни інших елементів системи.

MVP — це архітектурний шаблон, який був розроблений на основі MVC, основною ідеєю якого є відділення візуального користувацького інтерфейсу та бізнес-логіку обробки подій.

MVVM — це архітектурний шаблон, який використовується під час розробки застосунків. Полегшує розділення створення інтерфейсу від розробки логіки та моделі. Цей шаблон був створений з метою розділу праці дизайнера та програміста.

Scopus — це бібліографічний центр, база даних та інструмент для роботи з цитованістю статей. Індексує назви видань з медичних, гуманітарних та технічних наук.

Visual Studio — це інтегроване середовище розробки різноманітного програмного забезпечення та великої кількості інших інструментальних засобів. Є продуктом фірми Майкрософт. Середовище надає зможу розробляти як примітивні консольні програми, так і програми з графічним середовищем, веб-застосунки тощо.

Бібліотека — це збірка запрограмованих елементів, підпрограм, яка використовується при розробці програмного забезпечення. Полегшує розподілу розробку додатків, так як одну бібліотеку можна використовувати в багатьох програмах.

Веб-застосунок — це розроблений інтернет-додаток, в якому браузер є клієнтом, користувачем, а сервер виступає відповідачем. Особливістю є те, що функції мають виконуватися незалежно від браузера чи операційної системи користувача.

СКБД — набір взаємопов'язаних даних база даних і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

ВСТУП

При сучасному розвитку інформаційних технологій люди все частіше використовують різноманітні веб-сайти для пошуку наукових статей, конференцій та інших видів наукометричної інформації. Саме тому створюються веб-застосунки, які б мали змогу забезпечити комфортний пошук та доступ до цих даних. Зі збільшенням різного роду контенту та саме науковців постає питання в розробці такого ресурсу, де можна було б знайти інформацію про викладачів та їхні статті, зібрані з більшості відомих інформаційно-бібліотечних центрів. Оскільки на даних момент немає великої кількості агрегаторів наукових даних, які б містили посилання на різні інформаційні центри, актуальною є задача створення системи, яка б автоматизовано займалась збором цієї інформації.

Метою є створення системи, яка збиратиме наукометричні дані з різних інтернет ресурсів.

У наш час є схожі системи до даної. Наприклад, веб-застосунок Google Scholar збирає наукові статті з усього інтернету, надає доступ користувачам на публікацію власних статей та цитування статей інших користувачів. Scopus має схожі функції, але в порівнянні з вищенаведеним ресурсом має більш обмежені можливості, але вона особлива своїм глибоким аналізом статей та науковців.

В розробленій системі я створив можливість збору інформації зі Google Scholar про науковців, їх статті та можливість вибору перегляду інформації про них в інших інформаційно-бібліотечних центрах таких як Scopus та Електронний архів Київського політехнічного інституту Ігоря Сікорського. Як на мене це дасть змогу користувачам простіше та комфортніше здійснювати пошук різноманітних наукометричних даних.

1. ЗАВДАННЯ РОЗРОБКИ

Завданням роботи є створення системи, яка збиратиме наукометричні дані з різних інтернет ресурсів, надаватиме їх користувачам, які зможуть переглядати короткі відомості про статті, науковців, наукові напрями та інше, а також переходити на оригінальні веб-сторінки зібраних даних. Користувачі матимуть можливість сортувати, відокремлювати та зберігати статті.

Автоматизована система була розроблена у вигляді веб-застосунку для будь-якого браузера та будь-якої операційної системи. Систему можна як запускати зі свого персонального комп'ютера так і поставити її на віддалений хостинг та використовувати як веб-сайт.

Користувачем може бути будь-який бажаючий знайти наукометричні дані, почитати статті чи дізнатись щось нове про бажаного йому науковця чи викладача.

Наукометричними даними є реферативна та бібліографічна база даних з інструментами для аналізу цитованості статей, опублікованих у різноманітних наукових виданнях.

Вхідними даними в даній системі є лише унікальній ідентифікатор викладача чи науковця, який легко знайти в пошуковому запиті на Google Scholar. Також цей ідентифікатор можна знайти на сторінці науковця. Більше ніяких вхідних даних системі не потрібно. Це було зроблено таким чином, задля простоти використання. Вихідними даними є заповнена реферативна база даних, яка містить в собі інформацію про науковця, його статті та наукові напрямки з якими він працює.

В ході розробки автоматизованої системи було вирішено наступні задачі:

1) Алгоритм пошуку та збору наукометричних даних;
2) Алгоритм збереження та роботи з даними для їх подальшого аналізу;
3) Можливість додавати інформацію з інших інтернет ресурсів;
4) Створення автоматичного завантаження викладачів та науковців з кафедри АПЕПС ТЕФ.

2. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

В історії відомі різні типи та напрямки наукових публікацій. В еру, коли ще не було друкарських засобів та і тоді, коли з’явилася поліграфія, коли створення та публікація записів або надрукованих книг та статей, було занадто важким, довгим у розробці, трудомістким та дорогим, наукові статті та знання різного роду поширювались переважно у формі наукового трактату, тобто одні науковці переказували їх один одному. Таким чином людство наповнювалося інформацією.

Праці, статті, записки писалися без великого поспіху, зазвичай, вони мали в собі фундаментальний характер, були результатом багаторічних досліджень та практичних досліджень. Навіть тоді, коли дослідження доходило до стадії публікації, підготовка до публікації займала доволі багато часу, а сама публікація такої роботи майже завжди ставала значною подією не тільки в науковому, а й у суспільному житті звичайних людей.

Але з розвитком друкарства, науки людство повинно було придумати швидший спосіб поширення інформації. Зі століттями, роками були вигадані бібліотеки, архіви та інші способи. Сьогодні ж інформації стало на стільки багато, що ніяка фізичка бібліотека не в змозі зберігати всі записи в собі. Тому на заміню фізичним бібліотекам приходять інтернет-ресурси, інтернет-бібліотеки, які можуть мати в собі всі записи починаючи від перших рукописів ще до нашої ери, так і новітні записи та статті.

На сьогодні існують системи схожі до розробленої. Наприклад, веб-застосунок Google Scholar є агрегатором величезної кількості наукових статей, зібраних з різних наукових сайтів, форумів та інших ресурсів. Користувач здійснює пошук по можливій назві статті, імені викладачах та інших полях. Наукометричний портал Scopus має подібні функції до Google Scholar. Також є доволі цікавий приклад у вигляді веб-застосунку Web of Science, який має велику кількість пошукового функціоналу, але мінусом є те, що користувач має щомісяця купляти підписку на цей веб ресурс.

В розробленій системі було створено можливість збору інформації зі Google Scholar та його подальшого аналізу в базі даних, можливість вибору перегляду інформації про них в інших інформаційно-бібліотечних центрах таких як Scopus та Електронний архів Київського політехнічного інституту Ігоря Сікорського, редагування в подальшому адміністратором. Саме ці особливості надають змогу користувачам простіше та комфортніше і в більшому об’ємі отримувати інформацію про науковців, викладачів та їх статті, цитованість та інше.

2.1. Можливості та використання Scopus

Scopus — одна з найбільших баз даних рефератів та цитування рецензованої літератури та якісних веб-джерел із розумними інструментами для відстеження, аналізу та візуалізації досліджень.

Scopus — це програмний застосунок, який виконує функції бібліографічного центру. Бібліографічним центром називають базу даних та інструменти для роботи зі статтями. Його головною функцією є індексування назви видань з медичних, гуманітарних та технічних наук. Приклад використання застосунку Scopus наведено нижче (рисунок 2.1).

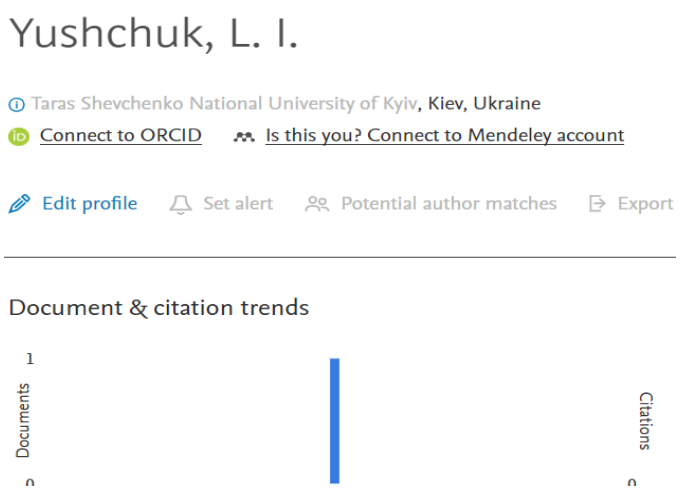

Рисунок 2.1 — Приклад використання Scopus

Науковці та статті потрапляють у цей ресурс автоматично, адже журнал, у якому було опубліковано його статтю входить до якогось переліку й збирається

відповідною базою даних. Збірники видань та журнал, які використовує Scopus постійно оновлюються. Вони відповідають жорстким критеріям науковості та обов'язково проходять суворий експертний відбір для потрапляння до реферативної бази даних.

Для авторів, які мають більше одного публікування, Scopus створює унікаьний обліковий запис для цього науковця. Профіль автора має унікальний ідентифікатор. Такі аккаунти мають можливість надавати інформацію, загальна кількість джерел, на які посилається автор, місце видання автора, роки публікацій, різноманітні індекси науковця тощо. Ця реферативна база даних дає можливість користувачам виконувати пошук по унікальним запитам, які ідентифікують авторів, їх статті, записи тощо. Scopus є досить серйозним сховищем наукометричних даних, який на даних момент нараховує більше 50 млн. записів, що характеризує його як один із найбільших онлайн бібліотек.

2.2. Використання Google Scholar, як основа для збору даних

Google Scholar — програмний засіб, розроблений компанією Google. Він був створений для пошуку великої кількості статей, зібраних з усього інтернету.

Google Scholar показує цитати на статті, звіти, книги в Інтернеті та інші матеріали, що відображаються в Інтернеті. Її пошукові запити охоплюють наукові матеріали частіше, ніж звичайний Google.

Google Scholar здійснює широкий пошук, тому майже завжди щось знаходить. Багато баз даних дозволяють Google бачити їх вміст, і багато елементів, які ви можете відкрити. Використовуйте тематичні бази даних та Google Scholar для більш повного пошуку. Вам доведеться оцінити те, що ви виявили, щоб переконатися, що якість кожного джерела, яке ви використовуєте, є адекватною для академічної роботи.

Google Scholar надає можливість користувачам знаходити цифрові копії інформації про викладачів та їх статей в онлайн-бібліотеках. Він працює таким чином, що за допомогою індексації повнотекстових статей, технічні звітів, записок, журналів

інших наукових документів, виконує пошук, знаходить бажані веб-сторінки з науковою інформацією. Так як більшість результатів пошуку в Google Scholar надають посилання на комерційні статті, люди мають доступ лише до малої кількості інформації такої як опис чи цитування самої статті. За більш розширену інформацію користувачеві доведеться платити за підписку, що, як на мою думку, є мінусом. Приклад використання застосунку Google Scholar наведено нижче (рисунок 2.2).

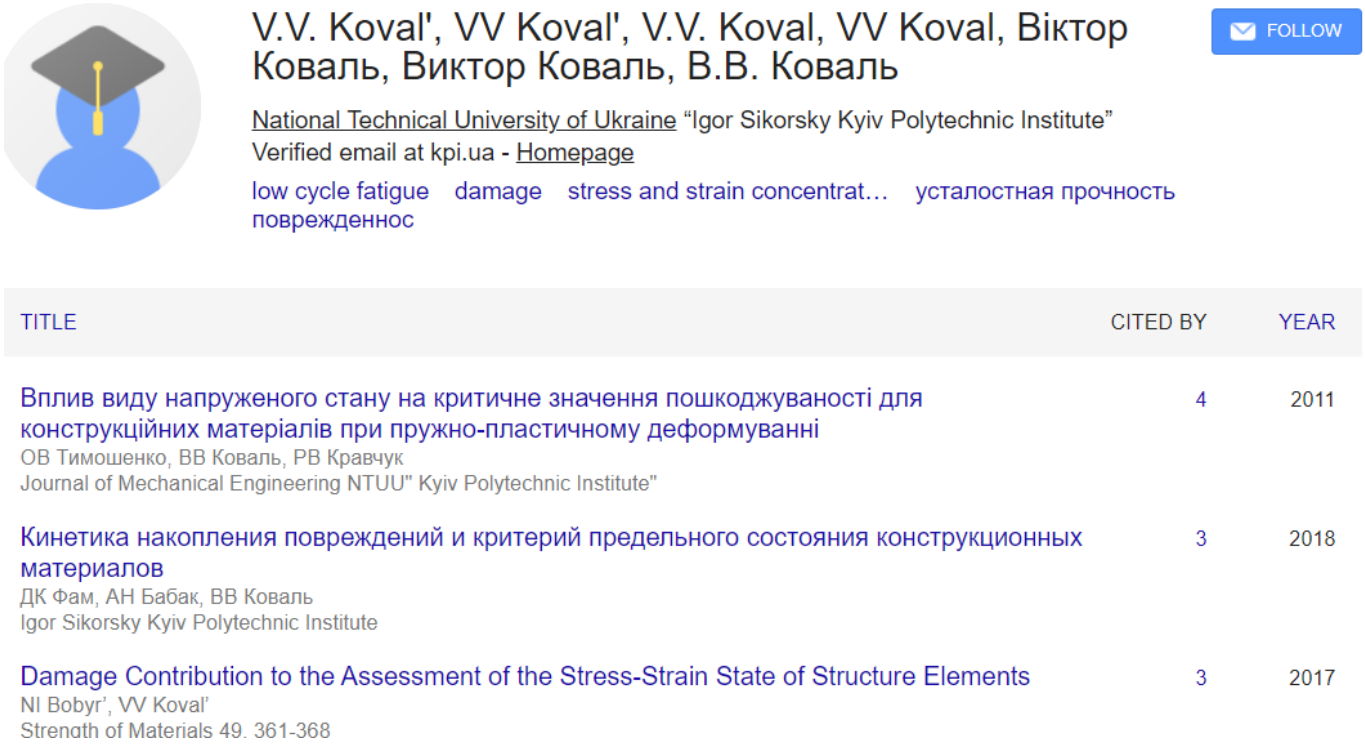

Рисунок 2.2 — Приклад використання Google Scholar

При розробці веб-застосунку та автоматизованої системи, Google Scholar був обраний, як основний ресурс звідки буде завантажуватися інформація в базу даних. Є декілька причин цьому: перша — це те, що ця інформаційна бібліотека має найбільшу кількість науковців, і в наслідок цього, найбільшу кількість статей, друга — так як це ресурс Google, ніколи не буде перебоїв з доступом до нього, це означає, що розроблена автоматизована система не матиме проблем при роботі з Google Scholar, завжди можна буде додати нового науковця, завжди можна буде завантажити кілька нових статей.

