

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Розробка системи автоматизованої перевірки
дипломних робіт на основі pdf документів»**

Виконав студент групи ІПЗм-22-2
КУЧЕРУК Ольга Віталіївна

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
 ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
 ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
 КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії

програмного забезпечення

Тетяна ВАКАЛЮК

«18» вересня 2023 р.

ЗАВДАННЯ на кваліфікаційну роботу

Здобувач вищої освіти: **КУЧЕРУК Ольга Віталіївна**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Розробка системи автоматизованої перевірки дипломних робіт на основі pdf документів»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2023 р. № 479/с**

Вихідні дані для роботи: вимоги до оформлення академічних робіт та зразки академічних робіт, методи та алгоритми призначені для перевірки дипломних робіт у форматі PDF, аспекти структуризації вищезгаданих робіт, які можуть бути піддані автоматизованій перевірці, та критерії форматування академічних праць.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1			
2			
3			

РЕФЕРАТ

Кваліфікаційна робота рівня магістр містить аналіз та вибір інструментарію розробки системи автоматизованої перевірки дипломних робіт на основі PDF документів.

Мета роботи полягає в розробці системи автоматизованої перевірки дипломних робіт на основі PDF документів. Результати дослідження надають необхідну базу для розробки сервісу з перевірки дипломних робіт.

Записка складається зі вступу, переліку умовних позначень, трьох розділів, та списку використаної літератури. Записка містить 19 ілюстрацій, 2 таблиці та 43 посилання на використані джерела. Повний обсяг роботи становить 64 сторінки, з них 50 сторінок основного тексту.

В першому розділі було проведено огляд функціоналу систем автоматичної перевірки дипломних робіт, аналіз наявних бібліотек для роботи з файлами PDF, вибір та обґрунтування мови програмування для створення сервісу з перевірки PDF документів та досліджено наявні підходи до аналізу та результату перевірки дипломних робіт.

В другому розділі було описано архітектуру сервісу та реалізацію алгоритмів: перевірки списку використаних джерел, перевірки відповідності змісту сторінкам і заголовкам в роботі на відповідність вимогам оформлення.

В третьому розділі описано тестування інноваційного сервісу перевірки дипломних робіт, проведено оцінку ефективності наукової та технологічної складової та оцінку ефективності сервісу. Оформлення академічних текстів має велике значення з точки зору наукової спільноти, оскільки це не лише сприяє чіткості й зрозумілості викладу ідей, але і підтримує стандарти та вимоги наукової етики. Створення такого сервісу ґрунтується на необхідності підтримувати високі стандарти якості наукових досліджень та забезпечення дотримання встановлених норм.

КЛЮЧОВІ СЛОВА: PFDJS-DIST, ПЕРЕВІРКА PDF ДОКУМЕНТІВ, PDF, АРХІТЕКТУРА, ТЕСТУВАННЯ, TYPESCRIPT, СЕРВІС, АЛГОРИТМ, JS.

ABSTRACT

The Master's degree work includes analysis and selection of tools for the development of an automated verification service based on PDF files.

The aim is to develop an automated system for academic work verification based on PDF documents. Research results have provided us with the necessary base for the automated scientific works verification service's development.

Masters thesis note consists of an introduction, a shortening list, three chapters, and sources. The thesis includes 19 pictures, 2 tables, and 43 sources. The work consists of 64 pages, 50 of which are main text.

The first section reviewed the functionality of the automatic verification of systems, analysed existing libraries for working with PDF files, selected and justified programming languages for creating a service for verifying PDF documents, and explored available approaches to the analysis and the result of verifying the theses.

The second section describes the service's architecture, provides a description and implementation of the algorithms for verifying the correctness of the listing of the sources used and the verification of the conformity of the content to the pages and headers in the work.

The third section describes testing of innovational service for academic works verification. Evaluation of scientific and technological component's effectiveness and evaluation of service efficiency were carried out. The academic texts' clearance is of great importance from the point of view of the scientific community, as it not only promotes clarity and comprehensibility of the presentation of ideas, but also supports scientific ethics' standards and requirements. The creation of such a service is based on the need to maintain high standards of quality of scientific research and ensuring compliance with established norms.

KEYWORDS: PDFJS-DIST, PDF-DOCUMENTS VERITIFICATION, PDF, ARCHITECHTURE, TESTING.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОБОТИ	10
1.1 Огляд автоматичної перевірки дипломних робіт	10
1.2 Вибір та обґрунтування мови програмування для створення сервісу з перевірки PDF документів	13
1.3 Аналіз існуючих бібліотек для роботи з файлами PDF	16
1.4 Наявні підходи до аналізу та результату перевірки дипломних робіт	24
Висновки до розділу 1	28
РОЗДІЛ 2. РЕАЛІЗАЦІЯ СЕРВІСУ ДЛЯ АВТОМАТИЧНОЇ ПЕРЕВІРКИ ДИПЛОМНИХ РОБІТ	30
2.1 Архітектура сервісу	30
2.2 Опис та реалізація алгоритмів перевірки на правильність оформлення списку використаних джерел.....	35
2.3 Опис та реалізація алгоритмів перевірки на відповідності змісту сторінкам і заголовкам в роботі.....	40
Висновки до розділу 2	42
РОЗДІЛ 3. ЕФЕКТИВНІСТЬ ТА ТЕСТУВАННЯ СЕРВІСУ ДЛЯ ПЕРЕВІРКИ АКАДЕМІЧНИХ РОБІТ	44
3.1 Тестування та налагодження	44
3.2 Оцінка ефективності наукової та технологічної складової	48
3.3 Оцінка ефективності сервісу.....	52
Висновки до розділу 3	55
ВИСНОВКИ.....	56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

JS – JavaScript

MVVM – Model-View-ViewModel

NLP – Natural Language Processing

PDF – Portable Document Format

ROE – Return on Equity

SVG – Scalable Vector Graphics

TS – TypeScript

БАК – вища атестаційна комісія

ОПМ – обробка природної мови

ПЗ – програмне забезпечення

COA – сервісно-орієнтована архітектура

ВСТУП

Актуальність теми: В сучасному освітньому середовищі, де цифрові технології інтенсивно впроваджуються в усі сфери навчання, виникає необхідність вдосконалення засобів перевірки академічних робіт студентів. Зростання обсягів інформації та легкість її поширення призводять до нових викликів у забезпеченні високої якості освіти та уникненні недоліків оформлення академічних записок. У цьому контексті, розробка ефективного сервісу з перевірки академічних робіт у форматі PDF (portable document format) набуває важливого значення. У сучасному світі, де освіта визнається однією з ключових складових успішного розвитку суспільства, актуальність створення сервісу з перевірки академічних робіт у форматі PDF для магістерських записок набуває великого значення. Суттєва частина вищої освіти спрямована на розвиток та поглиблення знань студентів у конкретних галузях, що вимагає подання власних досліджень та аналізу. Завданням сервісу є сприяння цьому процесу, забезпечуючи якісну перевірку та обробку академічних робіт, щоб студенти могли концентруватися на глибині та якості своїх досліджень.

Один із ключових аспектів актуальності даної ініціативи полягає у покращенні якості освіти. Забезпечуючи доступність ефективного інструменту для перевірки магістерських робіт, ми сприяємо виявленню можливих неточностей та недоліків у роботах, що, в свою чергу, визначає шляхи подальшого удосконалення досліджень та розвитку студентів. Сервіс виявиться корисним для викладачів, які матимуть можливість оперативно надавати конструктивний фідбек та спрямовувати зусилля студентів на вдосконалення тих аспектів, які потребують уваги.

Створення сервісу для перевірки академічних робіт підкреслить важливість чесності та відкритості в навчальному процесі. Студенти матимуть можливість переконатися у справедливості та об'єктивності оцінювання своїх робіт, що сприятиме підвищенню морального та етичного рівня в університетському середовищі.

Питанням розробки мікросервісної архітектури застосунків набуло чималої уваги у 21 столітті. Для аналізу та розробки нових сервісів та програмних забезпечень використовуються праці та наукові джерела іноземних та вітчизняних вчених, зокрема А. Вергун, Л.Савенкова, С.Чуканова [1], О.Гайтан, А.Горошко [2], Г.Мур [3], тощо.

Отже, розробка сервісу автоматизованої перевірки документів на виявлення помилок має велику актуальність в сучасному світі, де якість і відповідність академічним стандартам відіграють важливу роль у різних сферах діяльності.

Мета роботи полягає в розробці системи автоматизованої перевірки дипломних робіт на основі PDF документів.

З метою досягнення поставленої задачі були сформульовані наступні **завдання**:

- аналіз існуючих бібліотек для роботи з файлами PDF;
- вибір та обґрунтування мови програмування для створення сервісу з перевірки PDF документів;
- архітектура сервісу;
- аналіз наявних підходів до аналізу та результату перевірки дипломних робіт;
- опис вимог до компонентів сервісу та реалізація алгоритмів перевірки на правильність оформлення списку використаних джерел та перевірки на відповідності змісту сторінкам і заголовкам в роботі;
- тестування та налагодження сервісу;
- оцінка ефективності наукової та технологічної складової.

Об'єктом дослідження є вимоги до оформлення академічних робіт та зразки академічних робіт.

Предметом дослідження виступають методи та алгоритми призначені для перевірки дипломних робіт у форматі PDF, аспекти структуризації

вищезгаданих робіт, які можуть бути піддані автоматизованій перевірці, та критерії форматування академічних праць.

Наукова новизна даної роботи полягає в розробці інноваційного сервісу перевірки академічних робіт студентів у форматі PDF на відповідність вимог оформлення дипломних записок. Використання розробленого сервісу дозволить автоматизувати процес перевірки наукових робіт, покращити їх якість та сприяти подальшому науковому розвитку. Наукова новизна даної роботи ще більше підкреслюється тим, що на момент виконання подібний інноваційний сервіс для перевірки академічних робіт студентів у форматі PDF не існує у світі.

Практична цінність полягає в наданні користувачам зручного та ефективного інструменту для автоматичної перевірки дипломних робіт на наявність помилок.

Апробація. За темою магістерської роботи було подано статтю до друку [4] та опубліковано 2 тез конференції [5, 6].

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОБОТИ

1.1 Огляд автоматичної перевірки дипломних робіт

Разом зі зростанням комп'ютерних мереж та великих даних люди перевантажені великою кількістю інформації та документів в інтернеті, що викликає бажання багатьох дослідників розробити технологічний підхід, який може автоматично перевіряти тексти, а також створювати резюме, що містить важливі речення та включає всю важливу відповідну інформацію з оригінального документа. Так інформація швидко надходить і не втрачає первісного змісту документа.

У звіті працівників, які займаються адміністративною роботою в офісному середовищі, було виявлено, що до 42% з них стверджують, що їх робота могла бути більш продуктивною, якби вони мали менше завдань, пов'язаних з адмініструванням, таких як: сортування документів, ведення баз даних та перевірка оформлення документів. Значна кількість робочого часу витрачається на перевірку документів щодо орфографії, стилістичного оформлення, відповідності вимогам та наявності необхідних полів.

У сучасних галузях досі існує проблема обробки та управління фізичними документами, що призводить до ряду негативних наслідків:

- довгі терміни виконання: вручну перевіряти великий обсяг документів вимагає значних зусиль і займає значний час, особливо коли документи є обсяжними та займають більше однієї сторінки;
- помилки: один з головних недоліків ручного процесу полягає в тому, що помилки не можна повністю уникнути. Працівник може пропустити необхідний підпис, важливу частину документа (наприклад, розділ плану курсової або дипломної роботи), або

допустити незначні орфографічні помилки, такі як відсутність коми або неправильно розставлені розділові знаки;

- **масштабованість:** зростання обсягів роботи та потоку документів ускладнює масштабованість процесу обробки. Ручна обробка великого обсягу документів стає ще більш проблематичною і витратною з точки зору ресурсів.

Однак, впровадження автоматизованої системи обробки та управління документами може розв'язувати ці проблеми, дозволить автоматично перевіряти PDF-файли на орфографічні та стилістичні помилки, а також забезпечить ефективне сортування та категоризацію. Крім того, така система легко масштабується, що дозволяє оптимізувати процеси обробки документів навіть при збільшенні їх обсягів.