2.3. Опис та аналіз предметної області

Автоматизована система збору наукометричних даних — це програмне забезпечення, головною функцією якого є збір, парсинг даних з інформаційно-бібліотечних центрів та завантаження цих даних до бази даних.

Зазвичай такі парсери будуються на основі запитів до сервера і подальшої обробки відповіді. В даному випадку веб-застосунок виступає клієнтом, який хоче отримати веб-сторінку, а віддалений сервер Google Scholar надає цю сторінку клієнту. Хорошим плюсом такого виду збору даних є те, що не потрібно купляти підписки на віддалені API, але є і мінус в тому, що такий спосіб є дещо повільнішим, але звичайний користувач ніяким чином не помітить цього.

2.4. Аналіз функціональних особливостей системи

Веб-застосунок або ж автоматизована система збору наукометричних даних має два типи користувачів: гість та адміністратор.

Гість має доступ до таких розділів сайту:

– Викладачі, який містить перелік всіх викладачів та можливість відсортувати їх;
– Статті, який містить перелік всіх статей та можливість їх перегляду, сортування та скачування;
– Наукові напрями, який містить перелік всіх напрямів, з якими працюють науковці та викладачі, таким чином можна буде знайти науковців за їх науковим фахом;
– Пошук за місцем викладання науковця.

Тобто гість має можливість без всіляких проблем знайти статті, які його цікавлять, або ж інформацію про науковців чи викладачів, їх місце роботи, індексування.

Адміністратор має доступ до таких розділів сайту:

– Головна, на якому можна керувати всім сайтом;
– Викладачі, Наукові напрями, Статті — відповідно управляти кожним із цих списків окремо (редагувати або ж видаляти записи, тощо);
– Налаштування, де користувач-адміністратор може змінити свій пароль до облікового запису;
– Парсер — найголовніша сторінка веб-застосунку, саме на ній відбувається збір даних з сайтів, управління та завантаження нових науковців та статей;
– Guid, на якому описана інструкція з використання панелі адміністратора та всього сайту.

Адміністратор має доступ до всіх розділів, що надає йому легке та комфортне управління сайтом.

Система адаптивна також як для персональних комп’ютерів так і для мобільних пристроїв, що робить її ще більш комфортною у використанні.

2.5. Висновок до розділу

Було розглянуто вже існуючі аналоги створюваної системи для різних платформ, які збирають наукометричну інформацію з усього інтернету.

Було проаналізовані різні особливості системи та моменти, які повинні бути реалізовані в застосунку.

Розглянуті програми мають переваги:

— Гарний та сучасний інтерфейс;
— Розроблені алгоритми, які автоматично обчислюють індекси;
— Безкоштовний перегляд початкової інформації.

Недоліки:

— Для перегляду всіх даних про статті, користувач має робити платну підписку;
— На головній сторінці не можна фільтрувати дані про статті та науковців.

3. ЗАСОБИ РОЗРОБКИ

Для розробки програмного забезпечення в сучасних реаліях не можливо не використовувати Інтернет. Саме тому постає питання в тому, щоб засоби розробки відповідали сьогоденним тенденціям роботи з засобами Інтернет. Також система повинна бути доволі швидкою, так як це веб-застосунок. Саме тому для написання програми я обрав мову програмування C#. Вона сповна відповідає технічним потребам щодо створення веб-застосунку, так як ця мова програмування постійно оновлюється було вирішено використовувати останню версію .NET 5. Розробка велась у середовищі Visual Studio 2019.

3.1. Мова програмування C#

C# — це об'єктно-орієнтована мова програмування, яка була розроблена з метою забезпечити безпечну типізацію для основної платформи .NET. Компанією розробником виступає Microsoft Research.

Синтаксис дотримується строгої статичної типізації змінних, також він підтримує всі можливі властивості об’єктно-орієнтованого програмування такі як: перевантаження операторів, поліморфізм, атрибути, події, вказівники на функції-члени класів, властивості, винятки тощо.

C# підтримує препроцесорні директиви на основі препроцесора C, це надає розробнику можливість визначити символи, швидку розробку. Різні умовні директиви, такі як #if, #endif чи #else можуть бути використані під час написання коду. Директиви типу #region дають команду IDE для згортання та розгортання фрагментів коду.

Взагалі мова програмування C# розроблялася як мова програмування складового рівня, яка б використовувала в CLR, тобто б вона виконували всі

можливості самого CLR. Це стосується найбільше системи типів, яка використовується в C #, яка саме відповідає FCL.

3.2. Середовище розробки Visual Studio 2019

Microsoft Visual Studio 2019— одна із версій продукту фірми Майкрософт, яка містять інтегроване середовище розробки програмного забезпечення та інших інструментальних засобів. (рисунок 3.1).

Рисунок 3.1 — Логотип IDE Visual Studio

Цей продукт дає змогу розробляти як консольні програми, так і програми з графічним інтерфейсом, включно з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Visual Studio має в собі редактор коду, який підтримує IntelliSense, а також рефакторинг коду. Вбудований відладчик працює як відладчик вихідного рівня, так і відладчик машинного рівня. Інші вбудовані інструменти включають Веб-дизайнер, кодовий профіль, конструктор класів та конструктор схем баз даних, конструктор для створення додатків з графічним інтерфейсом. Він включає в себе плагіни, що розширюють функціональність майже на кожному рівні, включаючи додавання підтримки систем управління версіями та додавання нових наборів інструментів,

таких як редактори і візуальні дизайнери для мов, специфічних для домену, або наборів інструментів для інших аспектів життєвого циклу розробки програмного забезпечення. Нижче зображений інтерфейс редактору (рисунок 3.2).

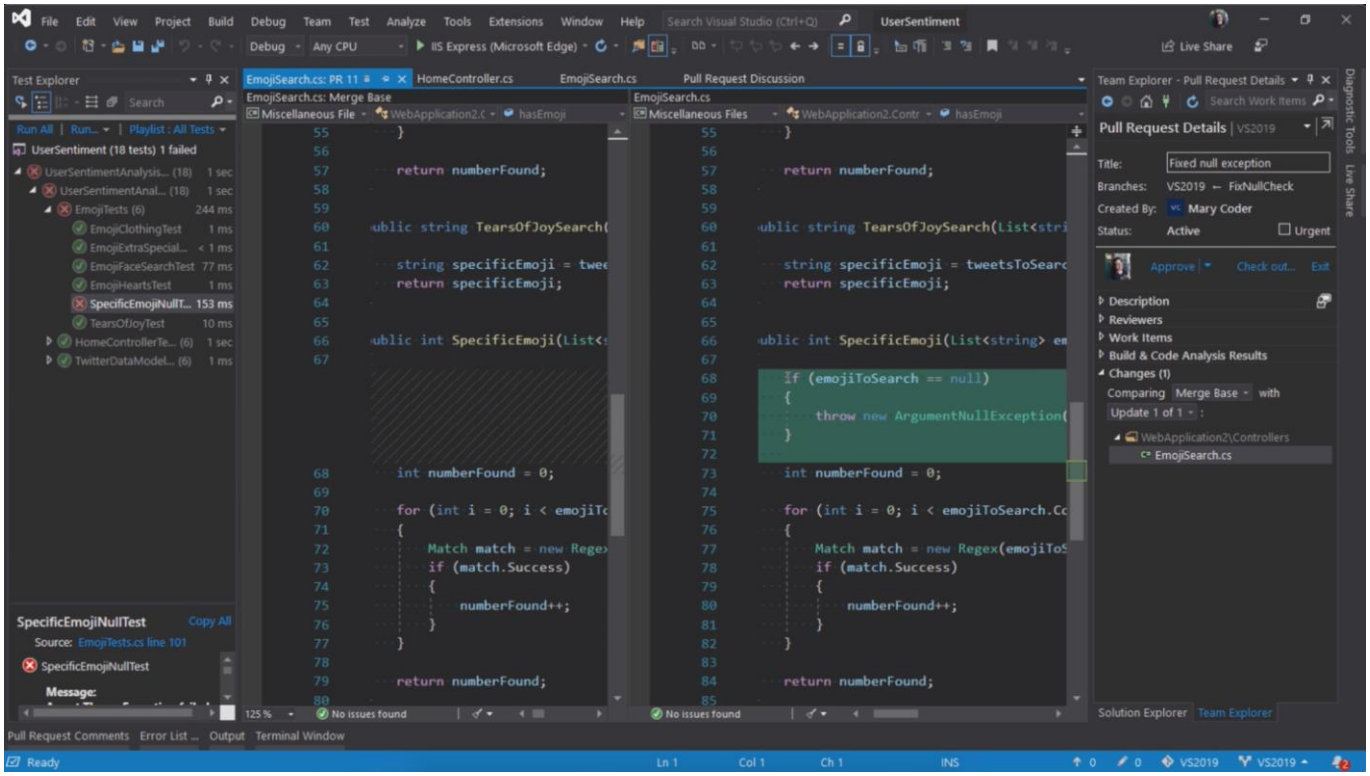

Рисунок 3.2 — Інтерфейс IDE Visual Studio

Особливістю та великим плюсом Microsoft Visual Studio є те, що цей продукт постійно оновлюється, користувач може використовувати тільки ті елементи розробки, які йому потрібні, інші ж просто не завантажувати на персональний комп’ютер або ноутбук. Також Microsoft Visual Studio допомагає при написанні коду, радить використовувати різні методики програмування.

3.3. СКБД Microsoft SQL Server Management Studio 2019

Для створення бази даних використовувалася система керування базами даних (СКБД) Microsoft SQL Server Management Studio 2019.

SQL Server, також відома як SSMS, — це інтуїтивно зрозумілий та багатоцільовий інструмент розробки та адміністрування баз даних, який в основному використовується фахівцями SQL, включаючи розробників баз даних SQL, адміністраторів баз даних та команди інфраструктури для управління середовищами SQL Server.

Студія управління SQL Server (SSMS) використовується для керування як екземплярами SQL Server, так і його базами даних, ефективністю та швидкістю, включаючи нестандартні функції безпеки.

З часом він був вдосконалений компанією Microsoft і пройшов ретельне тестування, гарантуючи, що він забезпечує як розробку баз даних, так і адміністрування серверів, щоб підтримувати роботу вашого середовища баз даних.

Це один з найпопулярніших інструментів керування базами даних та адміністрування серверів для SQL Server, і Microsoft постійно вдосконалює функції цього інструменту.

Студія управління SQL Server (SSMS) — це високопродуктивний набір інструментів для розробки та управління базами даних SQL, що пропонує загальну підтримку для налаштування, адміністрування та управління серверами баз даних SQL. Студія управління SQL Server (SSMS) — це графічний інструмент для створення та управління базами даних SQL, а також підтримка завдань адміністрування сервера.

На думку Microsoft, SQL Server Management Studio (SSMS) — це інтегроване середовище для управління будь-якою інфраструктурою SQL.

3.4. Framework ASP.NET Core для розробки веб-застосунків

Особливістю ASP.NET є те, що він побудований на CLR, що надає можливість розробникам писати код ASP.NET, використовуючи будь-яку підтримувану мову .NET.

ASP.NET — це технологія розробки веб-застосунків і веб-сервісів від компанії-гіганта Майкрософт. Вона є складовою частиною платформи Microsoft.NET і

розвитком старішої технології Microsoft ASP. На сьогодні останньою версією цієї технології є ASP.NET Core 6.0.

Технологія ASP.NET широко поширена при розробці хмарних застосунків, веб-сайтів, різноманітних API.

Розробники можуть писати програмний код для ASP.NET, використовуючи майже будь-які мови програмування, які входять у комплект .NET Framework. В свою чергу ASP.NET має перевагу у швидкості виконання в порівнянні зі скриптовими технологіями, тому що при першому запиті код компілюється і поміщається в спеціальний кеш, і пізніше тільки виконується, не вимагаючи витрат часу на синтаксичний розбір , оптимізацію. Нижче графічно наведено плюси використання фреймворку ASP.NET Core (рисунок 3.3).

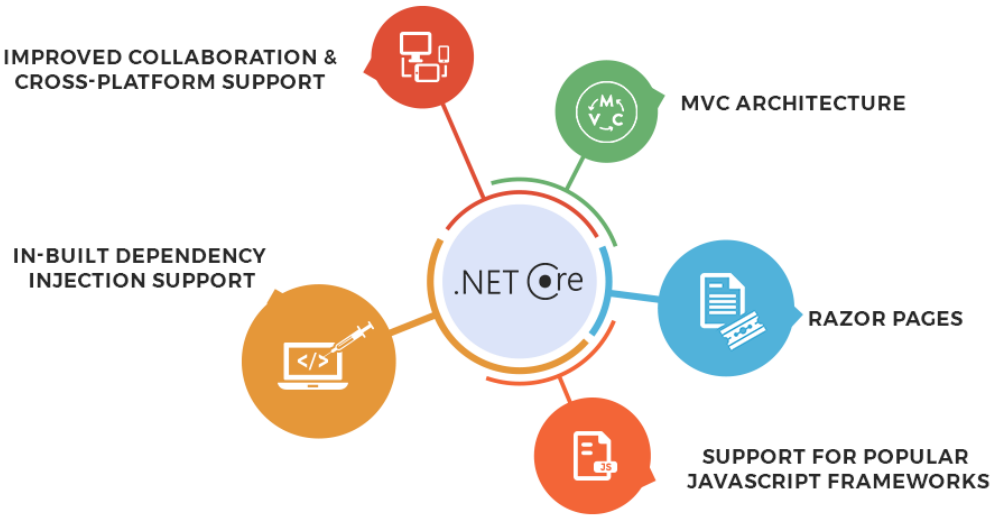

Рисунок 3.3 — Плюси використання ASP.NET Core

Переваги використання ASP.NET:

– ASP.NET має перевагу у швидкості в порівнянні з іншими технологіями, заснованими на скриптах (PHP, тощо);
– Розширюваний набір елементів управління;
– Розширюваний набір бібліотек ;
– ASP.NET спирається на багатомовні можливості .NET, що дає змогу писати код сторінок мовами C#, VB, C/C++ та інші;

– Поділ візуальної частини та бізнес-логіки.