В сучасних умовах більшість організацій визнає необхідність новітнього підходу, який включає застосування автоматизації та методів машинного навчання для поліпшення бізнес-процесів, пов'язаних з обробкою документів. З метою більш глибокого розуміння потенційних переваг автоматизованої перевірки, було проведено порівняння між ручною та автоматичною перевіркою документів, а результати цього порівняння наведені у таблиці (табл. 1.1).

Таблиця 1.1

Порівняння ручної та автоматичної перевірки документів

Ручна перевірка	Автоматична перевірка
Потенційна втрата бізнесу: висока ймовірність втрати або переривання бізнесу через затримки.	Економія коштів: завдяки автоматизованим перевіркам можна отримати більшу ефективність за менших витрат. Ви зможете скоротити час, витрачений на перевірку або виправлення інформації в документах, як мінімум на 70%.
Схильність до помилок: людині дуже легко пропустити порожнє поле або відсутню точку даних.	Мінімізація помилок: комп'ютер, на відміну від людини, не втомлюється, не відволікається і тому відсоток помилок буде зводитися до 1%.

Продовження таблиці 1.1

Повільність: коли у вашій компанії є робочий процес, у якому вам потрібно перевірити, чи мають певні документи певну інформацію, чи значення - це займає багато часу. Що може призвести до довшого часу виконання.	Швидкість: автоматизоване рішення виявляє документи з відсутньою інформацією майже в режимі реального часу.
Важко масштабувати: перевірка документів вручну є складним і тривалим завданням. Створення команди потребує значних інвестицій у персонал. Це ускладнює масштабування організації за вимогою.	Підвищення продуктивності: швидша обробка та економія часу дозволять співробітникам зосередитися на інших важливіших завданнях.

Наукові дослідження та розвиток автоматичної перевірки текстів сприяють постійному вдосконаленню цього підходу. Одним з напрямків розвитку є використання методів машинного навчання та штучного інтелекту для покращення точності та надійності автоматичної перевірки. Застосування нейронних мереж та глибокого навчання дозволяє моделям аналізувати контекст, розпізнавати складні відношення між словами та виразами, а також виявляти відхилення від норми.

Ще одним важливим напрямком є розширення функціональності автоматичної перевірки текстів. На сьогодні не лише орфографічні та стилістичні помилки є об'єктом перевірки, але й інші аспекти, такі як граматики, пунктуація, синтаксична правильність та відповідність граматичним правилам. Розвиток таких систем дозволяє забезпечити більш глибоку та всебічну перевірку текстів, що є важливим у вимогливих документах, наукових публікаціях та інших текстах, де точність та якість є критичними.

Також, стандартизація та розробка мовних моделей впливають на якість автоматичної перевірки текстів. Розробка відповідних ресурсів, таких як корпуси текстів, лексичні бази даних та граматичні правила, дозволяє покращити розпізнавання та обробку мови. Налаштування діалогу між мовною моделлю та користувачем дозволяє враховувати специфічні потреби та вимоги

користувача, забезпечуючи більш персоналізований та ефективний процес перевірки тексту.

У результаті постійного розвитку технологій автоматичної перевірки текстів спостерігається зростання її застосування у різних сферах, таких як: освіта, наукові дослідження, бізнес та інформаційні технології. Застосування автоматичної перевірки текстів сприяє підвищенню продуктивності та якості роботи, зниженню витрат часу та ресурсів, а також забезпечує високу точність та надійність перевірки. Перспективи подальшого розвитку цього напрямку полягають у постійному удосконаленні алгоритмів та моделей, розширенні функціональності та універсальності систем, а також врахуванні специфіки конкретних мов та текстових жанрів.

1.2 Вибір та обґрунтування мови програмування для створення сервісу з перевірки PDF документів

У процесі написання сервера для автоматичної перевірки PDF документів використовуються різні підходи, бібліотеки та методи. Тому перш ніж починати працювати над реалізацією такого сервера необхідно спочатку дослідити всі існуючі та найбільш популярні методи та обрати найбільш підходящий з урахуванням його сумісності з певною мовою програмування.

Даний сервіс повинен мати можливість використовуватись як на стороні сервера, так і на стороні клієнту. В такому випадку найзручнішим рішенням при виборі мови програмування буде JavaScript (JS).

JS – це полегшена інтерпретована мова програмування з функціями першого класу. JS – це прототипна, багатопарадигмова, однопотокова, динамічна мова, що підтримує об'єктноорієнтовані, імперативні та декларативні стилі [7].

JS став практичною стандартною мовою для розробки веб сервісів та додатків, завдяки своїй високій широті застосування та підтримці в усіх

основних веббраузерах. Його використання в нашому сервісі гарантує доступність для користувачів з будь-якого пристрою та платформи.

Мова розроблена для використання в браузерах, але згодом стала підтримуватися також на серверному боці за допомогою середовища виконання Node.js. Такий широкий спектр використання дозволяє програмістам писати код, який працює як на клієнтському, так і на серверному рівнях, що спрощує розробку та утримання проектів.

JS має асинхронний характер програмування, що дозволяє виконувати операції паралельно та ефективно обробляти багато вхідних запитів без блокування виконання інших завдань, що особливо важливо в веб розробці, де швидкість та ефективність взаємодії з користувачем мають ключове значення.

Універсальність JS виявляється не лише в контексті платформ, але і в можливості використання різних фреймворків та бібліотек для реалізації веб додатків. Фреймворки, такі як React, Angular та Vue.js, надають зручні інструменти для створення користувацького інтерфейсу, динамічні можливості JS сприяють взаємодії з цими фреймворками, створюючи динамічні та високопродуктивні веб додатки.

Однією зі значущих переваг JS є активна та численна спільнота розробників, що безперервно вносить внески у вдосконалення мови та розв'язання актуальних проблем. Спільнота створює багато корисних бібліотек та інструментів, сприяючи швидкому розвитку та впровадженню нових ідей у сфері програмування.

JS також визначається своєю гнучкістю та можливістю взаємодії з іншими технологіями. Мова легко інтегрується з різними середовищами та має широкі можливості для розширення функціоналу за допомогою різноманітних плагінів та модулів.

Першим кроком у розробці сервісу було вибір необхідних інструментів та технологій. Для реалізації сервісу було обрано об'єктноорієнтовану мову програмування TypeScript (TS), яка є надмножиною JS. TS додає до JS статичну типізацію, що дозволяє розробникам визначати типи даних для

змінних, функцій та інших елементів коду, що може допомогти покращити якість коду, виявити помилки на ранніх етапах розробки та зробити код більш переносимим. TS надає можливість створення сучасних та надійних додатків зі строгою типізацією, що особливо корисно при розробці складних систем.

Статична типізація означає типізацію, при якій типи даних визначаються на етапі компіляції. Компілятор може перевірити, чи правильно використовуються типи в коді, і вивести помилку, якщо вони використовуються неправильно [8].

TS має ряд переваг, зокрема:

- покращена якість коду: статична типізація може допомогти виявити помилки на ранніх етапах розробки;
- більша переносимість: TS код може бути легко перенесений на різні платформи та середовища виконання;
- покращена продуктивність: TS може допомогти покращити продуктивність коду, оскільки компілятор може оптимізувати код на основі типів даних.

Використання TS дозволить типізувати код, що є однією з ключових переваг даної мови програмування. Згідно з офіційною документацією TS, вищезгадана мова програмування налічує всі особливості JS та має додаткову систему типізації, яка здатна вказувати на помилки на етапі розробки. Це все означає, що наш працюючий JS код також є TS кодом. Основна перевага TS полягає в можливості підкреслювати неочікувану поведінку в коді зменшуючи кількість багів [9].

TS здатний полегшувати підтримку та розширення великих кодових баз. Завдяки чіткій декларації типів, розробники можуть ефективно співпрацювати та взаємодіяти з великими проектами, що полегшує роботу команд та сприяє швидкому розвитку проектів. TS має високий рівень сумісності зі стандартом JS та дозволяє використовувати наявний JS-код без будь-яких значущих модифікацій, що робить перехід до TS більш гладким та безпечним для проектів, які вже використовують JS.

TS також надає можливість використовувати передові концепції об'єктноорієнтованого та функціонального програмування, що дозволяє розробникам створювати більш структурований та масштабований код. Налічує широкий спектр інструментів для розробки та підтримки.

Засоби автоматичної перевірки типів, інтегровані середовища розробки та інші ресурси сприяють розробникам у підтримці високого рівня якості програмного продукту. Використання інтерфейсів, абстракцій та інших концепцій підвищує читабельність коду та сприяє його подальшій підтримці.

1.3 Аналіз існуючих бібліотек для роботи з файлами PDF

Спроба вибору оптимального інструменту для роботи з PDF-файлами є ключовим етапом у розробці програмного забезпечення, і вимагає глибокого розуміння функціональних можливостей та ефективності доступних бібліотек.

Згідно з документацією, PDF – це особливий формат файлів із запровадженим стандартом [10]. Даний формат файлів використовується з метою отримання однакового вигляду файлів невеликого розміру на більшості комп'ютерах, але при виборі бібліотеки треба бути максимально обережним. З цієї причини було проведено дослідження 6 найпопулярніших бібліотек для роботи з PDF та отримано наступні результати.

PDFKit – бібліотека генерації документів PDF для Node і браузера, що робить створення складних, багатосторінкових документів для друку легким [11]. Це одна з найперших бібліотек реалізованих в JS, яка з 2012 року набула популярності та постійно оновлюється. Бібліотека підтримує власні шрифти та вбудовані зображення, але має надзвичайно малий API.

Серед основних переваг даної бібліотеки можна зазначити:

- розширені можливості, такі як робота з векторними графіками, додавання тексту, наявність маркованих списків, вбудовані шрифти тощо;

- зрозумілий і простий API, що дозволяє вам швидко і легко працювати з PDF-файлами;
- надає розширену функціональність, одночасно зберігаючи простоту розробки. Оскільки PDFKit побудована на JS, вона може бути використана в багатьох різних середовищах, включаючи браузер та серверну сторону (Node.js), що робить бібліотеку гнучкою і доступною для розробників.

До недоліків можна віднести:

- обмежену підтримку деяких специфічних функцій PDF, таких як: непрості форми, вбудовані відео або анімації;
- документація для PDFKit може бути обмеженою або неповною, що ускладнює розуміння функціональності та використання бібліотеки.

PDFMAKE – це JS-бібліотека, яка дозволяє генерувати PDF- файли безпосередньо на стороні клієнта або на серверній стороні (Node.js) [12]. Основна її відмінність від попередньої полягає в парадигмі програмування: PDFKit приймає класичний імперативний стиль, в той час, як PDFMAKE працює на декларативному підході.

До основних переваг використання даної бібліотеки відносять:

- ефективність розробки з наявністю простого та зрозумілого інтерфейсу програмування, що забезпечує зручну розробку та генерацію PDF-документів і дозволяє економити час і ресурси розробників, що є важливим у наукових дослідженнях;
- PDFMake надає широкий спектр можливостей для налаштування вигляду та форматування PDF-документів, що дозволяє точно контролювати шрифти, кольори тексту, розташування елементів, стилізацію таблиць та інші аспекти дизайну документів, відповідаючи вимогам наукових стандартів.

Недоліки використання бібліотеки PDFMake можуть включати:

- обмежені можливості редагування існуючих PDF-документів: PDFMake більш сфокусований на створенні нових документів, а не на редагуванні вже існуючих PDF-файлів. Якщо існує необхідність внести зміни до наявного PDF, користувачу може знадобитися додаткова робота з іншими бібліотеками або інструментами;
- великий розмір файлу: залежно від обсягу та складності документа, PDF-файли, створені за допомогою PDFMake, можуть мати великий розмір, що може створити проблеми зі збереженням, передачею або завантаженням таких файлів;
- відсутність підтримки деяких продвинутих функцій PDF: PDFMake може не підтримувати всі розширені функції, які можуть бути доступні в стандарті PDF. Наприклад, деякі складні типи форм, вбудовані мультимедійні елементи або інші специфічні можливості можуть бути обмежені або не підтримуватися повністю;
- відсутність підтримки платформи браузера: PDFMake призначений для використання на боковій Node.js або у середовищі Electron, і може бути обмежено підтримкою на платформі браузера.

Якщо розробник планує використовувати бібліотеку на клієнтській стороні, в такому випадку може знадобитися інша альтернатива.

jsPDF – це JS-бібліотека, яка дозволяє генерувати PDF-документи у веббраузері [13]. Дану бібліотеку не можна віднести до найлегших для опанування, але вона має надзвичайно збагачену документацію для ознайомлення.

Для більш детального аналізу наведемо основні переваги:

- jsPDF дозволяє створювати нові PDF-документи з нуля. Можна додавати текст, зображення, графіки, таблиці, посилання та інші елементи до документа;
- бібліотека надає можливість експортувати HyperText Markup Language (HTML) код у PDF-документ та використовувати

Cascading Style Sheets (CSS) для стилізації та форматування елементів у HTML-сторінці, що буде перетворена на PDF;

- jsPDF підтримує генерацію зображень з тексту, Scalable Vector Graphics (SVG) зображень та інших векторних форматів, що дозволяє створювати графіки та діаграми безпосередньо у PDF-документі;
- бібліотека надає можливість маніпулювати наявними PDF-документами, додавати, видаляти або змінювати сторінки, а також об'єднувати різні PDF-файли в один. Бібліотека може працювати у багатьох сучасних веббраузерах, що робить її кросплатформовою і доступною для веб розробників.

В якості порівняння з іншими бібліотеками, розглянемо також недоліки:

- jsPDF не надає повну підтримку для редагування наявних PDF-файлів. Бібліотека більше призначена для створення нових документів з нуля;
- jsPDF може мати обмежену підтримку деяких складніших форматів, таких як: вкладені таблиці або мультимедійні елементи;
- створення складних або великих PDF-документів може вимагати значних ресурсів, таких як: пам'ять та обчислювальна потужність, що може вплинути на продуктивність додатків, особливо на більш старих або обмежених пристроях.

Puppeteer – бібліотека Node.js, яка надає високорівневий API для управління Chrome/Chromium над протоколом DevTools. Puppeteer працює в обмеженому режимі за замовчуванням, але може бути налаштований на запуск у повному («головному») Chrome/Chromium [14]. Бібліотека надає інтерфейс для автоматизації та керування браузером Chrome або Chromium.

Основні переваги використання бібліотеки Puppeteer включають:

- автоматизація браузерних дій: Puppeteer дозволяє контролювати браузер Chrome або Chromium з програмного середовища Node.js, виконуючи різні дії на вебсторінках;

- генерація PDF-документів: за допомогою Puppeteer можна генерувати PDF-документи з вебсторінок або HTML-коду. Це зручно для створення звітів, документації або інших видів вмісту у форматі PDF;
- редагування та маніпуляція з PDF-файлами: puppeteer дозволяє внесення змін у наявні PDF-файли, такі як додавання або вилучення сторінок, редагування тексту, додавання зображень та інше, що надає можливість керувати вмістом і структурою PDF-файлів.

Основні недоліки бібліотеки:

- залежність від Chrome або Chromium: puppeteer працює з браузерами Chrome або Chromium, що може бути обмеженням, якщо вам потрібно використовувати інші браузери для роботи з PDF-файлами;
- використання puppeteer для роботи з PDF-файлами може вимагати значних ресурсів, таких як: пам'ять та обчислювальна потужність, особливо якщо вам потрібно обробляти великі або складні PDF-документи;
- інтеграція та налаштування puppeteer для роботи з PDF-файлами може бути складним завданням, особливо для початківців. Розуміння принципів роботи та правильна конфігурація можуть вимагати часу та досвіду.

Pdf-lib – це JS-бібліотека, призначена для створення та редагування PDF-документів та повністю імплементована в TS. Підтримує об'єднання, розділення, вбудовування PDF-файлів і має багато функцій, яких просто немає в інших бібліотеках, працює в будь-якому середовищі виконання JS, включаючи браузери, Node, Deno і навіть React Native [15]. Однією з найпопулярніших функцій є підтримка Uint8Array й ArrayBuffer для вбудовування файлів шрифтів, що дозволяє використовувати файлову систему, якщо ви працюєте з node, і xhr, якщо ви працюєте в браузері. Порівнявши її з іншими бібліотеками, користувач зможе відчувати її перевагу з точки зору продуктивності, і, звичайно, її можна використовувати з Webpack.

Переваги використання бібліотеки:

- pdf-lib дозволяє створювати нові PDF-документи з нуля та редагувати існуючі. Тут можна додавати, змінювати та видаляти сторінки, шрифти, зображення, графічні об'єкти та інші елементи PDF;
- бібліотека надає широкий набір функцій для маніпуляції з PDF-документами, таких як: об'єднання, розбиття та обрізка PDF-файлів, ротація сторінок, налаштування атрибутів документа та багато іншого;
- pdf-lib підтримує роботу з інтерактивними формами в PDF-документах, дозволяючи вставляти, видаляти та редагувати поля форм. Також можна застосовувати шифрування до PDF-документів для забезпечення безпеки;
- бібліотека pdf-lib може використовуватися як на стороні клієнта (у веббраузерах), так і на стороні сервера (у середовищі Node.js), що робить її кросплатформовою та універсальною для різних типів проектів.

Але, крім переваг дана бібліотека також містить ряд недоліків:

- складність використання для початківців: pdf-lib є потужною бібліотекою з розширеними можливостями, що може вимагати певного рівня експертизи та знань JS та формату PDF для ефективного використання;
- обробка великих або складних PDF-документів за допомогою pdf-lib може вимагати значних ресурсів, таких як: пам'ять та обчислювальна потужність. Потрібно враховувати ці вимоги при плануванні інфраструктури проекту;
- pdf-lib має обмежену підтримку деяких складніших типів шрифтів та ресурсів, що може впливати на точність відображення та збереження деяких елементів PDF.

Pdf-parse є популярним інструментом для обробки та аналізу PDF-файлів в середовищі Node.js. Згідно з документацією – це чисто

кроссплатформлений модуль JS для вилучення текстів х формату PDF [16]. Бібліотека pdf-parse використовується шляхом виклику відповідного методу, якому передається шлях до PDF-файлу. Результатом роботи є об'єкт, що містить витягнутий текст та інформацію, пов'язану з розпізнаванням структури документа. Застосування бібліотеки pdf-parse знайшло широке використання у сфері обробки та аналізу PDF-файлів. Дана бібліотека дозволяє витягувати текстовий контент для подальшого аналізу, пошуку та використання в інших додатках. Крім того, вона надає можливість отримувати інформацію про структуру документа, що може бути корисним для подальшої обробки, відображення або аналізу даних. Бібліотека pdf-parse має високу надійність та продуктивність, що робить її привабливим вибором для наукових досліджень, де потрібно обробляти та аналізувати PDF-документи, а зручний інтерфейс та здатність до точного розпізнавання розмітки дозволяють реалізувати складні завдання з обробки документів, що вимагають аналізу тексту та його структури.

Бібліотека pdf-parse має переваги над вищеописаними в наступних можливостях:

- pdf-parse демонструє здатність витягати текстовий контент з PDF-файлів, що відкриває можливості для подальшого аналізу та обробки даних;
- бібліотека pdf-parse дозволяє розпізнавати структуру документа, включаючи розмітку, шрифти та координати тексту, що дає змогу точніше аналізувати та використовувати вміст PDF-файлів;
- pdf-parse надає розширений набір функцій для обробки PDF-файлів, що включає витягування метаданих, отримання інформації про сторінки та інші атрибути документа. Бібліотека pdf-parse спеціально розроблена для використання в середовищі Node.js, що забезпечує зручну інтеграцію та сумісність з іншими модулями та інфраструктурою проекту.

До недоліків відносять наступні характеристики:

- pdf-parse може мати обмежену підтримку деяких складніших функцій, таких як: розпізнавання графіки або витягування структурованих даних з PDF-файлів;
- у разі, якщо вихідні PDF-файли мають складну або неправильну структуру або містять пошкоджені дані, pdf-parse може зазнавати проблем з точністю розпізнавання та витягуванням контенту;
- обробка великих або складних PDF-файлів за допомогою pdf-parse може вимагати значних обчислювальних ресурсів, таких як: пам'ять та швидкодія процесора.

Pdfjs-dist – бібліотека PDF, яка побудована на HTML5. Ціль бібліотеки полягає в створенні універсальної платформи на основі веб стандартів для аналізу та відображення PDF-файлів [17]. Pdfjs-dist бібліотека від Mozilla, яка має величезну підтримку від Mozilla org, а це забезпечує надійність підтримки сервісу в майбутньому.

PDFJS-dist є високоякісною бібліотекою, яка підтримує широкий спектр функцій PDF-файлів, включаючи:

- перегляд сторінок;
- пошук тексту;
- збільшення та зменшення масштабу;
- друк;
- завантаження та збереження сторінок;
- може швидко та плавно відображати PDF-файли;
- може використовуватися в будь-якому веббраузері або додатку.

Недоліки:

- PDFJS-dist може бути великою бібліотекою, яка може займати багато місця на диску;
- може бути складною бібліотекою для використання, особливо для початківців.

Після аналізу отриманих результатів, оцінки параметрів використання різних бібліотек для роботи з PDF та дослідження на практиці, було вирішено, що оптимальним вибором для даного контексту є бібліотека pdfjs-dist. Саме за допомогою даної бібліотеки існує чудова можливість розпарсити документ формату PDF для подальшої роботи.

Зважаючи на потреби й задачі дипломної роботи, бібліотека pdfjs-dist може бути заслужено вибраною варіантом для обробки та аналізу PDF-файлів. Однією з головних переваг є здатність ефективно витягувати текстовий контент з PDF-документів, що надає дослідникам можливість проводити глибокий аналіз та обробку документів, що містять наукові джерела та інші важливі дані.

Крім того, pdfjs-dist забезпечує простоту використання та гнучкість у роботі з текстом та надає можливість виконувати розширену обробку даних, таку як виявлення та аналіз структурованої інформації, виділення ключових термінів або додаткових метаданих, що сприяє покращенню якості дослідження та робить бібліотеку pdfjs-dist потужним інструментом для наукової аналітики. Таким чином, обрана бібліотека допомагає забезпечити точність та надійність аналізу PDF-документів, що є критичними факторами у наукових дослідженнях.

1.4 Наявні підходи до аналізу та результату перевірки дипломних робіт

Для виконання аналізу та отримання результату перевірки документів використовують різноманітні підходи, які базуються на традиційних методах, таких як: ручна перевірка та експертна оцінка, тоді як інші використовують сучасні автоматизовані методи, включаючи штучний інтелект, машинне навчання та обробку природної мови.

Використання машинного навчання — один із методів, який використовуються найчастіше, а також дозволяє розробляти моделі, які

можуть автоматично класифікувати та аналізувати документи за допомогою навчання на великих обсягах даних.

Термін «машинне навчання» означає автоматичне виявлення значущих патернів у даних. За останні декілька десятиліть він став поширеним інструментом при виконанні майже будь-якого завдання, яке вимагає вилучення з великих баз даних [18]. Використання алгоритмів машинного навчання дозволяє виявляти складні залежності та патерни в даних, що сприяє більш точному та ефективному аналізу.

Окремий підхід до аналізу та перевірки документів полягає у використанні обробки природної мови (Natural Language Processing, NLP). Обробка природної мови (ОПМ) – це підвид штучного інтелекту, який зосереджується на інтерпретації комп'ютерної лінгвістики [19]. Сфера охоплює кілька областей інтерпретації тексту та аудіо шляхом інтеграції методів машинного навчання, які поводяться статистично. Також охоплює сферу прагматичних досліджень комп'ютерної лінгвістики, яка стала дуже широкою та потужною завдяки застосуванню різноманітних методів [19]. ОПМ дозволяє виконувати автоматичне розпізнавання, аналіз та розуміння текстової інформації, що міститься у документах. Застосування ОПМ методів дозволяє виявляти ключові терміни, розрізняти семантичні відтінки, аналізувати синтаксичну та граматичну структуру речень, а також виявляти та коригувати орфографічні та граматичні помилки, що дозволяє отримати більш глибоке розуміння змісту та контексту документів під час їх аналізу та перевірки.

Окрім того, сучасні підходи до аналізу та перевірки документів включають інтеграцію штучного інтелекту в процес обробки документів, дозволяють автоматизувати та покращити точність аналізу документів за допомогою розпізнавання образів, автоматичного виявлення аномалій та асоціацій, а також використання алгоритмів самонавчання для постійного вдосконалення системи, що дає можливість отримати більш надійні результати та забезпечити швидку та ефективну перевірку документів.