3.5. Використання патерну MVC

Основною ідеєю використання архітектурних патернів MVC, MVVM, MVP, MVI є розподіл логіки та UI/UX частини програми. Це є запорукою правильно програмування, адже модулі таким чином менш зв’язані і їх легше тестувати окремо.

Моделлю, як правило, розуміється частина, що містить у собі функціональну програму для бізнес-логіки. Модель повинна бути повністю незалежною від інших продуктів. Модель володіє знаннями про себе і не знає про контролери та представлення.

Якщо взяти патерн MVP (рисунок 3.4), то презентер має доволі цікаві обов’язки та можливості у використанні. Він має двосторонню комунікацію з представленням.

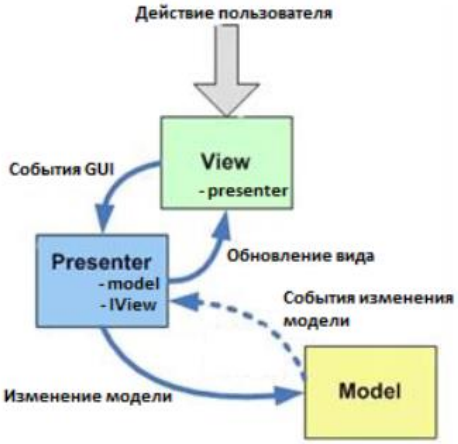

Рисунок 3.4 — MVP

Інший архітектурний патерн під назвою MVVM дозволяє зв’язувати елементи представлення з властивостями та подіями моделі-представлення. В даному випадку можна сказати, що кожен елемент патерну не знає про існування кожного іншого шару. Схема патерну MVVM зображена нижче (рисунок 3.5).

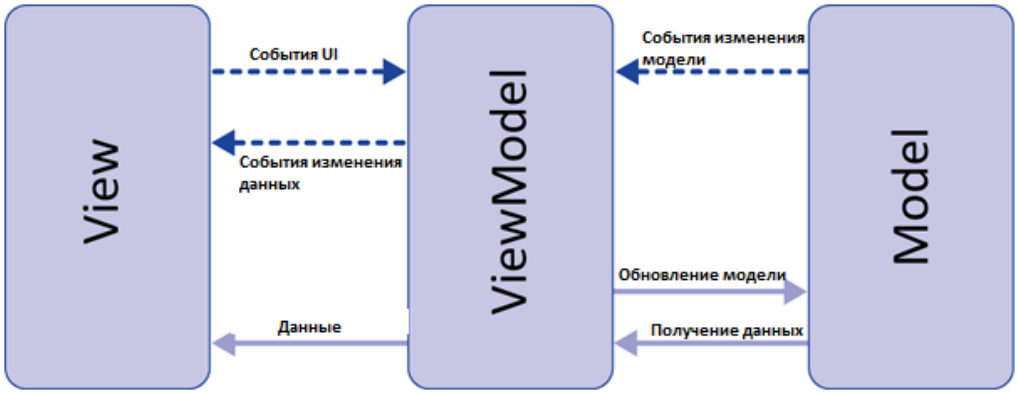

Рисунок 3.5 — MVVM

Але більшого використання при створенні веб-застосунків набув архітектурний патерн MVC. Основною ідея цього патерну в тому, що і контролер і представлення залежать від моделі, але модель ніяк не залежить від цих двох компонент. Схема патерну MVC зображена нижче (рисунок 3.6).

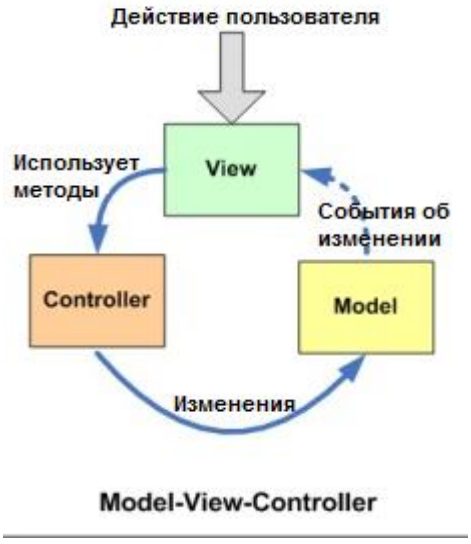

Рисунок 3.6 — MVC

Контролер перехоплює подію ззовні і відповідно до закладеної в нього логіки, реагує на цю подію, змінюючи модель, за допомогою виклику відповідного методу. Після зміни моделі чи, можливо створення нової моделі, контролер створює екземпляр представлення та повертає її користувачеві.

3.6. Технологія Entity Framework Core

Entity Framework — це набір технологій в ADO.NET, які підтримують та полегшують розробку програмно-орієнтованих на дані програмних додатків. Ця технологія допомагає автоматично моделювати сутності, взаємозв'язки та логіку бізнес-проблем, які вони вирішують, а також повинні працювати з інтерфейсами різних даних, що використовуються для зберігання та отримання даних.

Entity Framework має кілька підходів розробки:

- Code First
- Database First

Перший — Code First. Передбачається, що ви спочатку пишете код на C #, а потім база даних створюється з цього коду. Для цього підходу дуже важливо визначити класи моделі або сутності, які будуть зберігатися в базі даних, описати її в класах C # як модель та написати контекстний клас, який буде працювати з базою даних, що використовується. Підхід Code First — це найбільш уживаний програмістами на C #.

Другий підхід, Database First, підходить для тих, хто добре знає SQL, але в цьому випадку не потрібно знання C #. Першим кроком є створення бази даних, потім генерується модель бази даних EDMX. Цей XML у файлі .edmx містить інформацію про структуру бази даних, модель даних та їх співставлення між собою. Visual Studio має графічного дизайнера, який допоможе вам працювати з .edmx

Model-First — це третій підхід ORM. Він часто використовується архітекторами, оскільки такий підхід може не знати синтаксису SQL або C#. У цьому випадку спочатку створюється графічна модель EDMX, після чого у фоновому режимі створюються класи моделі C#, а потім база даних створюється на основі діаграми EDMX.

Усі таблиці бази даних визначені в Entity Framework як класи моделі або сутності, як правило, засновані на 1 таблиці, як Користувачі, — 1 клас у .NET, як

Користувач. Ці пари називаються умовами і визначаються в класі контексту даних як DbSets, і це підхід за замовчуванням.

Хоча існують такі механізми, як API Fluent та анотації даних, можна замінити ці правила або додаткові правила конфігурації (рисунок 3.7).

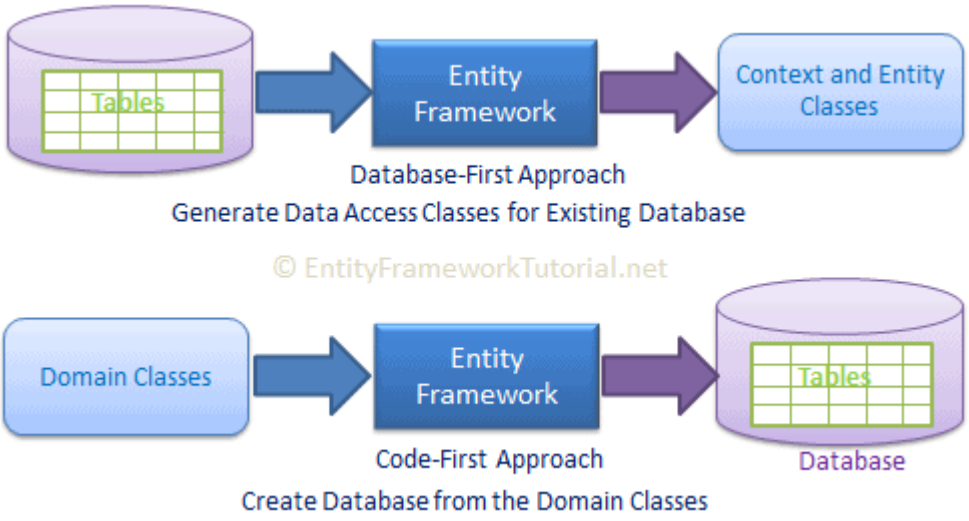

Рисунок 3.7 — Модель використання Entity Framework Core

Модель даних сутності визначає концептуальну модель даних, використовуючи техніку моделювання, яка сама називається Entity Data Model, розширеною версією моделі сутність-взаємозв'язок.

3.7. Технологія використання парсингу

Розроблена система повинна брати інформацію з Google Scholar. Є різні варіанти вирішення цієї задачі, але для себе я вирішив використовувати технологію парсингу.

Парсинг — це автоматичний збір загальнодоступної інформації з Інтернету, який проводиться без використання сайтів API. Ви можете собі уявити, як людина відкриває браузер, переглядає веб-сайти та копіює з них дані. Аналіз аналогічний, тільки це не людина, що йде, а робот. Цим займаються пошукові системи, агрегатори, скорингові компанії, що продають персональні дані та багато інших.

Парсинг — це метод, при якому рядок даних перетворюється на дані іншого типу. Скажімо, ви отримуєте свої дані у необробленому HTML, синтаксичний аналізатор візьме цей HTML і перетворить його в більш читабельний формат даних, який можна легко прочитати та зрозуміти.

Добре зроблений синтаксичний аналізатор визначить, яка інформація потрібна із рядка HTML, і згідно з правилами та заздалегідь написаним кодом парсерів він відбере необхідну інформацію та перетворить її, наприклад, у JSON, CSV або таблицю. Схема роботи парсера наведена нижче (рисунок 3.8).

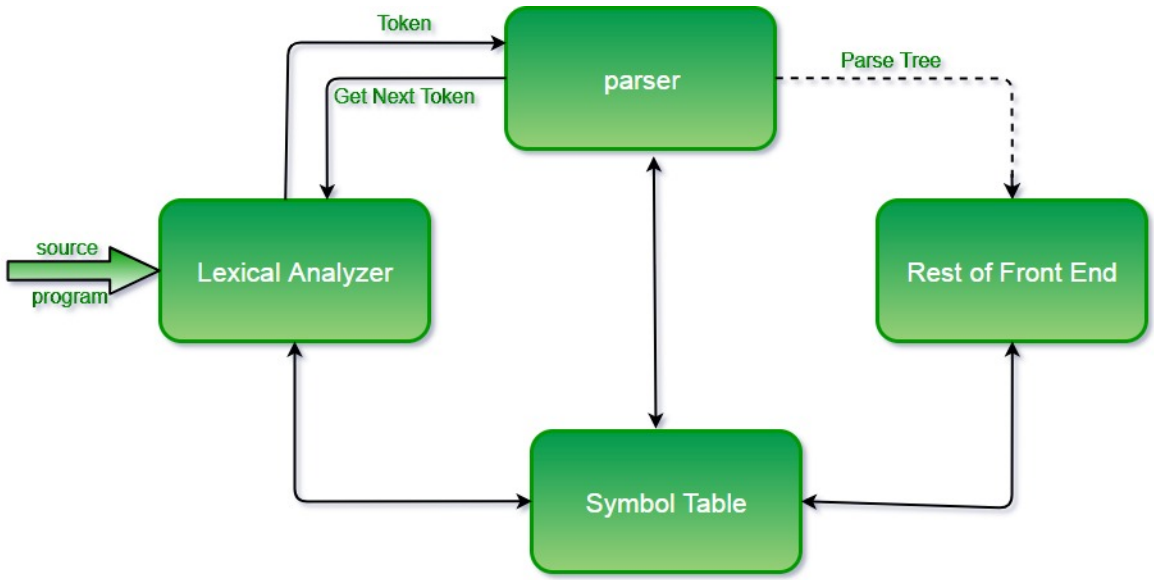

Рисунок 3.8 — Схема роботи парсера

Важливо згадати, що сам парсер не прив'язаний до формату даних. Це інструмент, який перетворює один формат даних в інший, як він перетворює його, і що залежить від того, як був побудований парсер.

Насправді автоматизований збір інформації називається збором, а синтаксичний розбір — це етап збору, на якому необхідна інформація витягується із завантажених даних.

3.8. Мова програмування JavaScript, HTML та CSS

Так як результатом створення автоматизованої системи для збору наукометричних даних є веб-застосунок, в ході розробки були використані такі засоби як HTML, CSS та мова програмування JavaScript. Нижче схематично зображені основні функції HTML, CSS та JS (рисунок 3.9).

Рисунок 3.9 — Основні функції HTML, CSS та JS

JavaScript — це текстова мова програмування, що використовується як на стороні клієнта, так і на стороні сервера, що дозволяє зробити веб-сторінки інтерактивними. Там, де HTML і CSS — це мови, які надають структуру та стиль веб-сторінкам, JavaScript надає веб-сторінкам інтерактивні елементи, які залучають користувача.

HTML — це абревіатура, що розшифровується як Hyper Text Markup Language, яка використовується для створення веб-сторінок та веб-додатків.

HTML складається з серії коротких кодів, набраних текстовим файлом автором сайту — це теги. Потім текст зберігається у форматі html і переглядається через браузер, наприклад Internet Explorer.

CSS розшифровується як каскадні таблиці стилів. Саме мова кодування надає веб-сайту зовнішній вигляд та макет. Поряд з HTML, CSS є фундаментальним для веб-дизайну. Без цього веб-сайти все одно були б звичайним текстом на білому тлі.

CSS дозволяє розробнику:

- Вказувати нестандартні шрифти;
- Вказувати колір і розмір тексту та посилань;
- Застосовувати кольори до фонів, блоків тощо;
- Розміщувати елементи веб-сторінки у вікна та переміщувати ці вікна до певних позицій на сторінці.