Крім того, варто згадати про використання гібридних підходів, які комбінують різні методи та техніки аналізу та перевірки документів для досягнення кращих результатів. Наприклад, можна поєднувати машинне навчання з обробкою природної мови для забезпечення якісного розпізнавання та аналізу текстової інформації в документах. Такий гібридний підхід дозволяє використовувати переваги обох методів і покращити якість аналізу та перевірки. Наприклад, при розпізнаванні та аналізі тексту документа можна використовувати методи обробки природної мови для виявлення ключових термінів, розпізнавання іменованих сутностей, аналізу синтаксичних та семантичних зв'язків між словами та фразами. Використання обробки природної мови дозволяє отримати глибше розуміння тексту, його семантики та контексту. Після отримання аналітичних відомостей про текстову інформацію можна використовувати методи машинного навчання для класифікації та аналізу документів. Наприклад, можна використовувати моделі машинного навчання, навчені на великому обсязі даних, для класифікації документів на певні категорії або виявлення аномалій.

Машинне навчання дозволяє виявляти складні залежності та патерни у даних, що поліпшує точність та ефективність аналізу документів.

Поміж інших підходів до аналізу та перевірки документів є використання експертних систем, що базуються на правилах. Експертні системи використовують знання експертів у певній доменній області та правила логіки для аналізу та класифікації документів. Дані системи можуть бути налаштовані таким чином, щоб застосовувати спеціалізовані правила для виявлення певних аномалій або недоліків у документах. Використання експертних систем може бути особливо корисним у випадках, коли потрібна точна інтерпретація документів за допомогою знань експертів.

Інший підхід, що варто відзначити – використання глибинного навчання для аналізу та перевірки документів. Глибинне навчання – це галузь машинного навчання, що ґрунтується на наборі алгоритмів, які намагаються моделювати високорівневі абстракції в даних, застосовуючи глибинний граф

із декількома обробними шарами, що побудовано з кількох лінійних або нелінійних перетворень [20]. Глибинне навчання є підкатегорією машинного навчання, яка зосереджується на навчанні нейронних мереж з багатьма шарами для автоматичного виявлення складних патернів у даних. Використання глибинного навчання може дозволити розробити моделі, які здатні аналізувати великі обсяги текстової інформації, розпізнавати відтінки семантики та контексту, а також забезпечувати високу точність та ефективність аналізу документів.

Також, в контексті аналізу та перевірки документів, важливим підходом є використання алгоритмів обробки зображень, що особливо актуально в разі, коли документи містять скановані або цифрові зображення, які потребують аналізу та перевірки. Застосування алгоритмів обробки зображень дозволяє автоматично виявляти та виділяти текстову інформацію, виконувати розпізнавання символів, виявляти та аналізувати графічні елементи, такі як діаграми або рисунки. Підхід доповнює аналіз тексту та забезпечує більш повну обробку документів.

Додатковою стратегією для аналізу та перевірки документів є використання технологій обробки мовлення, що охоплює аналіз та інтерпретацію усного або письмового мовлення, зокрема розпізнавання мови, синтез мови, аналіз семантики та інші техніки. Використання технологій обробки мовлення дозволяє виявляти та розуміти ключові фрази, ідіоми, сарказм, а також виконувати автоматичне коригування орфографії та граматики у текстових документах. Підхід сприяє покращенню розуміння змісту та контексту документів.

Загалом, наявні підходи до аналізу та перевірки документів охоплюють використання машинного навчання, обробки природної мови, експертних систем, глибинного навчання, алгоритмів обробки зображень та технологій обробки мовлення. Комбінування цих підходів або розробка гібридних методів може забезпечити більш точний, ефективний та автоматизований аналіз та перевірку документів з різних доменних областей та задач.

Розглянувши можливості використання машинного навчання, проаналізуємо використання алгоритмів при розробці сервісу.

Алгоритм – система формальних правил, що визначає зміст і порядок дій над вхідними даними та проміжними результатами, необхідними для отримання кінцевого результату при розв’язуванні задачі [21].

Підходи до аналізу та перевірки документів надають широкий спектр можливостей для дослідження та розробки ефективних систем. Вибір конкретного підходу залежить від особливостей досліджуваної задачі, обсягу документів та вимог щодо точності та ефективності аналізу.

Дослідивши декілька джерел стосовно алгоритмів [22-27] та проаналізувавши вищеописані підходи, нами було вирішено уникати елементи машинного навчання чи обробки природної мови, забезпечуючи прозорість функціонування системи. Натомість було вирішено розробити власні алгоритми перевірки списку використаних джерел на правильність оформлення згідно вищої атестаційної комісії (ВАК) України, на співвідношення сторінок в джерелах та в документі та на співвідношення елементів змісту в документі та в змісті.

Висновки до розділу 1

Даний розділ охоплює теоретичну інформацію про автоматичну перевірку дипломних, аналіз існуючих бібліотек для роботи з файлами PDF на різних мовах програмування, а також вибір та обґрунтування мови програмування для створення сервісу з перевірки PDF документів. На додачу, в розділі наведені підходи до аналізу та результату перевірки документів.

В результаті проведеного дослідження, на даному етапі було вирішено використовувати бібліотеку pdfjs-dist, оскільки вона гарантує надійність та високу якість підтримки сервісу, активно оновлюється та адаптується до змін в стандартах PDF і вказує на довіру до технічної надійності, що є важливим фактором для успішного впровадження веб сервісів.

Згідно результатів отриманих в 1 розділі на даному етапі, було вирішено розробити інноваційні алгоритми для перевірки списку використаних джерел згідно з вимогами ВАК, перевірки правильності розташування джерел в бібліографії і в документі та перевірку на відповідності змісту сторінкам та заголовкам в змісті та в документі.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ СЕРВІСУ ДЛЯ АВТОМАТИЧНОЇ ПЕРЕВІРКИ ДИПЛОМНИХ РОБІТ

2.1 Архітектура сервісу

Розробка стійких та ефективних програмних рішень починається з обґрунтованого підходу до архітектури сервісу, що визначає основні принципи та структурні засади системи. Ефективність архітектурного рішення визначається не лише його здатністю задовольняти поточні вимоги, але і його готовністю до майбутніх розширень та змін. Врахування принципів масштабованості та безпеки є важливим етапом у процесі створення стабільного та надійного програмного рішення.

Багато академічних робіт та документів оформлюються у форматі PDF, що робить його одним із найпоширеніших форматів для подання робіт. Сервіс перевірки повинен бути здатний працювати з PDF-файлами, визначати текст, графіку та інші елементи документа, ідентифікувати внутрішню структуру та перевіряти її відповідність академічним стандартам.

Сервіс виконує перевірку відразу після завантаження PDF файлу в систему та видає результат. В результаті студенти та викладачі зможуть побачити наявні помилки, що допоможе студентам їх виправити та покращити свої бали при здачі академічної роботи. Для викладачів цей сервіс значно економить час, адже замість годин прочитання, викладач за лічені хвилини отримає результат оформлення звіту академічної роботи студента. Всі результати перевірки надаються в електронному вигляді.

Перш ніж переходити до розробки, розглянемо можливі варіанти архітектури та оберемо найбільш підходящий саме для нашого сервісу.

Архітектура програмного забезпечення (англ. software architecture) – це набір структур або уявлень, що мають різні рівні абстракції і що показують різні аспекти (структуру класів ПЗ, структуру розгортання, тобто прив'язки компонентів ПЗ до фізичних машин, можливі сценарії взаємодій

компонентів), що об'єднуються зіставленням всіх представлених даних із структурними елементами ПЗ [28].

Сервісно-орієнтована архітектура (COA) – це метод розробки програмного забезпечення, який використовує програмні компоненти, які називаються послугами, для створення бізнес-додатків. Кожна послуга надає бізнес-функціональність, і послуги також можуть взаємодіяти між собою на різних платформах і мовах. Розробники використовують COA для повторного використання послуг у різних системах або об'єднання декількох незалежних послуг для виконання складних завдань [29].

Клієнт-серверна архітектура – це ще один важливий аспект програмної архітектури, який впливає на розробку програмного забезпечення та його взаємодію з користувачами та серверами. Архітектура клієнт – сервер (client-server architecture) – це концепція інформаційної мережі, в якій основна частина її ресурсів зосереджена в серверах, що обслуговують своїх клієнтів [30] (рис. 2.1).

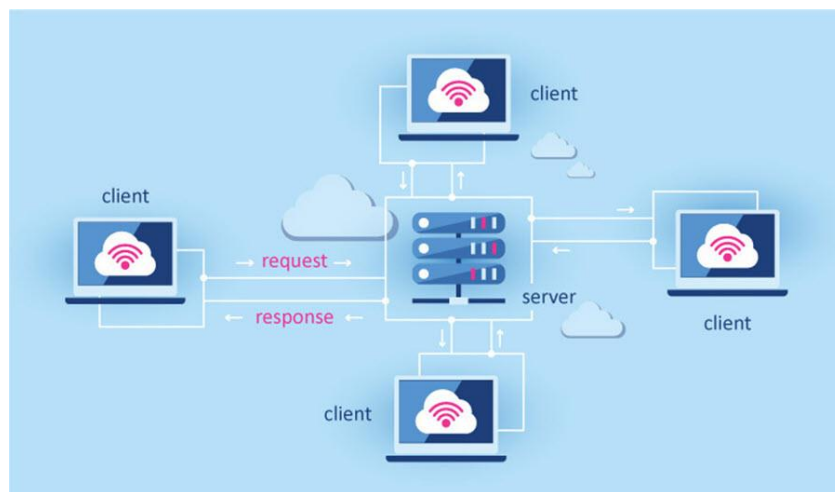


Рис. 2.1 Архітектура клієнт сервер [31]

Клієнт-серверна архітектура визначає спосіб, яким клієнти та сервери взаємодіють у розподіленій системі. У такій системі клієнти є користувачами або клієнтськими програмами, які звертаються до серверів, що надають певні послуги або ресурси. Основна ідея полягає в тому, що сервери відповідають на запити клієнтів, надаючи їм необхідну інформацію чи функціональність.

Одночасно клієнти можуть бути різними пристроями або програмами, що взаємодіють із серверами через мережу.

Архітектура MVVM (Model-ViewViewModel) – шаблон, що підтримує двостороннє зв'язування даних між View та ViewModel та дозволяє автоматично розповсюджувати зміни, які виникають усередині ViewModel до View [32].

Іншими словами, MVVM – це шаблон архітектури програмування, який використовується для розділення даних, логіки та інтерфейсу користувача, використовується в основному для розробки графічного інтерфейсу користувача. Шаблон MVVM часто представляють у лінійному порядку (рис. 2.2).

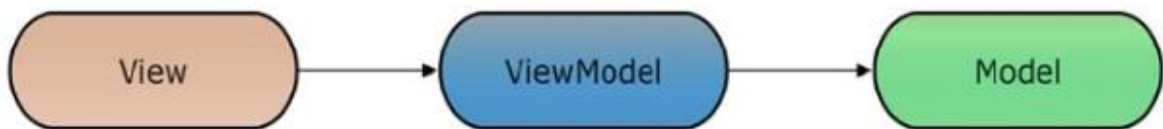


Рис. 2.2 Шаблон MVVM [33]

MVVM складається з трьох компонентів [34] (рис. 2.3):

- модель – описує використовувані в застосунку дані. Моделі можуть містити логіку, безпосередньо пов'язану цими даними, наприклад, логіку валідації властивостей моделі;
- модель подання (ViewModel) – пов'язує модель і уявлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу `INotifyPropertyChanged` автоматично йде зміна відображуваних даних в поданні, хоча безпосередньо модель і уявлення не пов'язані;
- уявлення (View) – визначає візуальний інтерфейс, через який користувач взаємодіє із застосунком.

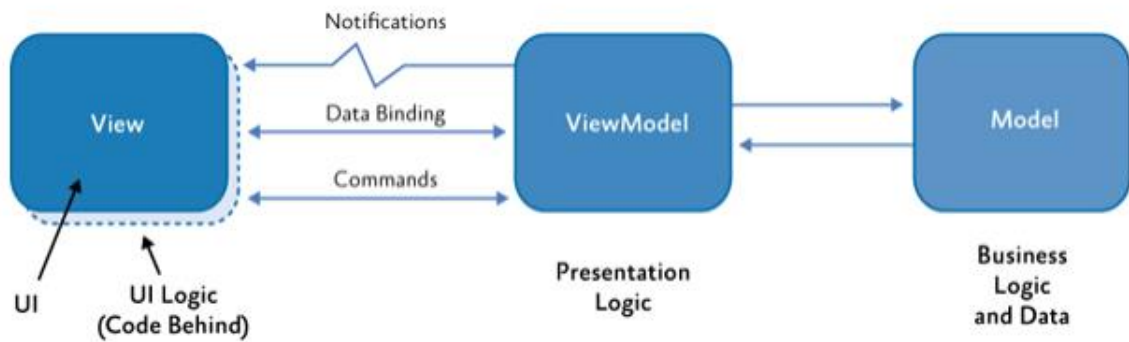


Рис. 2.3 Патерн MVVM [34]

Мікросервісна архітектура розподіляє додаток на більш дрібні, повністю незалежні компоненти, що забезпечує їм більшу гнучкість і масштабованість. Цей тип архітектури передбачає велику кількість невеликих сервісів, кожен з яких виконує свої власні функції та може бути незалежно розгорнутий. Такі сервіси виконуються в окремих процесах та комунікують між собою через веб запити або віддалені виклики процедур [35].

Мікросервіси дозволяють командам розробників працювати над окремими сервісами незалежно один від одного. Це полегшує розробку, випробування та розгортання, а також сприяє гнучкості та масштабованості системи. Важливо мати чітко визначені інтерфейси між мікросервісами для ефективної взаємодії.

Кожен сервіс може використовувати власний технологічний стек, базу даних та мови програмування, що дозволяє використовувати найкращі практики та інструменти для кожного конкретного випадку (рис. 2.4).



Рис. 2.4 Типова довідкова архітектура мікросервісного веб додатка [36]

Правильно спроектована архітектура допоможе зробити сервіс більш надійним, масштабованим та легко підтримуваним.

Розглянувши найбільш відомі варіанти побудови архітектури та приймаючи до уваги тему даної дипломної роботи, провівши необхідні дослідження ми можемо підсумувати, що в нашому сервісі використовується мікросервісна архітектура, оскільки наш сервіс створений шляхом розбиття на окремі сервіси (рис. 2.5).

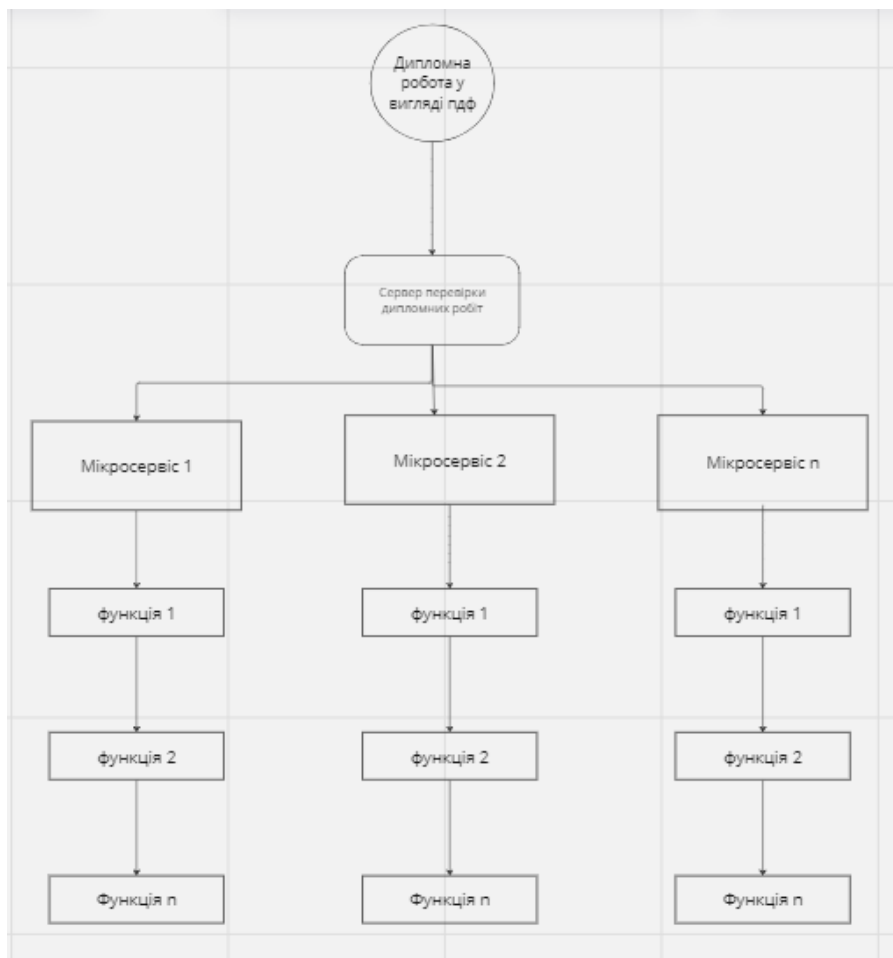


Рис. 2.5 Розбиття сервісу перевірки дипломних робіт на мікросервіси

В даному контексті, виокремлення різних функціональних блоків алгоритмів, які відповідають за різні аспекти перевірки магістерської роботи, вказує на використання мікросервісної архітектури. Кожен «мікросервіс» виконує конкретну функцію і може бути незалежно розгортаним, модифікованим та масштабованим.

Мікросервіс 1 відображає алгоритм перевірки джерел бібліографії на правильність оформлення згідно з вимогами ВАК, який складається з декількох функцій перевірки.

Мікросервіс 2 означає перевірку джерел бібліографії на правильність розміщення в списку використаної літератури та в документі. Алгоритм розбитий на декілька функцій для покращення читабельності коду.

Останній, третій мікросервіс в даній магістерській роботі відповідає за перевірку відповідності елементів змісту в змісті та в документі та містить декілька функцій.

На схемі зображено мікросервіс n, через те, що інші мікросервіси з перевірки входять до кваліфікаційної магістерської роботи Желізко В.В. і докладно описані в [37].

Застосування мікросервісної архітектури у магістерській роботі дозволяє розбити складний процес перевірки на невеликі, самодостатні компоненти, що спрощує розробку, тестування та підтримку. Кожен мікросервіс може функціонувати як незалежна одиниця, забезпечуючи гнучкість та масштабованість в роботі з різними вимогами перевірки магістерської роботи.

2.2 Опис та реалізація алгоритмів перевірки на правильність оформлення списку використаних джерел

Сервіс можна розглядати як програмний продукт, який надає певну послугу користувачам або іншим системам. Сервіси можуть бути різними за своїм призначенням, але завжди складаються з певних основних компонентів. У даній магістерській роботі ми виокремлюємо наступні компоненти мікросервісу з перевірки на правильність оформлення списку використаних джерел:

- перевірка правильності оформлення списку літератури(згідно з вимогами ВАК);

- перевірка на розміщення джерел по порядку в тексті та в бібліографії;
- перевірка форматування джерел [число] [число, число] [число - число, число];
- перевірка на розділові знаки в джерелах.

Перевірка правильності оформлення списку літератури

Всі академічні роботи, які подаються до захисту в спеціалізованих вчених радах, повинні містити список використаних джерел, оформлений згідно вимог ВАК України. Єдиних положень на цей момент не існує, тому нами були обрані загальні, згідно з якими список використаних джерел повинен містити повні відомості про всі джерела, на які посилається автор у своїй роботі.

Зважаючи на різноманіття вимог, нами було вирішено писати сервіс, опираючись на один з існуючих стандартів:

1. електронний ресурс оформлено правильно, якщо містить фразу «[Електронний ресурс]»;
2. обов'язково - фраза – «– Режим доступу до ресурсу:» + посилання;
3. нумерація джерел має бути у форматі «цифра - крапка – пробіл»;
4. джерела мають зустрічатися в порядку зростання.

Якщо джерела не відповідають обов'язковим вимогам, має виводитися повідомлення про помилку і саме джерело, яке містить помилку.

Перевірку списку літератури можна розділити на декілька етапів:

1. етап - визначення початку та кінця списку літератури;
2. етап - визначення елементів списку літератури з розпаршених даних;
3. етап - безпосередня перевірка елементів списку літератури.

Дані етапи формують задачі, для виконання яких ми застосуємо наступні алгоритми.

Бібліографію можна знайти по змісту за умови, якщо вона правильно вказана в змісті. Якщо по змісту визначити не вдалося будемо можна шукати послідовно по сторінках. Крім того, краще починати шукати з останньої сторінки послідовно перевіряючи сторінки допоки не знайдемо, бо список літератури повинен бути в кінці.

Після знаходження початку списку літератури потрібно знайти його кінець. Кінець списку літератури буде за умови, якщо в змісті зазначені додатки, в іншому випадку, список літератури буде міститися до кінця документа. Визначити додатки ми спершу можемо по змісту. Якщо сторінки не співпадають, ми застосовуємо той самий алгоритм, що використовували для пошуку списку літератури тільки з додатками, а саме – пошук з кінця заголовка «Додатки».

Після виконання першого етапу переходимо до визначення елементів списку літератури з розпаршених даних, які ми маємо у вигляді масиву текстових рядків. Алгоритм має розпізнати, коли новий рядок є початком нового елементу списку літератури. Для цього ми створюємо пустий масив, в який будемо додавати знайдені елементи. Циклом проходимося по рядках, перевіряючи їх. Якщо рядок починається з цифри до 1000, після якої є кома, ми робимо припущення, що це початок нового елементу списку літератури та зберігаємо це значення в константу. Якщо наступний рядок не є новим елементом списку літератури, ми додаємо його значення до збереженої константи, а якщо рядок є новим елементом – збережену константу додаємо в масив елементів списку літератури, а значення рядка зберігаємо в константу. Після закінчення циклу потрібно додати останнє збережене значення в масив елементів.

Було встановлено умовну константу 1000, щоб у список літератури не потрапили цифри, які не є номером нумерації елементів списку літератури. А саме, рік публікації. Дане припущення може бути покращене в майбутньому.

Після виконання другого етапу, переходимо до останнього. Як вже було сказано вище, ми перевіряємо елементи списку літератури за допомогою циклу, а саме:

- чи джерела зустрічаються по порядку в порядку зростання;
- формат номера та чи відповідає він вимогам оформлення;
- наявність обов'язкових атрибутів, наприклад, чи є в джерелі фраза [електронний ресурс] та фраза «доступ до ресурсу».

Для виконання перевірки застосовуємо шаблонні вирази. Якщо знаходимо розбіжності встановлюються відповідні помилки, вказуючи помилку та елемент списку літератури, де знайдено помилку.

Перевірка на розміщення джерел по порядку в тексті та в бібліографії та перевірка на розділові знаки в джерелах

Кожна академічна робота повинна містити список літератури, який є обов'язковим елементом оформлення. Список літератури дозволяє читачеві швидко знайти джерела, на які автор посилається в тексті роботи. Однак, не завжди автори правильно використовують список літератури. Згідно з вимогами, посилання на літературні джерела приводяться в тексті у квадратних дужках та мають бути в порядку зростання від цифри один. Формат чисел має відповідати одному з шаблонів [число], [число, число], [число - число, число] [38].

Розділові знаки допомагають читачеві зрозуміти структуру бібліографії та її зміст. Для того, щоб наш сервіс міг перевіряти чи правильно в бібліографії розміщені розділові знаки, спочатку треба ознайомитися з вимогами, згідно з якими до і після приписаного знака має бути проміжок в 1 друкований знак, винятки становлять крапка і кома. У винятках проміжки мають бути після крапки або коми, а квадратні та круглі дужки мають виділятися проміжками лише ззовні [38].

Одна з головних проблем – неправильне розташування джерел бібліографії в документі. Дуже часто при написанні дипломної роботи

студенти розташовують посилання на бібліографічні джерела в тому порядку, в якому їх знаходять, тобто посилання можуть бути в змішаному порядку.

Для розв'язання цієї проблеми було розроблено алгоритм перевірки правильного розташування літературних джерел в документі, який полягає в знаходженні всіх згадувань літературних джерел і їх перевірці за допомогою шаблонних виразів.

Для того, щоб знайти бібліографічні джерела, потрібно послідовно перевірити всі сторінки, на яких мають бути посилання на літературні джерела, а саме від вступу до списку літератури.

Нам потрібно валідувати знаки перед посиланням і після, тому вибірка буде передбачати два знаки до посилання та один знак після. В посиланні мають міститися у квадратних дужках будь-які комбінації пробілів, ком, дефісів, тире і цифр.