3.9. Система контролю версій Git

Під час розробки автоматизованої системи, я використовував систему контролю версій Git. Ця технологія створення програмного забезпечення надає розробнику можливість легше та швидше писати код, модифікувати його та зберігати різні версії проекту (рисунок 3.10).

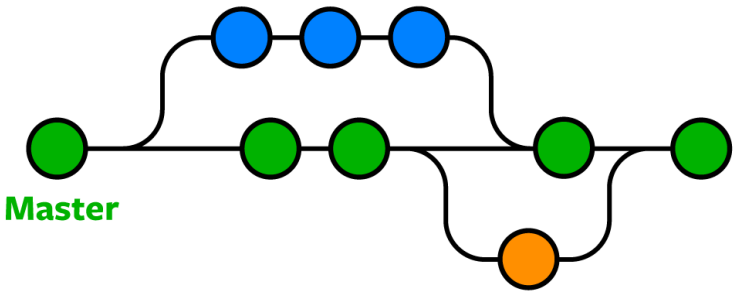

Рисунок 3.10 — Приклад створення гілки в Git

Маючи розподілену архітектуру, Git є прикладом розподіленої системи контролю версій. Замість того, щоб мати єдине місце для повної історії версій програмного забезпечення, як це поширено в колись популярних системах контролю версій, таких як CVS або Subversion, у Git робоча копія коду кожного розробника також є сховищем. який може містити повну історію всіх змін.

На відміну від деяких програм контролю версій, Git не обмінюється іменами файлів при визначенні того, яким має бути сховище та історія версій дерева файлів, натомість Git фокусується на самому вмісті файлу. Зрештою, файли вихідного коду часто перейменовуються, розділяються та переставляються.

3.10. Висновок до розділу

У даному розділі були описані методи та засоби, які були використані при створенні веб-застосунку. Були описані способи їх використання, плюси та мінуси.

Для розробки автоматизованої системи було обрано мову C# через те, що вона сучасна та підходить для створення швидких та структурно правильних додатків. Мова надійна та безпечна у використанні через те, що вона об’єктно-орієнтована, також вона високо продуктивна.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Розроблений програмний продукт дозволяє користувачу збирати та завантажувати наукометричні дані з інтернет ресурсів, їх. Після завантаження та аналізу даних, користувач матиме можливість переглядати ці дані на веб-сторінках та взаємодіяти з ними.

4.1. Реалізація парсера на основі бібліотеки AngleSharp

При розробці веб-застосунку основною бібліотекою для збору даних з інтернету було обрано бібліотеку AngleSharp.

AngleSharp — це відкрита бібліотека .NET, яка надає можливість синтаксичного аналізу гіпертекстів на основі кутових дужок, таких як HTML, SVG та MathML. Важливим аспектом саме AngleSharp є те, що CSS також може бути проаналізований та використаний у подальшому користувачем. Внутрішній синтаксичний аналізатор побудований за офіційною специфікацією W3C, що означає правильний розподіл та аналіз.

На основі цієї бібліотеки був створений парсер наукометричних даних ArticleParser. Він побудований таки чином, що його функції можна розширювати у майбутньому. Таким чином буде збільшуватись кількість цільових веб-сайтів, з яких збиратимуться дані. Також модуль легко можна буде замінити, на якийсь інший, якщо цього потребуватиме веб-застосунок, цього вийшло досягнути за допомогою використання інтерфейсів та абстракцій, які знищують сильну зв’язку та надають більшої гнучкості у подальшій розробці. Принцип роботи парсера зображено нижче (рисунок 4.1).

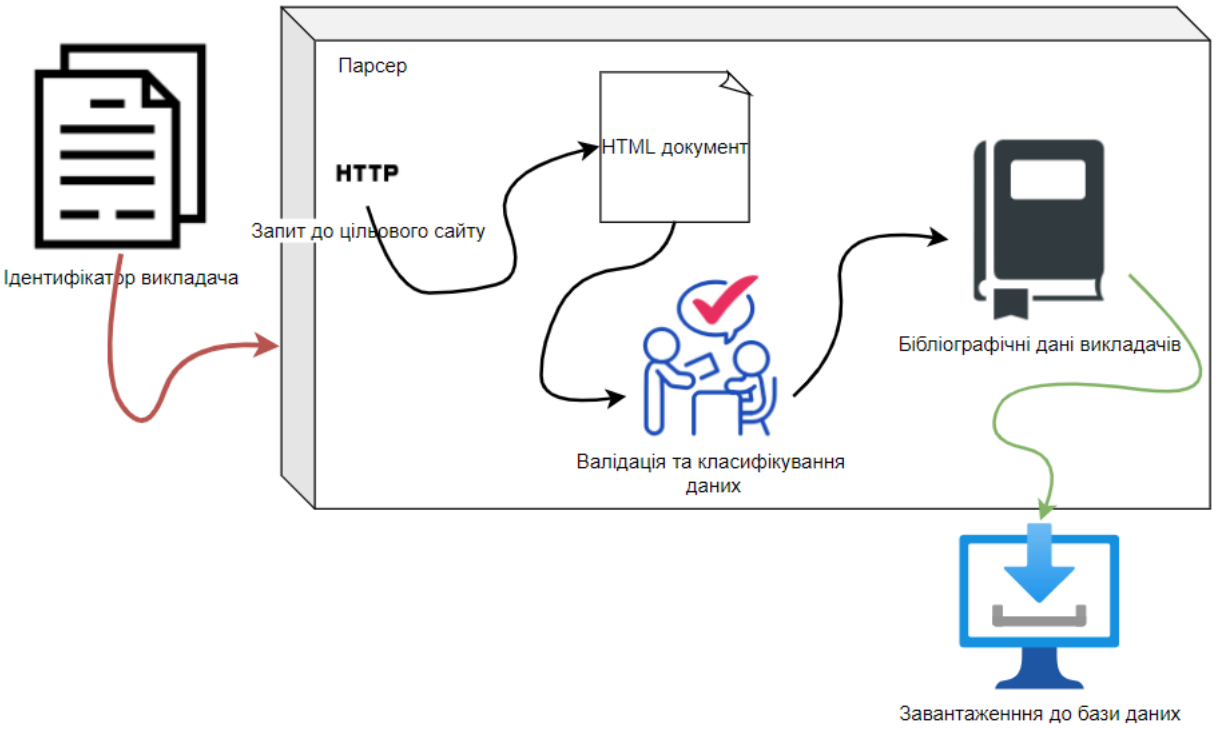

Рисунок 4.1 — Схема роботи парсера

Загалом парсер працює таким чином, що на вхід він отримує ідентифікатор науковця. Після цього відбувається запит до цільового сайту, який надає HTML документ. Система його валідує та класифікує відповідно до класів моделей. Також відбувається валідація дублікатів даних. Фінальним етапом роботи є створення колекції бібліографічних даних, яка завантажується до бази даних.

4.2. Реалізація доступу до бази даних

При розробці веб-застосунку, було розроблено модуль DAL (Data access layer), який відповідає за доступ до бази даних, роботою з нею, редагування. Такий спосіб розробки досить поширений у наші дні, так як з такою структурою модуля його можна легко розширювати. При розробці цього модуля було використано патерн Unit of Work.

Unit of Work називається єдиною транзакцією, яка включає кілька операцій вставки, оновлення, видалення тощо. Якщо сказати це простими словами, це означає,

що для певної дії користувача (скажімо, реєстрація на веб-сайті) всі транзакції, такі як вставка, оновлення, видалення тощо, виконуються в одній транзакції, а не виконуються кілька транзакцій бази даних. Це означає, що одиниця роботи передбачає різні операції в одній транзакції.

Його особливість полягає в тому, що одиниці шаблонів роботи призначені для створення рівня абстракції для доступу до даних та рівнем бізнес-логіки програми. Завдяки цим шаблонам системи залишиться ізольованою та матиме захист від змін у сховищі даних. Такий спосіб роботи з даними дає можливість полегшити розробку проекту у майбутньому, модульне тестування або ж кероване тестування (рисунок 4.2).

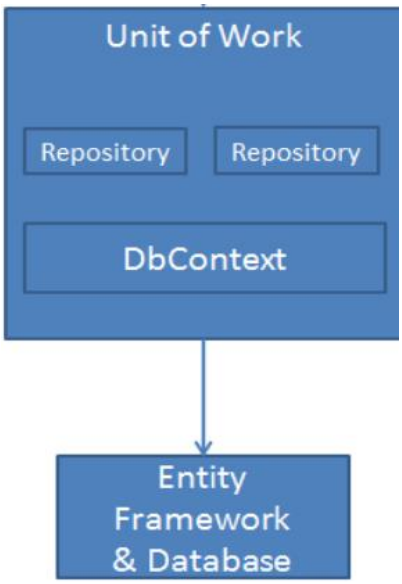

Рисунок 4.2 — Модель використання шаблону проектування Unit of Work

Під час розробки даного модуля також було використано вбудовану бібліотеку LINQ для роботи з колекціями даних. LINQ — це єдиний синтаксис запитів у C#, який дозволяє отримувати дані з різних джерел та форматів.

Запити LINQ повертають результати у вигляді об’єктів. Це дозволяє використовувати об'єктно-орієнтований підхід до набору результатів і не турбуватися про перетворення різних форматів результатів у об'єкти (рисунок 4.3).

Рисунок 4.3 — Схема за якою відбувається запит LINQ.

Таким чином, кожен запит LINQ повинен запитувати до певного типу джерел даних, чи це може бути масив, колекції, XML чи інші бази даних. Після написання запиту LINQ його потрібно виконати, щоб отримати результат.

4.3. Структура проекту

Розроблений проект складається з трьох модулів: ArticlesParser, DAL, WebDiploma, що дозволяє розмежувати залежності між частинами веб-застосунку. ArticleParser — відповідає за синтаксичний аналіз цільових сайтів та збір даних з них. DAL — відповідає за доступ до бази даних. Про ці 2 модуля описано в пунках вище.

WebDiploma — це головна частина всього веб-застосунку. У ньому зосереджене сам веб-сайт, використання бази даних і контролери, які викликають збір наукометричних даних з модуля ArticleParser. При написанні цього модуля було використано архітектурний патерн MVC, про який було описано вище.

Ця частина система займається тим, що відповідає за запити браузера на показ сторінок, надсилання та витягування даних з бази даних (використовуючи DAL) та надсилання запитів на парсинг даних (використовуючи ArticleParser). Додаток делегує запити, які він отримав від браузера, іншим частинам проекту.

Нижче зображена схема, яка описує структуру проекту (рисунок 4.4).

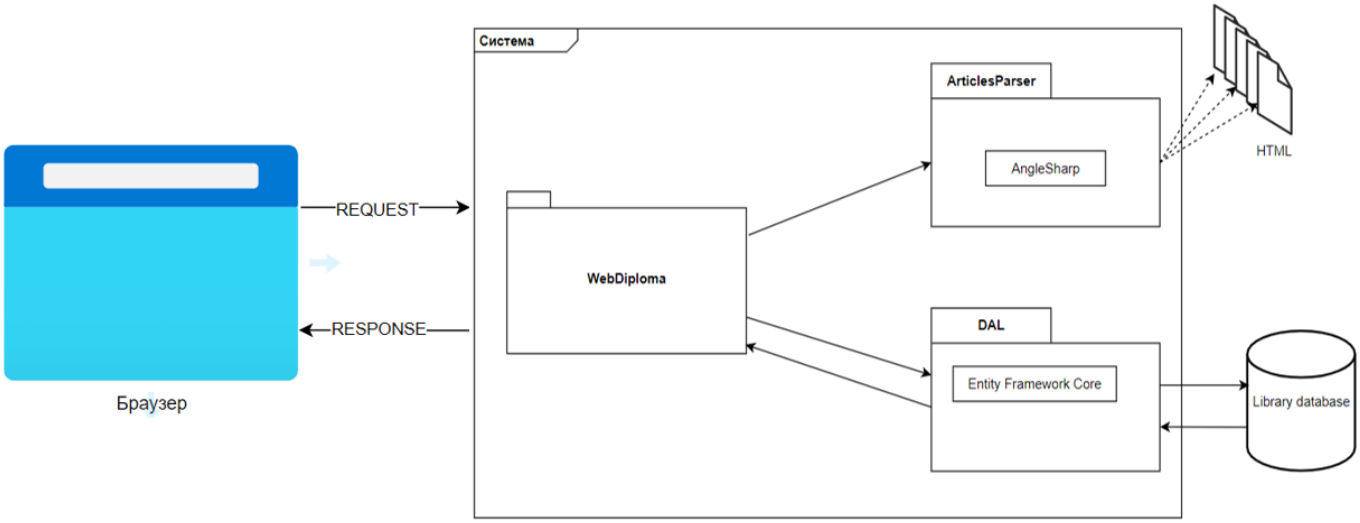

Рисунок 4.4 — Структура проекту

Додаток працює таким чином, що віддалений сервер отримує запит від клієнта-браузера. Опрацьовує його за допомогою ArticlesParser, який виконує пошук даних, після цього виконується збереження в базу даних і після цього веб-сервер надає відповідь браузеру.

4.4. Діаграма прецедентів

При створенні автоматизованої системи, було також розроблено діаграму прецедентів, яка застосовується для моделювання вигляду розроблюваної системи з точки зору користувача.

Діаграма прецедентів — в графічній мові зображення UML, це діаграма, яка відображує взаємодію акторів з прецедентами в створюваній системі. Цю діаграму також можна називати діаграмою використання системи.

Розроблена діаграма є інструкцією з використання системи, яка складається з деяких акторів-виконавців, прецедентів-подій обмежених границею системи, різних асоціацій між ними, взаємозв’язку між акторами. Нижче наведена діаграма прецедентів автоматизованії системи (рисунок 4.5).

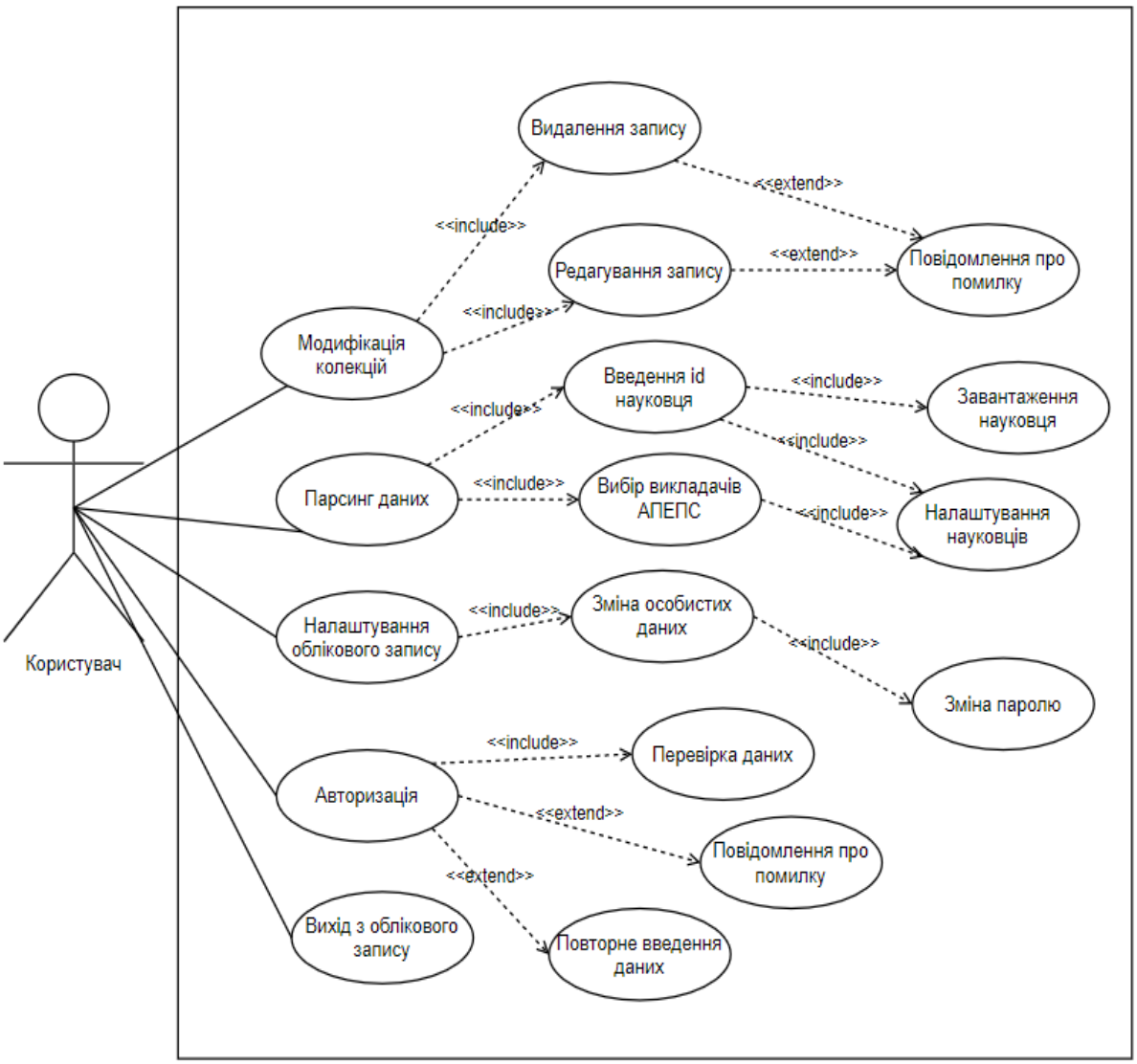

Рисунок 4.5 — Діаграма прецедентів

Головною ідеєю створення цієї діаграми є те, що розроблювана система може представлятися у вигляді великої кількості акторів, що саме мають взаємодію з системою за допомогою описаних в ній прецедентів. Кожен можливий варіант використання системи повинен визначати набір подій, які система повинна виконати при взаємодії з актором.

Для опису взаємодії акторів та прецедентів на діаграмі відображаються відносини.

Зазвичай виділяють 4 типи відносин між подіями та акторами:

- асоціації (association);
- включення (include);

- розширення (extend);
- узагальнення (generalization).

Головним актором в автоматизованій системі є Користувач. Він має можливість авторизуватись та вийти з облікового запису. Після входу в систему користувачу може: налаштувати обліковий запис, змінити пароль для авторизації, автоматизовано збирати дані, обираючи потрібний йому варіант парсингу, редагувати та видаляти записи з бази даних.

4.5. Архітектура класів

Автоматизована система збору даних складається з деякої кількості класів. Разом ці класи мають архітектуру, яка забезпечує правильну обробку даних, безпечну заміну чи розширення системи у майбутньому. При розробці велику роль має саме структура класів і їх відносини.

При розробці частини, яка займається збором наукометричних даних була використана інтерфейсна технологія створення бібліотек. Основною ідеєю такої розробки є те, що є набір функціоналу описаного в інтерфейсі і є декілька класів, які реалізують цей функціонал.

Інтерфейс — це засіб подолати відсутність кількох спадкоємств у C #, тобто ви не можете успадковувати з декількох класів, але можете реалізувати кілька інтерфейсів. ООП намагається нагадувати те, як об’єкти визначаються в реальному житті, а інтерфейси — це дуже логічний спосіб групування об’єктів з точки зору поведінки.

Інтерфейс — це контракт між собою та будь-яким класом, який його реалізує. У цьому контракті зазначено, що будь-який клас, який реалізує інтерфейс, буде реалізовувати властивості, методи та / або події інтерфейсу. Інтерфейс не містить реалізації, лише підписи функціональних можливостей, які надає інтерфейс. Інтерфейс може містити сигнатури методів, властивостей, індексаторів та подій.

Нижче зображено структуру бібліотеки, яка займається збором (рисунок 4.6).

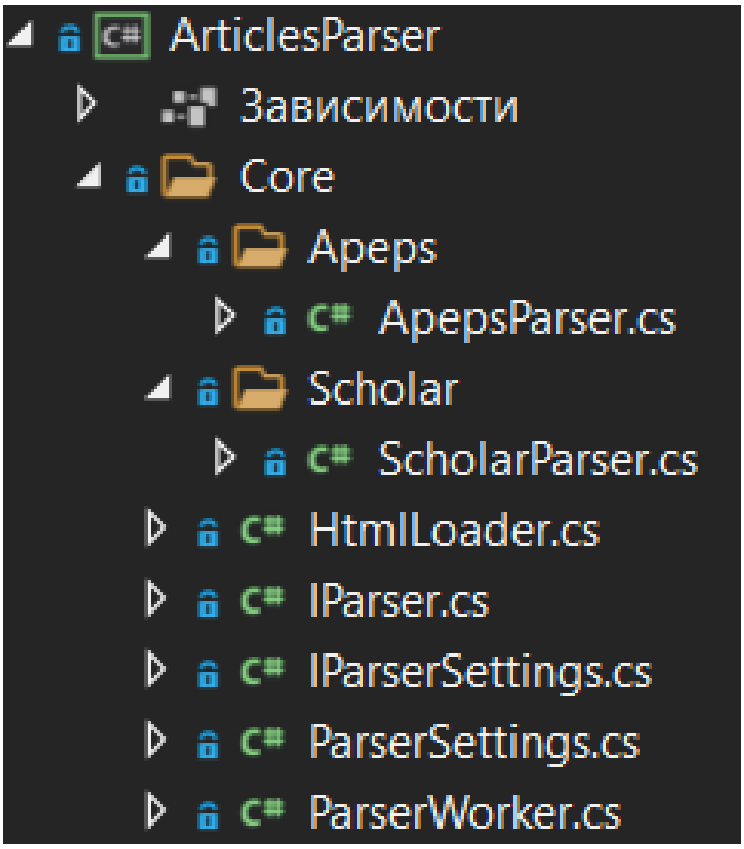

Рисунок 4.6 — Структура автоматизованої системи збору даних

Основні класи:

— Інтерфейс IParser — контракт, який має в собі 2 методи для реалізації ParseTeacher() та ParseArticle(). Перший займається збором інформації про науковця чи викладача. Другий займається пошуком та збором даних про статті науковців;

— Інтерфейс IParseSettings — контракт, який має перелік полів та методів, завдання яких є налаштування парсера;

— Клас ParseSettings — клас, який реалізує інтерфейс IParseSettings, використовується при налаштування основного парсера та при утворенні запиту до Google Sholar;

— Клас ApepsParser — клас, який на вхід приймає DOM файл, потім розбирає його та отримує лише важливу інформацію з усієї веб-сторінки відносно викладачів кафедри АПЕПС, яких потрібно додати до бази даних;

— Клас ScholarParser — клас, який на вхід приймає DOM файл, потім розбирає його та отримує лише важливу інформацію про викладачів, статті, наукові напрямки з усієї веб-сторінки та повертає колекцію даних веб-застосонку;
— Клас ParseWorker — клас, який використовує класи ApepsParser та ScholarParser. Він є головним в цьому ланцюзі, адже саме він приймає запити від веб-застосунку та делегує пошук відповідним парсерам.

Отож розробка на основі інтерфейсу може значно полегшити життя розробника, а наші програми — набагато чистіші та розширювані.

4.6. Опис структури бази даних

Розроблений веб-застосунок використовує реляційну базу даних. Таблиці даних, що використовуються в реляційній базі даних, зберігають інформацію про пов'язані об'єкти. Кожен рядок містить запис з унікальним ідентифікатором — відомий як ключ — і кожен стовпець містить атрибути даних. Кожен запис присвоює значення кожній ознаці, завдяки чому зв’язки між точками даних легко ідентифікувати. Стандартним інтерфейсом користувача та прикладних програм реляційної бази даних є мова структурованих запитів Transact-SQL.

Одним з ключових етапів проектування бази даних є розробка її концептуальної моделі.

На концептуальному рівні відбувається інтегрований опис предметної області, для якої саме ведеться розробка бази даних. Побудова концептуальної моделі є обов’язковим етапом під час створення бази даних, саме цей етап дозволяє однозначно окреслити межі предметної області, виділити її основні сутності, взаємозв’язки між ними. Концептуальна модель бази даних, яка використовувалася для збереження наукометричних даних наведена нижче (рисунок 4.7)

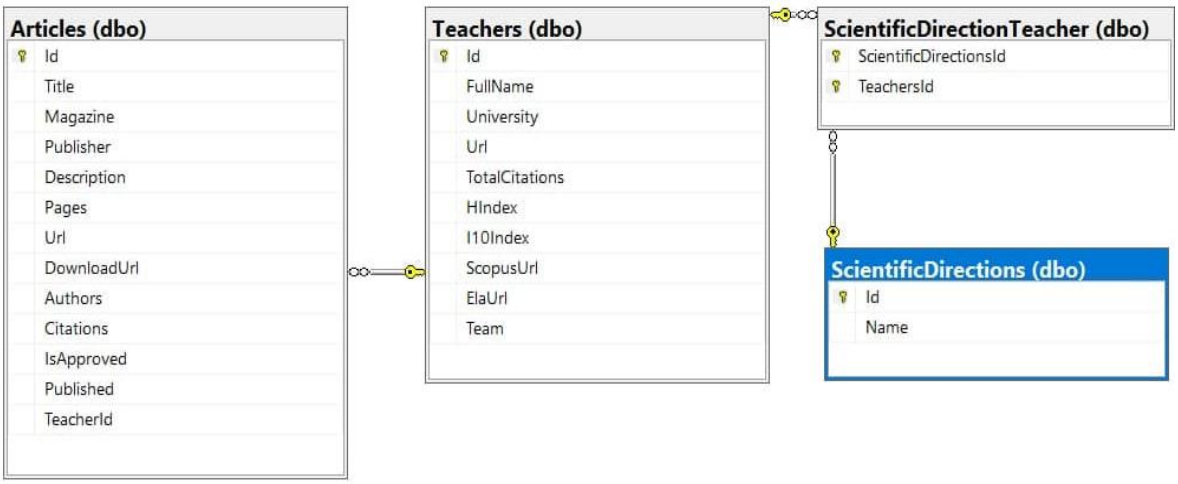

Рисунок 4.7 — Концептуальна модель

База даних складається з 4 таблиць:

— Статті: зберігаються статті з такими полями: ідентифікатор, назва, журнал видавець, опис, сторінки, посилання на оригінал, посилання на скачування, автори, цитування, дата публікації;

— Наукові напрями: зберігаються назви наукових напрямів, за якими працюють науковці;

— Науковці: зберігаються зібрані дані про науковців з такими полями: ідентифікатор, повне ім’я, місце викладання, посилання на оригінальний обліковий запис, загальні цитування, 2 індекси та команда, в якій він працює.

— Науковці-Наукові напрями: таблиця, яка має зв’язок багато до багатьох, використовується для зв’язування таблиць Науковців та Напрямів.

4.7. Висновок до розділу

У даному розділі були описані всі програмні реалізації використані під час розробки автоматизованої системи, веб-додатку. Було наведено основну архітектуру застосунку, архітектуру класів, діаграму прецедентів та повністю описана структура бази даних, яка використовується для зберігання наукометричних даних.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для забезпечення правильної роботи розробленої системи автоматизованого збору наукометричних даних бажано дотримуватися наведених нижче вимог при інсталяції веб-застосунку та рекомендацій щодо її використання.

5.1. Системні вимоги та інсталяція

Для запуску створеного веб-застосунку необхідна наявність встановлених пакетів платформи .NET 5, Windows Server 2019 року та можливість роботи з T-SQL. Також програмний продукт можна поставити на віддалений хостинг.

Для коректної роботи автоматизованої системи необхідне дотримання наступних мінімальних вимог до апаратного забезпечення:

— процесор Intel Core i3 2.1 GHz;

— оперативна пам’ять мінімальним об’ємом 4 Гб;

— відео пам’ять Intel Graphics 620.

Після встановлення та запуску системи, буде відкритий веб-сайт на localhost. Відкрити його можна буде в будь-якому браузері.

У випадку, якщо веб-застосунок буде поставлений на віддалений хостинг, веб-сайт можна буде відкривати на будь-якому пристрої з доступом до мережі Інтернет. Веб-браузер повинен підтримувати виконувати скрипти JS та web-cookie.