Валідація літературних джерел полягає в перевірці символів перед посиланням:

- саме посилання;
- символів після посилання;
- послідовності використання посилань;
- чи є всі посилання які вказані в літературних джерелах.

Перед посиланням може бути пробіл, а перед пробілом можуть бути лише літери. Після закриваючої дужки посилання не може бути літера.

Перевірка посилань має здійснюватися за наступними критеріями:

- наявність зайвих пробілів;
- наявність пробілів після коми;
- наявності різних типів тире(має бути єдиний формат, а саме дефіс).

Для перевірки послідовності використання посилань розроблено наступний алгоритм.

Всі посилання одиничні (окрема цифра) та діапазони цифр, попередньо перетворюючи в послідовність цифр, додаємо в масив. Якщо цифра вже є в

масиві її додавати не потрібно, адже джерело згадується вдруге. Знаючи, яка має бути кількість посилань перевіряємо які елементи розміщуються не послідовно, та які елементи відсутні/зайві.

Якщо не виконуються умови на будь-якому етапі перевірки — виводимо деталі помилок.

2.3 Опис та реалізація алгоритмів перевірки на відповідності змісту сторінкам і заголовкам в роботі.

Перевірка відповідності змісту сторінкам і заголовкам в роботі

Як і в попередніх сервісах, перш ніж переходити до розробки визначимо основні вимоги до написання змісту та елементів змісту в документі. Згідно з вимогами державного університету «Житомирська політехніка» [39] у змісті подають найменування розділів, підрозділів та пунктів магістерської атестаційної роботи, зазначаючи номери початкових сторінок, відповідно до структури роботи. Заголовки змісту повинні точно повторювати заголовки по тексту атестаційної роботи. Розділи мають бути з великої літери, в змісті повинні мати порядкові номери, позначені арабськими цифрами. Найменування підрозділів записують малими літерами (крім першої великої), кількість в змісті повинна бути не більше п'яти, але не менше трьох. Підрозділи в змісті та на сторінці повинні мати порядкові номери, позначені арабськими цифрами. Номер розділу і підрозділу відокремлюють крапкою. В кінці кожного розділу мають бути висновки. Слово «висновки» має бути написане малими літерами (крім першої великої).

Після висновків до 3 розділу мають бути загальні висновки, написані великими літерами. Якщо в роботі є додатки, вони йдуть останнім елементом змісту і перевіряються лише при наявності в роботі. Підсумовуючи, можемо зробити висновки, що зміст має складатися з елементів, як показано на (рис. 2.6).

	ЗМІСТ
РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. НАЗВА РОЗДІЛУ	9
1.1 Назва підрозділу	9
1.2 Назва підрозділу	12
1.3 Назва підрозділу	17
1.4 Назва підрозділу	25
Висновки до 1 розділу	29
РОЗДІЛ 2. НАЗВА РОЗДІЛУ	30
2.1 Назва підрозділу	30
2.2 Назва підрозділу	35
2.3 Назва підрозділу	38
2.4 Назва підрозділу	41
2.5 Назва підрозділу	44
Висновки до розділу 2	46
РОЗДІЛ 3. НАЗВА РОЗДІЛУ	47
3.1 Назва підрозділу	47
3.2 Назва підрозділу	52
3.3 Назва підрозділу	55
Висновки до 3 розділу	62
ВИСНОВКИ	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	65
ДОДАТКИ	71
Додаток А	72
Додаток Б	73

Рис. 2.6 Основні елементи змісту

З метою удосконалення якості контролю даних, було розроблено алгоритм перевірки елементів в змісті та в документі, який можна розділити на декілька етапів:

- визначення елементів змісту;
- перевірка структури змісту;
- перевірка відповідності заголовків на сторінках до змісту.

Для визначення елементів змісту потрібно проаналізувати розпаршений документ і, у цьому випадку, буде зручно використовувати масив рядків як результат парсингу. Наступним кроком, потрібно визначити початок і кінець змісту. Зміст починається від слова “ЗМІСТ” і закінчується додатками. Якщо немає слова “ЗМІСТ”, необхідно вивести помилку про відсутність змісту. Якщо немає додатків, останній елемент змісту - список використаної літератури. З отриманого масиву рядків потрібно виокремити назву елементу змісту та номер сторінки на якій він вказаний. Для цього потрібно перетворити

масив рядків в масив об'єктів, в якому і буде назва елементу змісту і номер сторінки.

Розроблений алгоритм розпізнавання базується на визначенні структурованих елементів тексту. Він виявляє рядок, який закінчується цифрою, перед якою ідуть крапки, а наступний рядок починається з номера, який є елементом списку. Під час розпізнавання потрібно враховувати крапки, які розміщені між назвою та номером сторінки, їх не потрібно включати в назву елементу змісту.

Для перевірки структури змісту необхідно по черзі перевірити назви елементів змісту і вивести помилки, якщо певний елемент змісту не вказаний або не відповідає вимогам оформлення. Після цього визначається кількість підрозділів в розділах і перевіряється наявність висновків до підрозділів.

Маючи структуру змісту з визначеними назвами елементів змісту і номерами сторінок, перевіряємо наявність на вказаних сторінках даних заголовків і чи вони мають однаковий регістр. Для цього парсимо вказані сторінки в форматі масиву рядків і шукаємо відповідні заголовки. Важливою є перевірка назви розділу, який має починатися з нової сторінки.

Висновки до розділу 2

Було детально розглянуто реалізацію сервісу для автоматичної перевірки дипломних робіт. Обговорено архітектуру сервісу, описано вимоги до його компонентів, а також надано інформацію про розробку та реалізацію цих компонентів.

Було розібрано структуру системи, визначено основні компоненти та їх взаємодію. Висвітлено принципи, які лежать в основі архітектури, зокрема, її масштабованість, ефективність та забезпечення безпеки взаємодії з користувачами.

Надано перелік вимог до оформлення списку використаних джерел, елементів в змісті та в документі та описано алгоритми для реалізації вищезгаданих перевірок.

РОЗДІЛ 3. ЕФЕКТИВНІСТЬ ТА ТЕСТУВАННЯ СЕРВІСУ ДЛЯ ПЕРЕВІРКИ АКАДЕМІЧНИХ РОБІТ

3.1 Тестування та налагодження

Тестування є критичним етапом в розробці програмного забезпечення, гарантуючи його якість та надійність та допомагає виявляти помилки, забезпечуючи користувачів стабільною роботою продукту. Тестування на різних етапах розробки дозволить вчасно виявляти та виправляти можливі дефекти, гарантуючи високу якість та надійність нашого сервісу.

Тестування – це процес використання ПЗ при умові аналізу або запису отримуваних результатів з метою перевірки (оцінки) деяких властивостей тестованого об'єкта [40].

Модульне тестування, або юніт-тестування – процес програмування, який дозволяє ізолювати перевірити кожен окремий модуль вихідного коду [41]. Окремий тест пишеться для кожної нетривіальної функції або методу. Таке рішення дозволяє відносно швидко перевірити чи не призвела чергова зміна коду до регресії, тобто появи нових помилок у відтестованих модулях додатка, а також дозволяє виявляти та, як наслідок, усувати такі помилки [41].

Ручне тестування (manual testing) – це процес ручної перевірки програмного забезпечення на помилки. Тестувальник має відігравати роль користувача програми й використовувати властивості програми для знаходження помилок у роботі програми. Для професійного тестування тестувальник часто користується написаним планом тестування з варіантами тестування (test cases) [42].

З метою ручного тестування було створено декілька тестових файлів на основі нашої дипломної записки, які містили всі помилки, описані в правилах в сервісі.

Мікросервіс з перевірки списку використаної літератури, у випадку недоліків у записці, висвітлить наступні помилки:

- не вдалося визначити й перевірити «список використаних джерел»;
- джерела номер «{str1}» мають зустрічатися в порядку зростання;
- нумерація джерел «{str1}» має бути у форматі цифра - крапка – пробіл;
- відсутній вираз «[Електронний ресурс]» в «{str1}»;
- відсутній вираз «[Режим доступу до ресурсу:]» в «{str1}».

Перевіримо чи сервіс знайде вищеперераховані помилки на прикладі дипломної роботи (рис. 3.1)

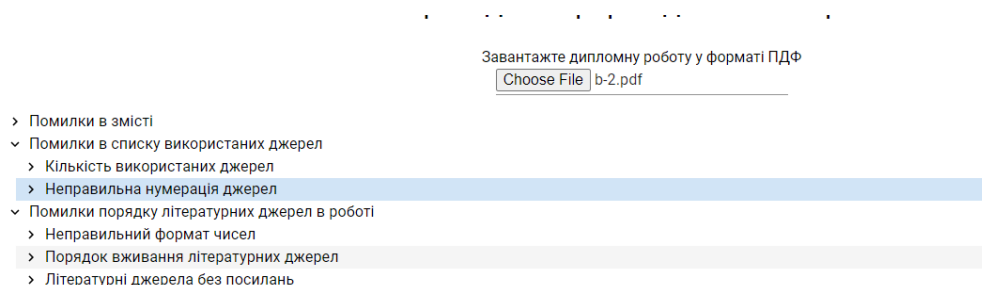


Рис. 3.1 Перевірка тестової дипломної роботи на помилки в джерелах

Ми отримали п'ять помилок, розглянемо їх детальніше та порівняємо з PDF файлом, щоб перевірити чи правильно працює наш сервіс. Спочатку порівняємо помилку про неправильну нумерацію джерел на сервісі (рис. 3.2) та в документі (рис. 3.3).

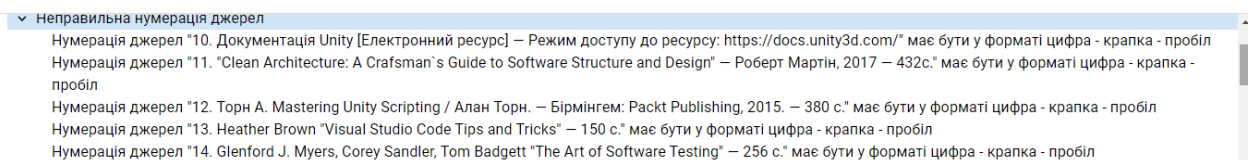


Рис. 3.2 Помилка в нумерації джерел

8. Пул об'єктів [Електронний ресурс] — Режим доступу до ресурсу:
<https://betterprogramming.pub/object-pooling-in-unity-aac9ccbc5dbd>
9. Документація Unity [Електронний ресурс] — Режим доступу до ресурсу:
<https://forum.unity.com/>
10. Документація Unity [Електронний ресурс] — Режим доступу до ресурсу:
<https://docs.unity3d.com/>
11. "Clean Architecture: A Craftsman's Guide to Software Structure and Design"
— Роберт Мартін, 2017 — 432с.
12. Торн А. Mastering Unity Scripting / Алан Торн. — Бірмінгем: Packt Publishing, 2015. — 380 с.
13. Heather Brown "Visual Studio Code Tips and Tricks" — 150 с.
14. Glenford J. Myers, Corey Sandler, Tom Badgett "The Art of Software Testing" — 256 с.

Рис. 3.3 Нумерація джерел в PDF файлі

Як бачимо, в джерелах 10-14 дійсно допущена помилка формату: після крапки немає пробілу.

Перевіримо чи правильно сервіс перевіряє формат чисел в документі. Нижче (рис. 3.4) показано помилку, яку нам видав сервіс.

▼ Помилки порядку літературних джерел в роботі

▼ Неправильний формат чисел

Неправильний формат чисел в джерелі: "1,11". Має бути [число], [число, число], [число - число, число]

Неправильний формат чисел в джерелі: "3,4". Має бути [число], [число, число], [число - число, число]

Неправильний формат чисел в джерелі: "14,15". Має бути [число], [число, число], [число - число, число]

Рис. 3.4 Помилка у форматі джерел в документі

Перевіримо чи дійсно допущена помилка в цьому джерелі в документі (рис. 3.5)

1.3. Вибір архітектури

Для розробки даної системи використовувався найпопулярніший патерн, а саме MVC [\[1,11\]](#).

Рис. 3.5 Формат джерела в документі

Як бачимо, при посиланні на джерело дійсно допущена помилка, адже після коми має бути пробіл.