5.2. Використання розробленої системи

Після установки веб-застосунку користувач переходить на веб-сайт. Користувач з самого початку матиме можливість переглядати такі сторінки:

1. Статті — це веб-сторінка, на якій користувач може переглядати вже зібрані статті, сортувати їх, фільтрувати вибірку (рисунок 5.1). Якщо якась зі статей

зацікавила користувача, потрібно натиснути на назву статті і відкриється вікно з повною інформацію (рисунок 5.2), також можна перейти за посиланням на оригінальну сторінку, де зберігається стаття.

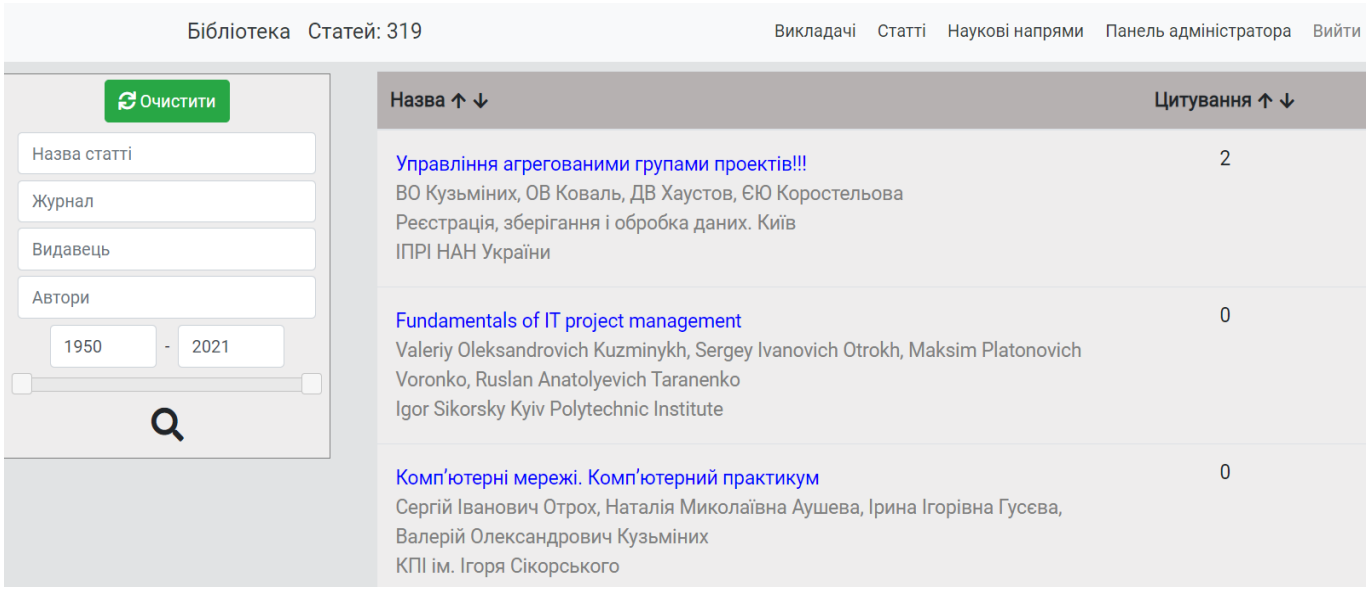

Рисунок 5.1 — Веб-сторінка Статті

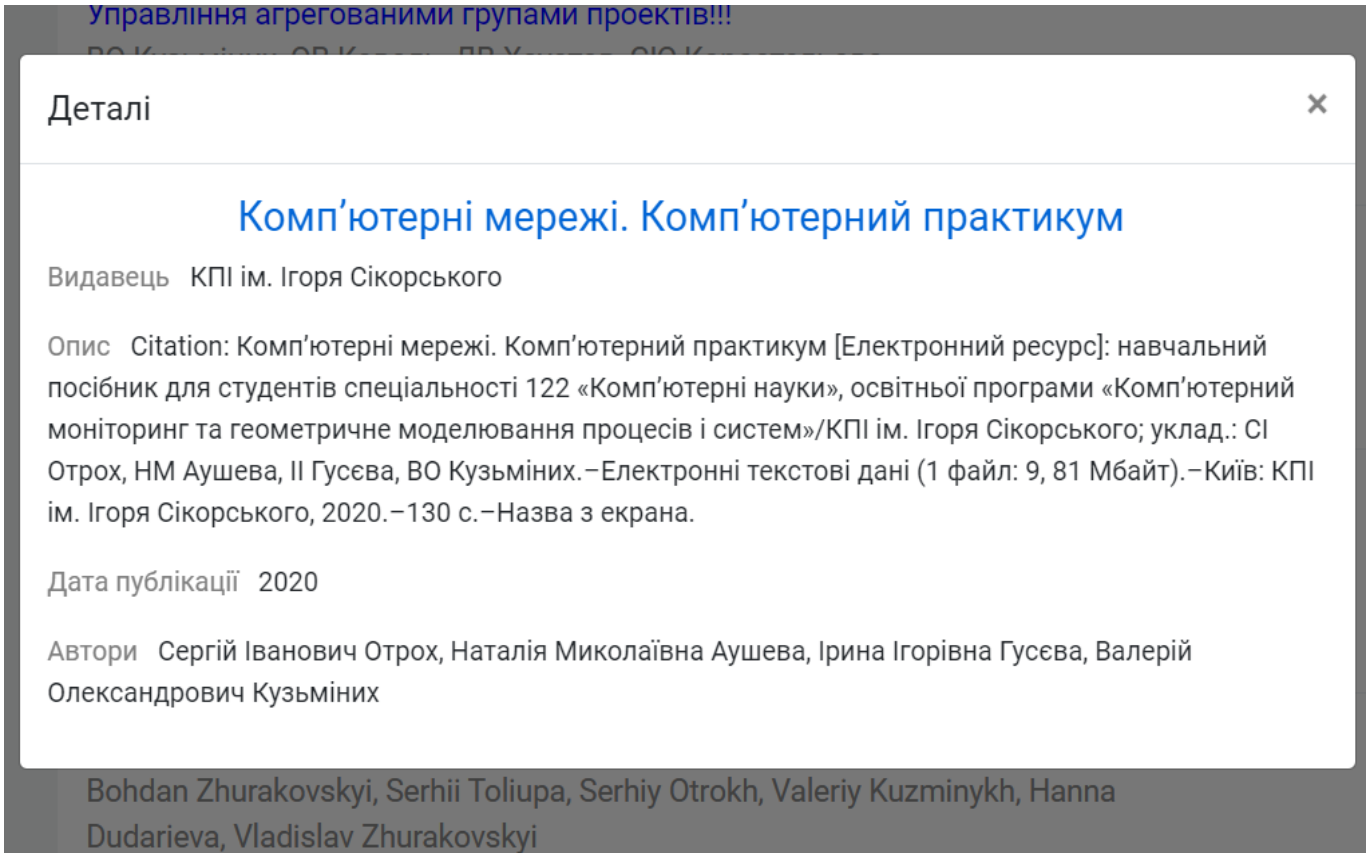

Рисунок 5.2 — Вигляд конкретно вибраної статті

2. Викладачі — це веб-сторінка, на якій користувач може переглядати список всіх викладачів та науковців, які зібрані в базі даних (рисунок 5.3), натиснувши на вибраного викладача, користувач потрапляє на його сторінку (рисунок 5.4), де є повна інформація про науковця: різні індекси, місце викладання, наукові напрямки і всі його статті, які також можна переглядати.

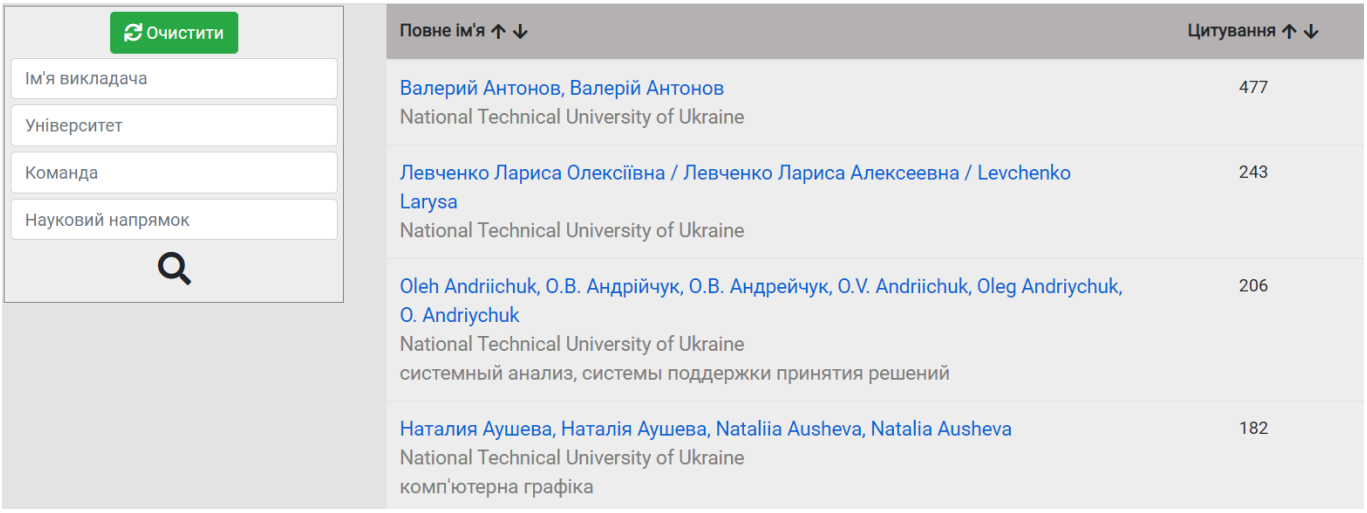

Рисунок 5.3 — Веб-сторінка Викладачі

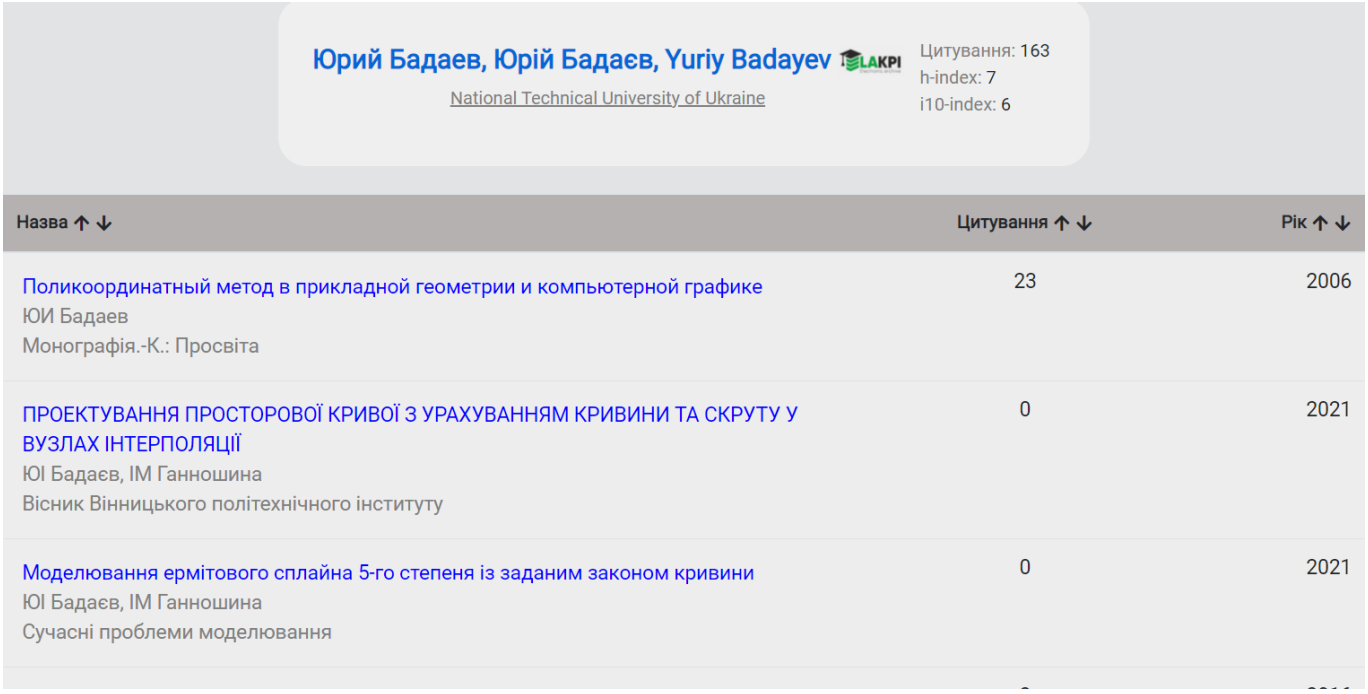

Рисунок 5.4 — Сторінка викладача

3. Наукові напрямки — це веб-сторінка, на якій користувач може переглядати наукові напрями, з якими працюють викладачі (рисунок 5.5) та переходити на веб-сторінки конкретних напрямів (рисунок 5.6). Також можна за допомогою пошуку знаходити викладачів за конкретним напрямом.

Рисунок 5.5 — Веб-сторінка наукових напрямів

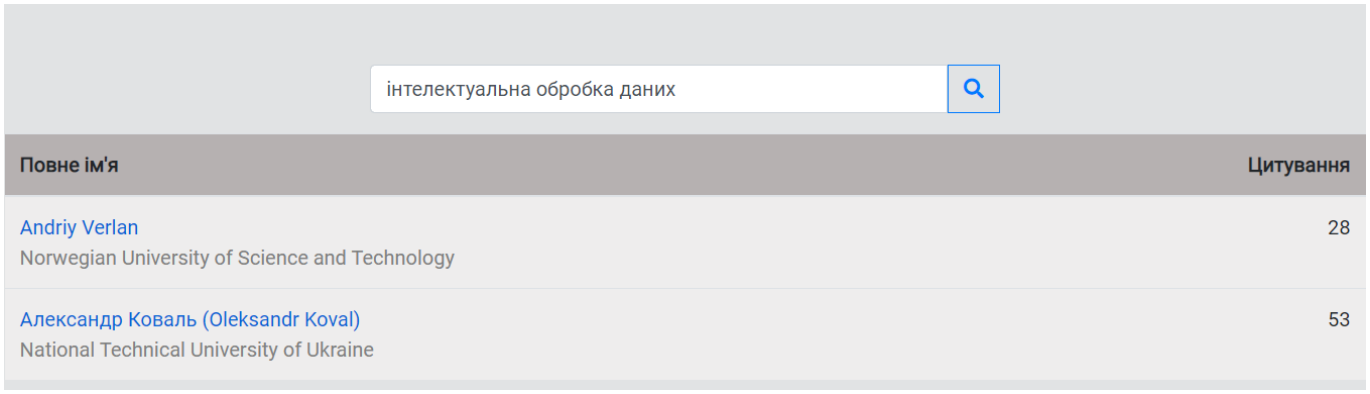

Рисунок 5.6 — Вибірка викладачів за конкретним науковим напрямом

Для того, щоб він зміг працювати з розробленою системою автоматичного збору наукометричних даних, потрібно перейти на сторінку “Увійти” (рисунок 5.7) та авторизуватися.

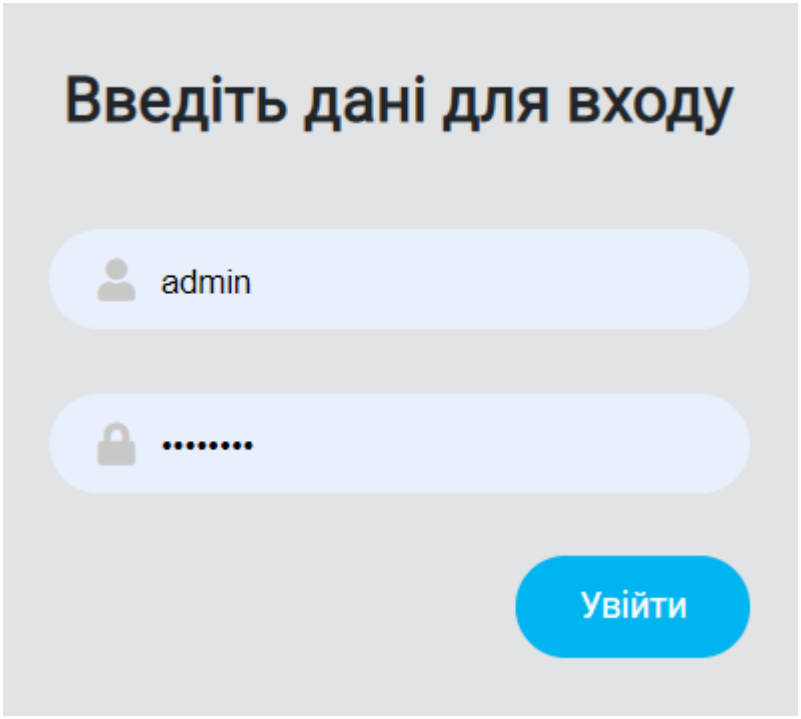

Рисунок 5.7 — Сторінка авторизації

Якщо користувач уведе невірні дані, він отримає повідомлення про це. Після цього він знову матиме змогу спробувати авторизуватися, допоки вхідні дані не будуть вірні.

Інакше, користувач успішно авторизується. Після цього він має перейти в панель адміністратора (рисунок 5.8).

Рисунок 5.8 — Головна сторінка панелі адміністратора

Коли користувач потрапляє на панель адміністратора, перед ним є можливість редагувати вже доданих викладачів, завантажувати їм статті чи редагувати, налаштовувати їм наукові напрями, за якими науковці викладають.

Якщо користувач входить вперше на панель адміністратора, він може перейти на веб-сторінку “Guid” (рисунок 5.9). На ній описані правила використання панелі та всі можливі функції взаємодії з нею.

Рисунок 5.9 — Веб-сторінка Guid

Щоб перейти безпосередньо до системи, яка займається збором наукометричних даних потрібно натиснути на посилання під назвою “Парсер”, вона знаходиться зліва на веб-сторінці (рисунок 5.10)

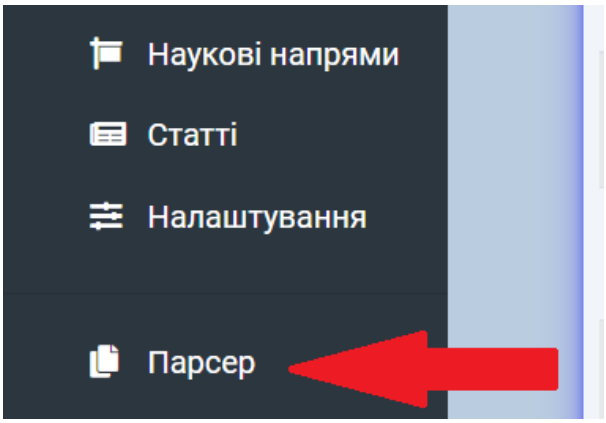

Рисунок 5.10 — Посилання на сторінку парсера наукометричних даних

Коли користувач переходить на сторінку парсера, він повинен у поле (рисунок 5.11) ввести ідентифікатор науковця (рисунок 5.12), його можна знайти на сторінці викладача, якого ви бажаєте додати до бази даних. Після того, як він увів ідентифікатор викладача, потрібно натиснути на кнопку справа або ж на Enter. Відбудеться запит до Google Scholar і після цього викладач буде доданий до нашої бази. На сторінці, що автоматично відкриється, можна додавати статті викладачам, натискаючи на кожну із них (рисунок 5.13). Після натиску на статтю, вона буде завантажена з інтернет ресурсу, показана на екрані на вспиваючому вікні. Якщо користувач вважає якусь статтю непотрібною для додавання до бази, він може просто її видалити, натиснувши на червону кнопку з піктограмою смітничка. Також після додавання статті, користувач може відразу відредагувати її поля та зберегти зміни.

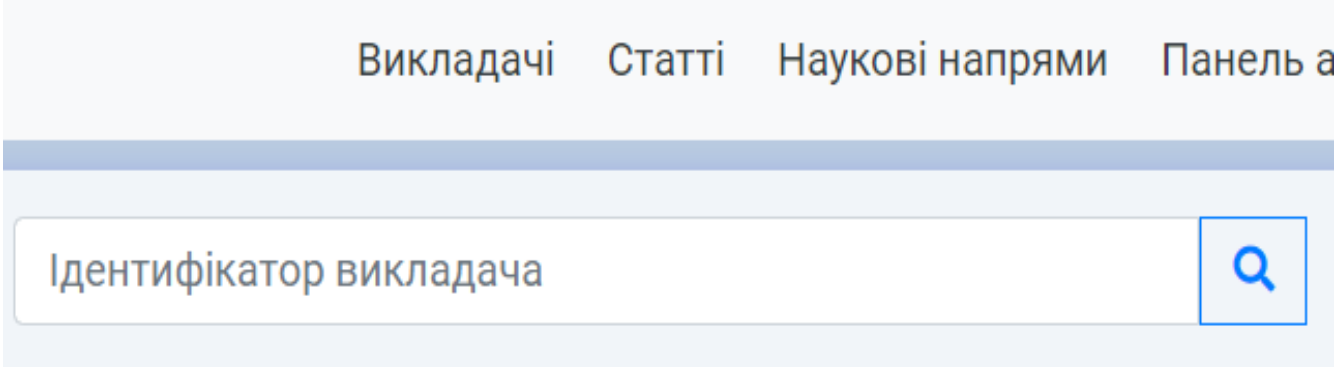

Рисунок 5.11 Поле для вводу ідентифікатора викладача

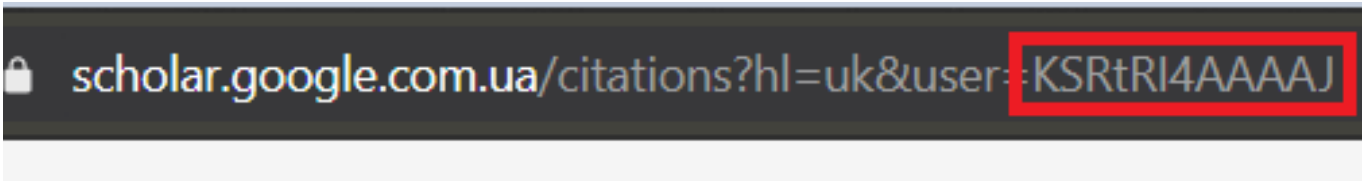

Рисунок 5.12 — Пошукова строка, з якої потрібно брати ідентифікатор

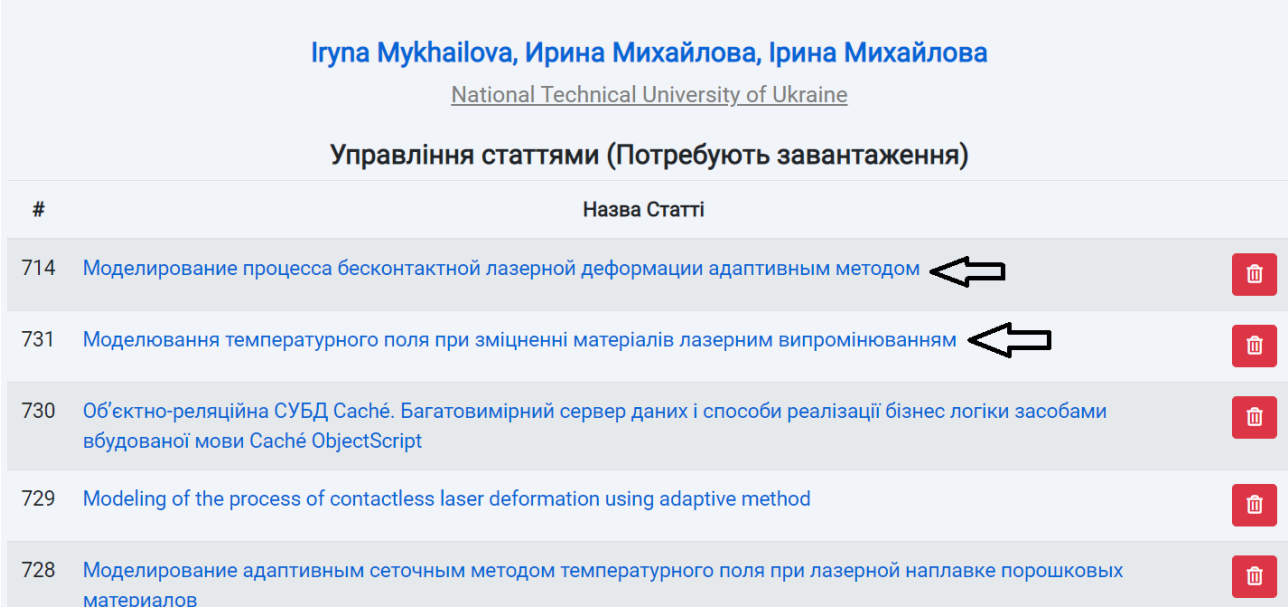

Рисунок 5.13 — Сторінка завантаженого викладача зі статтями, які потрібно завантажити

Також є інший спосіб збирати наукометричні дані, для цього потрібно перейти знову на сторінку парсера, таким чином як це було вище. На сторінці. окрім поля для введення ідентифікатора, також буде таблиця з викладачами кафедри АПЕПС ТЕФ

(рисунок 5.14), яких потрібно завантажити до бази даних. А нижче будуть показані вже завантажені викладачі (рисунок 5.15). Натиснувши на кнопку “Завантажити”, буде відкрита сторінка з цим викладачем, де ви знову ж таки зможете додавати, редагувати та видаляти статті. Натиснувши на кнопку “Детальніше”, буде відкрита сторінка з уже завантаженим раніше викладачем, де ви зможете працювати з його статтями.

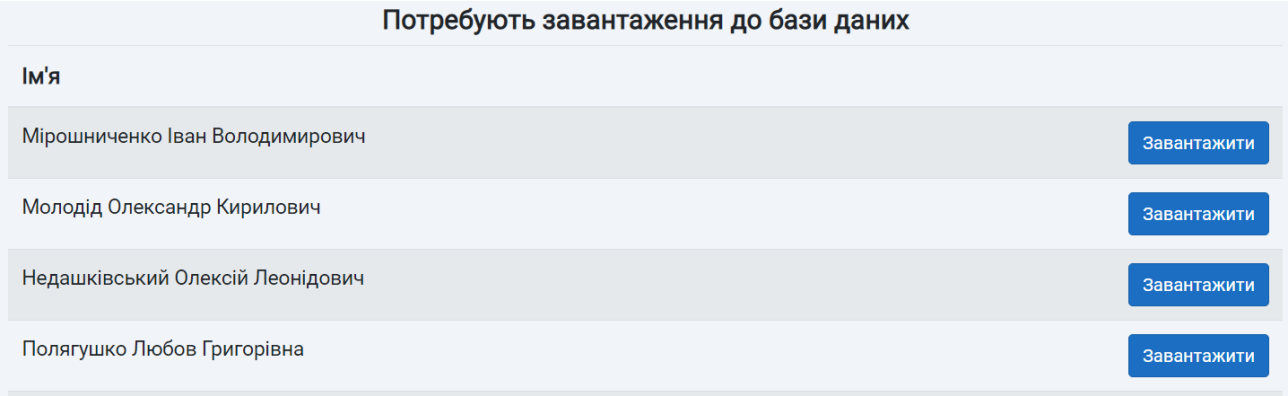

Рисунок 5.14 — Викладачі кафедри АПЕПС, який можна додати до бази даних

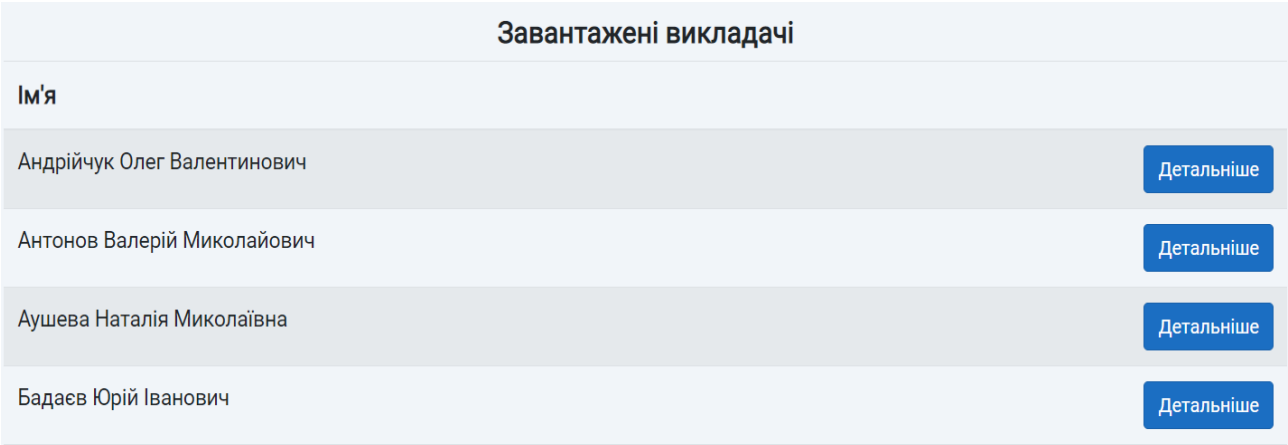

Рисунок 5.15 — Викладачі кафедри АПЕПС, які вже додані до бази даних

Таким чином користувач повинен використуваати автоматизовану систему збору наукометричних даних. Великим плюсом такого способу додавання викладачів та науковців є те, що база даних розрахована не тільки на університетських чи українських викладачів. Цей парсер може додавати будь-яких викладачів з усього світу, потрібно лише знати його особистий ідентифікатор Google Scholar.