Наступні помилки, які нам видає сервіс стосується порядку вживання джерел та їх оформлення згідно ВАК(рис. 3.6)

- ✓ Порядок вживання літературних джерел
Літературні джерела мають бути розташовані в порядку зростання. Розпізнані номери: 9,10,1,11,12,3,4,13,5,7,6,8,14,15
- ✓ Літературні джерела без посилань
Літературні джерела які не мають посилань: 2

Рис. 3.6 Помилка вживання та оформлення джерел у сервісі

Перевіримо чи в документі дійсно допущені вищезгадані помилки (рис. 3.7)

1.1. Технічне завдання на розробку системи

Основною метою розробки гри є створення захоплюючого та емоційного ігрового досвіду для гравця. Колекційно карткова гра з елементами РПГ та покроковою боївкою, далі Munchkin Rogue-like повинна надати можливість гравцеві відчувати себе у ролі головного героя, зануритися в ігровий світ і взаємодіяти з ним [9]. Необхідно розробити додаток, який наблизить геймплей до реалістичного.

Рис. 3.7 Перевірка вживання джерел в документі

Як бачимо, в документі джерела дійсно розташовані в неправильному порядку, адже в підрозділі 1.1 в самому початку використане 9 джерело, хоча має починатися з першого. А в підрозділі 1.3 (див. рис. 3.5) використані 1 та 11 джерело, що суперечить правилам оформлення послідовності джерел.

Перевіримо наш сервіс додатково за допомогою модульного тестування (рис. 3.8). Однією з ключових переваг модульного тестування є його спроможність полегшити розробку через ізоляцію та перевірку окремих фрагментів коду. Кожен модуль тестується незалежно, що дозволяє виявляти та виправляти помилки в окремих частинах програми без впливу на решту системи, що сприяє підвищенню надійності програми та забезпечує швидше виявлення та усунення дефектів.

```

PASS src/utils/pdfFormatters/getFilteredPdfPageObjByHeight.test.ts (17.82 s)
PASS src/services/checkPagesFidelity/helpers/validateTitle.test.ts (17.931 s)
PASS src/services/checkReferenceOrder/helpers/checkStarrSymbols.test.ts (17.965 s)
PASS src/services/checkReferenceOrder/helpers/findReferenceMatches.test.ts
PASS src/services/checkPagesFidelity/helpers/definePageNum.test.ts
PASS src/helpers/getRefListPageNumber.test.ts
PASS src/services/checkFrameService/helpers/getFrameSizeConst.test.ts
PASS src/services/checkReferenceOrder/helpers/trimAndReplaceDashes.test.ts
PASS src/services/checkReferenceList/helpers/checkReferenceNumberFormat.test.ts
PASS src/services/checkReferenceOrder/helpers/checkOrder.test.ts
PASS src/services/checkFrameService/helpers/getCorrection.test.ts
PASS src/services/checkPagesFidelity/helpers/findIndexByMatch.test.ts
PASS src/utils/pdfFormatters/getPageView.test.ts
PASS src/services/checkReferenceOrder/helpers/checkIsUsed.test.ts
PASS src/services/checkPagesFidelity/helpers/findChapterByName.test.ts
PASS src/services/checkPagesFidelity/helpers/getIsNextSubItem.test.ts
PASS src/services/checkReferenceOrder/helpers/checkInsideReferenceSymbols.test.ts
PASS src/services/checkPagesFidelity/helpers/findPageMatchByCase.test.ts
PASS src/services/checkReferenceList/helpers/getIsElRes.test.ts
PASS src/services/checkReferenceList/helpers/checkReferenceElRes.test.ts
PASS src/services/checkFrameService/helpers/getFrameHeight.test.ts
PASS src/services/checkReferenceList/helpers/getListFromStringRows.test.ts
PASS src/services/checkPagesFidelity/helpers/getChapterNumber.test.ts
PASS src/utils/stringFormatters/index.test.ts
PASS src/utils/pdfFormatters/helpers/removePageNumberRow.test.ts
PASS src/services/checkPagesFidelity/helpers/checkChapterItem.test.ts
PASS src/services/checkReferenceList/helpers/getIsValidRefNumber.test.ts
PASS src/services/checkReferenceOrder/helpers/getTransformedReference.test.ts
PASS src/services/checkReferenceOrder/helpers/checkAfterReferenceSymbols.test.ts
PASS src/services/checkPagesFidelity/helpers/concatTitle.test.ts
PASS src/services/checkReferenceList/helpers/checkListLength.test.ts
PASS src/utils/pdfFormatters/getPageString.test.ts
PASS src/utils/pdfFormatters/getPageStringRows.test.ts

Test Suites: 33 passed, 33 total
Tests: 113 passed, 113 total
Snapshots: 0 total
Time: 29.859 s, estimated 57 s
Ran all test suites.

```

Рис. 3.8 Модульне тестування сервісу

Згідно з результатами модульного тестування помилок в роботі сервісу також не знайдено.

Після отриманих результатів можна підсумувати, що модульне тестування швидше за ручне, адже юніт тести перевіряють сервіс за лічені секунди. Але приймаючи до уваги, що не всі програмісти мають профільну освіту, досвід та не всі знайомі з написанням тестів ручне тестування простіше, але довше за часом. Найкраще поєднувати обидва види тестування для отримання більш точних результатів та навичків написання юніт тестів.

3.2 Оцінка ефективності наукової та технологічної складової

Ключовим завданням сьогодення є пошук та впровадження ефективних інструментів, які сприяють підвищенню продуктивності роботи студентів та викладачів. У цьому контексті система перевірки дипломних робіт виступає як інноваційний засіб автоматизації навчального процесу, що має потенціал

значно полегшити та покращити роботу основних учасників освітнього процесу.

Перш за все, важливо визначити, що такий сервіс передбачає комплекс програмних та технічних інструментів, спеціально розроблених для автоматизації процесу перевірки дипломних робіт студентів.

Застосунок виконує аналіз тексту робіт набагато швидше, ніж це може зробити викладач, дозволяючи ефективніше використовувати ресурси закладу вищої освіти, що особливо актуально в умовах зростання числа студентів та обсягу робіт.

За даними Міністерства освіти й науки України, у 2022 році з усіх закладів вищої освіти України було загалом 63 671 випускники, з них понад 1000 випускники університету «Житомирська Політехніка».

На одного наукового керівника, в середньому, виділяють від 5 до 30 студентів. Обсяг роботи бакалавра становить 50 - 70 сторінок (без урахування додатків), а магістра 80 - 100. Якщо у керівника 5 робіт рівня бакалавр і в кожного від 50 до 70 сторінок (без урахування титульної сторінки, змісту, додатків), то це вже 250 - 350 сторінок до перевірки. Зазвичай, на розбір та перевірку роботи бакалавра потрібно від 10 до 20 годин. Однак, у деяких випадках, цей час може бути більшим або меншим. І це тільки одна робота. А коли їх більше, то години, витрачені на перевірку можуть доходити до неймовірної цифри 100 [43].

Приймаючи до уваги факт, що у викладача є ще багато різних обов'язків, виходить, що наукові керівники мають витрачати власний час на перевірку робіт студентів, що зовсім не економічно з точки зору енергоресурсу. А якщо брати магістерські роботи, то при наявності 4 - 5 студентів науковому керівнику треба перечитати в середньому 400 - 500 сторінок друкованого шрифту [43].

В середньому, на прочитання і перевірку магістерської роботи потрібно від 20 до 40 годин, але даний час може варіюватися від обсягу роботи студента.

Таким чином, на ручну перевірку 5 магістерських робіт у наукового керівника піде приблизно 200 годин. В середньому у наукового керівника може бути від 5 до 30 випускників різних наукових ступенів, що дуже затратно по часу [43].

При використанні сервісу перевірки дипломних робіт у викладача займе до 2 хв часу на перевірку однієї роботи. Таким чином, за 5 хвилин науковий керівник може перевірити 5 академічних робіт різних наукових ступенів, включаючи аспірантські роботи [43]. Все що треба зробити – завантажити необхідний файл та запустити перевірку, яка протягом кількох секунд повідомить результат (рис. 3.9).

Таким чином, на перевірку 30 академічних робіт різного ступеня у викладача піде не більше 25 хв, що надзвичайно зручно в порівнянні з ручною перевіркою.

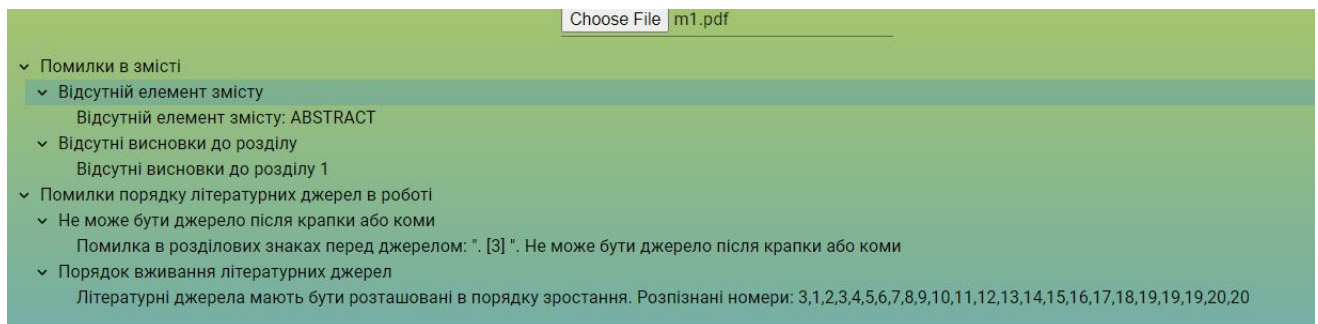


Рис. 3.9 Автоматизована перевірка дипломної роботи за допомогою сервісу

У випадку, якщо помилки в роботі відсутні, жодного повідомлення сервіс виводити не буде (рис. 3.10).

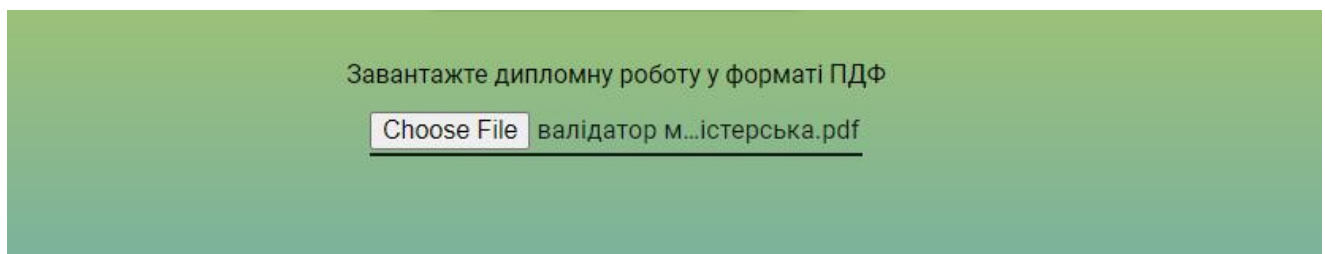


Рис. 3.10 Результат перевірки академічної роботи без помилок

З метою порівняння ручної та автоматизованої перевірки дипломних робіт було створено таблицю (табл. 3.1).

Задля розробки сервісу було проведено аналіз існуючих бібліотек та обрано для роботи pdfjs-dist. Деталі дослідження бібліотек докладно описано в розділі 1.3.

Таблиця 3.1

Порівняння ручної та автоматизованої перевірки дипломних робіт

Критерій	Ручна перевірка	Автоматизована перевірка
Час	Високо затратна за часом	Низько затратна за часом
Точність	Залежить від досвіду та кваліфікації викладача	Високо точна
Об'єктивність	Залежить від суб'єктивних факторів	Об'єктивна
Можливість виявлення плагіату	Низька	Висока
Можливість виявлення помилок в оформленні	Низька	Висока
Можливість виявлення невідповідності змісту поставленій темі	Низька	Висока

Для оцінки наукової складової варто зазначити, що в розділі 1.4 було досліджено наявні підходи до аналізу та результату перевірки дипломних робіт.