5.3 Висновок до розділу

У даному розділі було описано системні мінімальні вимоги до встановлення та роботи веб-застосунку. Також було надано повну інструкцію щодо використання створеного веб-застосунку користувачем.

Основними моментами є:

- Встановлення веб-застосунку відповідно до інструкції;
- Запуск веб-застосунку;
- Авторизація користувача;
- Перехід на панель адміністратора;
- Робота з парсером;
- Завантаження науковців та їх статей.

Також користувач може не тільки займатися збором наукометричних даних.

ВИСНОВКИ

Результатом виконання дипломного проекту розроблено програмний продукт, який забезпечую збір наукометричних даних. Під час створення програми було проаналізовано велику кількість способів отримання даних із інтернет ресурсів та був вибраний для реалізації.

Програмний продукт розроблений об’єктно-орієнтованою мовою програмування C#, програмної платформи .NET 5. В якості середовища розробки використано програмне забезпечення Visual Studio 2019.

Було проведено дослідження вже існуючих систем, які б могли виконувати схожі функції, проаналізовані основні недоліки та запропоновані сучасні інтерфейсні вирішення даної задачі.

Створений програмний продукт дозволяє вибирати користувачеві, які дані завантажувати. Також користувач з легкістю може редагувати їх і, якщо потрібно, видаляти. Система може розширюватися, додаючи інші синтаксичні аналізатора інших інтернет ресурсів на кшталт Google Scholar.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. C# 4.0: полное руководство.: Пер. с англ. — М.: ООО "И.Д. Вильямс", 2011. — 1056 с.

2. Entity Framework [Електронний ресурс] // Документація Microsoft. — 2020. — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Entity_Framework .

3. Використання наукометричних баз та їх інструментарію у наукових дослідженнях — Академія військового мистецтва, Варшава, Польща, 2019. — 122 — 128 с.

4. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В. Стеценко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. — Черкаси : ЧДТУ, 2010. — 399 с.

5. Наукометричні бази даних [Електронний ресурс] // Національна бібліотека України. — 2014. — Режим доступу до ресурсу: http://www.nbuv.gov.ua/node/1367 .

6. Практика синтаксичного аналізу: Пер. с англ — Дік Грюн —Нью-Йорк, 2008 — 662 с.

7. Паттерны для новичков: MVC vs MVP vs MVVM // Посібник— 2014. — Режим доступу до ресурсу: https://habr.com/ru/post/215605/ .

8. Руководство по ASP.NET MVC 5 [Електронний ресурс] // Посібник — 2017. — Режим доступу до ресурсу: https://metanit.com/sharp/mvc5/ .

9. C# 7.0 in a Nutshell: The Definitive Reference // Книга — Джозеф Албані — 2021 — 1007 с.

10. The Definitive ANTLR Reference: Building Domain-Specific Languages // Книга — Терренс Парк — 334 с.

11. Pro ASP.NET MVC 3 Framework Languages // Книга — Адам Фрімен — 124 с.

ДОДАТОК А

Система автоматизованого збору наукометричних даних

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР71235_21Б

Аркушів 2

Київ — 2021

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР71235_21Б 81-1	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР71235_21Б 12-1	SholarParser.cs	Компонент, що реалізовує алгоритм збору наукометричних даних
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР71235_21Б 12-2	ParserController.cs	Компонент збереження зібраних наукометричних даних до бази даних
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР71235_21Б 13-1	ParserController.docx	Опис компонента збереження зібраних наукометричних даних до бази даних

ДОДАТОК Б

Компонент збереження зібраних наукометричних даних до бази даних

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР71235_21Б 12-1

Аркушів 7

Київ — 2021

ScholarParser.cs

```
using AngleSharp.Html.Dom;
using DAL.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using AngleSharp.Dom;

namespace ArticlesParser.Core.Scholar
{
   public class ScholarParser : IParser
   {
      public Teacher ParseTeacher(IHtmlDocument document)
      {
         try
         {
            Teacher teacher = new();

            var fullname = document.QuerySelectorAll("div").Where(div => div.Id != null && div.Id ==
"gsc_prf_in").First();
            if (!string.IsNullOrEmpty(fullname.TextContent))
               teacher.FullName = fullname.TextContent;

            try
            {
               var univercity = document.QuerySelectorAll("a").Where(a => a.ClassList != null &&
a.ClassList.Contains("gsc_prf_ila")).First();
               if (!string.IsNullOrEmpty(univercity.TextContent))
                  teacher.University = univercity.TextContent;
            }
            catch (Exception)
            {
               var univercity = document.QuerySelectorAll("div").Where(div => div.ClassList != null &&
div.ClassList.Contains("gsc_prf_il")).First();
               if (!string.IsNullOrEmpty(univercity.TextContent))
```

```
            teacher.University = univercity.TextContent;
        }


        var scientificDirections = document.QuerySelectorAll("a").Where(a => a.ClassList != null &&
a.ClassList.Contains("gsc_prf_inta"));
        List<ScientificDirection> directions = new();
        foreach (var item in scientificDirections)
        {
            directions.Add(new ScientificDirection() { Name = item.TextContent });
        }
        teacher.ScientificDirections = directions;

        var domArticles = document.QuerySelectorAll("tr").Where(a => a.ClassList != null &&
a.ClassList.Contains("gsc_a_tr"));
        List<Article> articles = new();
        foreach (var item in domArticles)
        {
            var article = new Article()
            {
                DownloadUrl = item.FirstElementChild?.FirstElementChild?.GetAttribute("data-href"),
                Title = item.FirstElementChild?.FirstElementChild?.InnerHtml
            };
    }

    public Article ParseArticle(IHtmlDocument document)
    {
        try
        {
            Article article = new();

            IElement title;
            try
            {
```

```
                title = document.QuerySelectorAll("a").Where(a => a.ClassList != null &&
a.ClassList.Contains("gsc_vcd_title_link"))?.First();
                article.Url = title.GetAttribute("href");
                article.DownloadUrl = document.QuerySelectorAll("div")
                                        .Where(a => a.ClassList != null &&
a.ClassList.Contains("gsc_vcd_title_ggi")).First()
                                        .FirstElementChild?.GetAttribute("href");
            }
            catch (Exception)
            {
                title = document.QuerySelectorAll("div").Where(a => a.Id != null && a.Id ==
"gsc_vcd_title")?.First();
            }

            if (!string.IsNullOrEmpty(title.TextContent))
                article.Title = title.TextContent;

            var items = document.QuerySelectorAll("div").Where(a => a.ClassList != null &&
a.ClassList.Contains("gs_scl"));
        catch (Exception)
        {
            return null;
        }
    }
  }
}
```

ParserController.cs

```
namespace WebDiploma.Areas.Admin.Controllers
{
   [Area("Admin")]
   public class ParserController : Controller
   {
      private readonly ParserWorker _parserWorker;
      private readonly IUnitOfWork _unitOfWork;

      public ParserController(IParser parser, IUnitOfWork unitOfWork)
      {
         _parserWorker = new ParserWorker(parser);
         _unitOfWork = unitOfWork;
      }

      public async Task<IActionResult> Index()
      {
         if (teachersList == null || teachersList.Where(t => t.Id == id).ToList().Count == 0)
         {
            if (Request.Cookies["LastAction"] == null || (DateTime.Now -
(DateTime)(DateTime.Parse(Request.Cookies["LastAction"]))).Minutes > 1)
            {
               if (Request.Cookies["PerMinute"] != null && int.Parse(Request.Cookies["PerMinute"]) ==
10)
               {
                  var lastAction = new CookieOptions
                  {
                     Expires = DateTime.Now.AddMinutes(1)
                  };
                  Response.Cookies.Append("LastAction", DateTime.Now.ToString(), lastAction);
                  Response.Cookies.Delete("PerMinute");
               }
               else if (Request.Cookies["PerMinute"] != null && int.Parse(Request.Cookies["PerMinute"])
< 10)
```

```
            if (model != null)
            {
                model.Id = id;
                model.Url = url;
                model.ScopusUrl = scopusUrl;

                List<ScientificDirection> sDirections = new();
                var existedSDs = await _unitOfWork.ScientificDirectionRepository.GetAll();

                if (existedSDs.Count > 0)
                {
                    model.ScientificDirections.ForEach(item =>
                        sDirections.Add(existedSDs.Any(p => p.Name.ToLower() == item.Name.ToLower())
== true ?
                        existedSDs.First(p => p.Name.ToLower() == item.Name.ToLower()) : item));
                    model.ScientificDirections = sDirections;
                }

                _unitOfWork.TeacherRepository.Add(model);
                await _unitOfWork.SaveChanges();
                model.Articles = model.Articles.OrderByDescending(t => t.Citations).ToList();
            }
        }
        else
        {
            model = (await _unitOfWork.TeacherRepository.GetAll()).First(t => t.Id == id);
            model.Articles = model.Articles.OrderByDescending(t => t.Citations).ToList();
        }

        return View(model);
    }

    public async Task<IActionResult> Article(string href, string teacherId)
    {
```

```
            string url = $"https://scholar.google.com.ua{href}";

            Article model = await _parserWorker.ParseArticle(new ParserSettings(url));
            if (model != null)
            {
                var articlesList = await _unitOfWork.ArticleRepository.GetAll();
                _unitOfWork.ArticleRepository.Update(modelFromRepo);
                var teacher = await _unitOfWork.TeacherRepository.GetById(modelFromRepo.Teacher.Id);

                if (!string.IsNullOrEmpty(teacherId))
                    return Json(new
                    {
                        html = Utility.RenderRazorViewToString(this, "Article", model),
                        result = await _unitOfWork.SaveChanges(),
                        table = Utility.RenderRazorViewToString(this, "UnverifiedArticles", (await
_unitOfWork.ArticleRepository.GetAll())
                        .OrderByDescending(t => t.Citations).Where(p => p.TeacherId == teacherId).ToList())
                    });
                else
                    return Json(new
                    {
                        html = Utility.RenderRazorViewToString(this, "Article", model),
                        result = await _unitOfWork.SaveChanges(),
                        table = Utility.RenderRazorViewToString(this, "AllArticles", (await
_unitOfWork.ArticleRepository.GetAll())
                        .OrderByDescending(t => t.Citations).ToList())
                    });
            }
        }

        return Json(new
        {
            result = false,
            seconds = 60 - (DateTime.Now -
(DateTime)(DateTime.Parse(Request.Cookies["LastAction"]))).Seconds});}}}
```

ДОДАТОК В

Компонент збереження зібраних наукометричних даних до бази даних

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР71235_21Б 13-1

Аркушів 7

Київ — 2021

АНОТАЦІЯ

Додаток містить опис для автоматизованого збору наукометричних даних та збереження їх до реферативної бази даних, що виконує деякі завдання, визначені в розділі 1, а саме:

— Пошук наукометричних даних на цільових сайтах.

— Парсинг та збір даних.

— Збереження та редагування наукометричних даних до реферативної бази даних.

— Фільтрація та сортування збереженої інформації.

Веб-застосунок розроблений мовою програмування C# з використанням технології ASP.NET в інтегрованому середовищі розробки Visual Studio 2019.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ 65
2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ 66
3. ВХІДНІ ДАНІ 67
4. ВИХІДНІ ДАНІ 68

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку міститься опис основних компонентів веб-застосунку для автоматизованого збору наукометричних даних, що виконує задачі, визначенні в розділі 1. Додаток Б містить програмний код цих компонентів.

Для роботи веб-застосунку потрібно мати персональний комп’ютер чи мобільний пристрій та доступ до мережі Інтернет.

Веб-застосунок розроблений мовою програмування C# з використанням технології ASP.NET в інтегрованому середовищі розробки Visual Studio 2019.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний засіб покликаний вирішити задачу збору наукометричних даних, щоб спростити роботу з наповнення реферативної бази даних.

Це було реалізовано за допомогою кількох функцій системи, а саме:

1. Пошук наукометричних даних.
2. Збереження даних про науковців та їх статті.
3. Індексування науковців за їх статтями.
4. Перегляд завантажених даних, редагування та модифікація.

ВХІДНІ ДАНІ

Вхідна інформація для ініціації збору даних:

— Унікальний ідентифікатор науковця, взятий із Google Scholar.

— ПІБ викладача кафедри АПЕПС.

Вхідні дані для аналізу науковця:

— Дані для кількість статей.

— Кількість цитувань кожної зі статей.

— Місце викладання науковця.

ВИХІДНІ ДАНІ

Вихідні дані:

— Відфільтровані статті за цитуванням.

— Загальна статистика всіх статей та науковців.

— Завантажені наукоментричні дані до реферативної бази даних.