Новітнім та інноваційним рішенням в цьому контексті стало розроблення власних алгоритмів, які максимально точно та ефективно забезпечують виконання поставлених завдань. Алгоритми не лише враховують всі потрібні параметри перевірки, а й відповідають вимогам сучасної академічної спільноти. В результаті впровадження алгоритмів для перевірки правильності оформлення літературних джерел згідно ВАК України, правильності розташування джерел в тексті та в бібліографії та перевірку відповідності елементам змісту в змісті та в документі, можна забезпечити якісніший та об'єктивний аналіз дипломних робіт, що є важливим кроком у підвищенні рівня наукової діяльності та дотримання високих стандартів академічної доброчесності.

3.3 Оцінка ефективності сервісу

Після розробки та тестування важливо оцінити ефективність отриманого сервісу. Як бачимо з результатів тестування, застосунок для перевірки академічних робіт досить точно визначає помилки та видає звіт. Відтепер студентські роботи будуть перевірятися набагато швидше, ніж зазвичай. Керівнику дипломної роботи достатньо відкрити програму, завантажити необхідний файл та за кілька секунд звіт по роботі буде готово. Таким чином, у викладача займе всього годину перевірка 50 робіт, що значно економить людський час та ресурси.

При багатогодинній перевірці людина може не побачити найдрібніших недоліків у вимогах, таких як: крапка після цифри в підрозділі, а наш сервіс знайде та видасть вищезгадану помилку.

Якщо згадати описані в першому розділі аналоги та порівняти їх з нашим сервісом, ми побачимо надзвичайні переваги в новому застосунку, адже він надає звіт з виконання вимог в оформленні згідно з університетом «Житомирська політехніка». Існуючі до цього аналоги перевіряють лише орфографію та правопис, що не економить час так, як наш застосунок.

Задля кращої оцінки ефективності сервісу, перевіримо дану записку до кваліфікаційної магістерської роботи на правильність оформлення за допомогою розробленого застосунку (рис. 3.11).

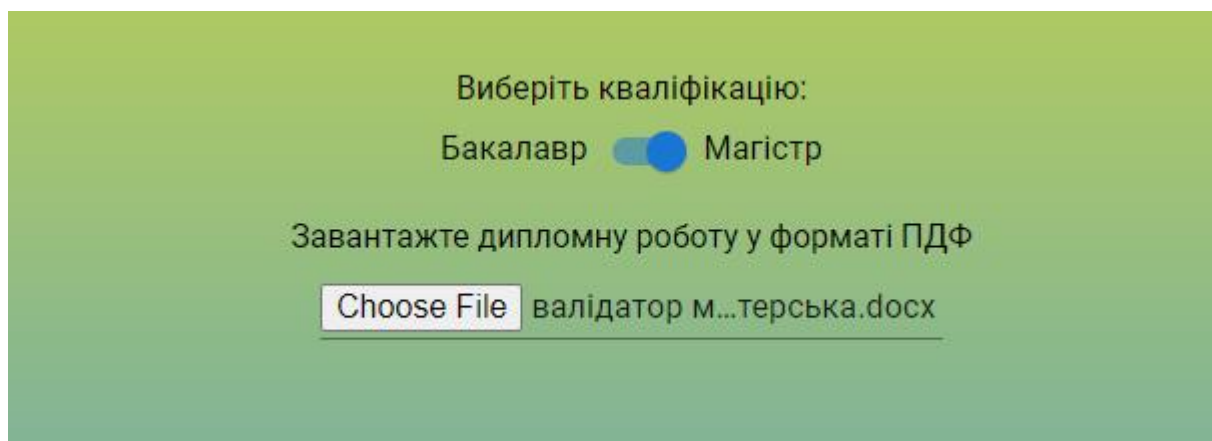


Рис. 3.11 Перевірка кваліфікаційної записки магістра за допомогою сервісу перевірки дипломних робіт

Варто зазначити, що на дану перевірку було витрачено менше хвилини, але отримано якісні результати та впевненість в правильності оформлення.

При наявності помилок в оформленні у студента або викладача може піти декілька годин на виявлення помилок, а за допомогою сервісу процес перевірки буде займати менше часу. (рис. 3.12)

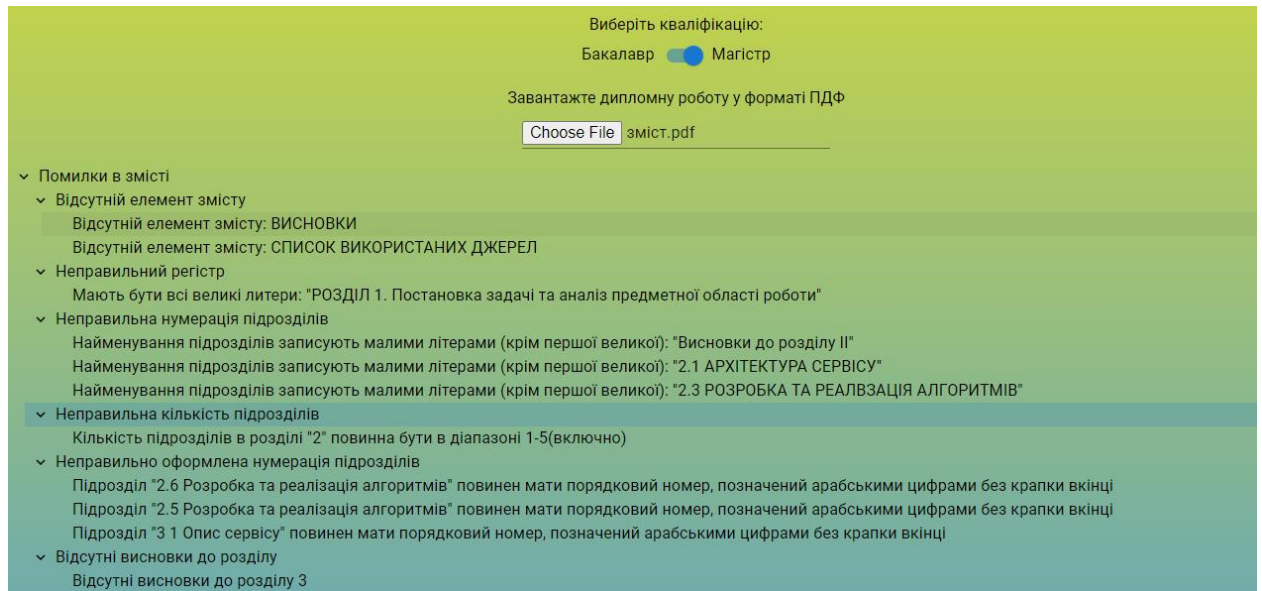


Рис. 3.12 Перевірка дипломної роботи на помилки в змісті

Як бачимо на рисунку вище, зміст роботи містить багато помилок і для їх знаходження треба до пів години. Наш застосунок здійснив перевірку та видав результат за 30 секунд. Варто звернути увагу на те, що у тестованому файлі помилки в оформленні існують лише в змісті, а якщо вся дипломна міститиме помилки з різних пунктів оформлення, часу на ручну перевірку необхідно ще більше. При перевірці дипломної роботи з великою кількістю помилок за допомогою сервісу було витрачено менше хвилини (рис. 3.13).

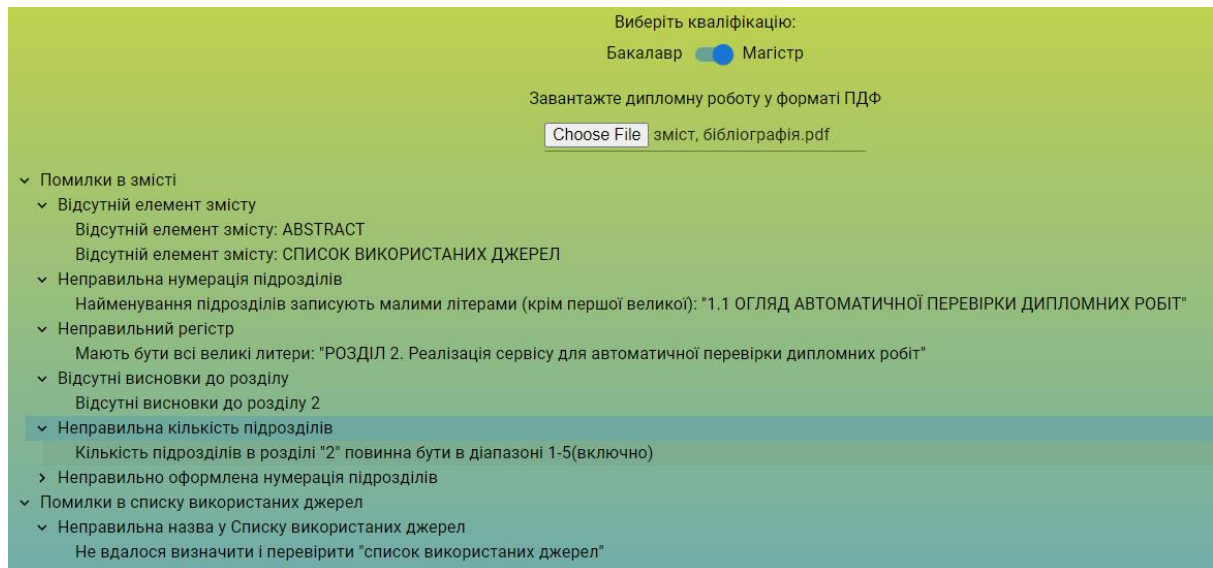


Рис. 3.13 Перевірка дипломної роботи на помилки в оформленні за допомогою сервісу

Оцінка ефективності нового сервісу може бути пов'язана з показником рентабельності власного капіталу (ROE) в контексті збалансованості витрат та результатів.

ROE - це показник, що вказує, наскільки ефективно використовується власний капітал, тобто скільки прибутку було згенеровано на кожну гривню залучених власних коштів [44].

Оцінка ефективності інноваційного сервісу перевірки дипломних робіт може бути пов'язана з показником ROE в контексті збалансованості витрат та результатів. Введення застосунку дозволяє економити час і ресурси, що може вплинути на покращення показників ефективності бізнесу.

Можна розглядати витрати на перевірку студентських робіт, як витрати компанії у вигляді робочих годин викладачів. Оскільки розроблений сервіс дозволяє зменшити ці витрати, то це може позитивно позначитися на показнику ROE, оскільки чистий прибуток залишається практично незмінним, а витрати зменшуються.

Підсумовуючи оцінку перевірки варто зазначити, що розробка інноваційного сервісу з перевірки дипломних робіт значно полегшить життя викладачів та студентів максимально скорочуючи час, витрачений на

перевірку оформлення згідно з вимогами. В майбутньому даний сервіс може бути покращений з додаванням нового функціоналу та розширенням можливостей застосунку.

Висновки до розділу 3

Даний розділ присвячений обговоренню ефективності та тестуванню сервісу з перевірки дипломних робіт на правильність оформлення згідно з вимогами.

Було описано процес ручного та модульного тестування, всі результати представлені у вигляді скріншотів. Згідно з отриманими результатами, сервіс працює якісно, без помилок та достатньо швидко, що надзвичайно важливо для викладачів та студентів.

Обговорювалась оцінка внеску наукових та технологічних аспектів у розвиток сервісу. Аналіз містить ефективність використаних технологій, адаптабельність до змін та забезпечення високого рівня наукової обґрунтованості.

Проведено оцінку загальної ефективності створеного сервісу. Враховані критерії включають швидкість роботи, точність перевірки, зручність використання для користувачів.

ВИСНОВКИ

У даній магістерській роботі описано розробку сервісу для перевірки дипломних робіт у форматі PDF.

При написанні сервісу було проведено огляд автоматичної перевірки дипломних робіт, обрано та обґрунтовано мову програмування для розробки застосунку, проаналізовано існуючі бібліотеки для роботи з файлами PDF з метою обрання найбільш підходящої. В роботі описано архітектуру сервісу, а також основні вимоги та реалізацію інноваційних алгоритмів перевірки на правильність оформлення списку використаних джерел та перевірки на відповідності змісту сторінкам і заголовкам в роботі. Крім того, було протестовано сервіс на коректну та якісну роботу, а також обговорено ефективність застосунку, наукової та технологічної складової.

У результаті розробки було досягнуто поставленої мети - створено сервіс для автоматизованої перевірки дипломних робіт. Специфікація функцій включає перевірку правильності оформлення джерел бібліографії згідно ВАК України, розміщення джерел у документі та в бібліографії в порядку зростання, а також перевірку елементів змісту в документі та в бібліографії.

За отриманими результатами можна визначити, що сервіс успішно виконує поставлені завдання, забезпечуючи користувачам детальний звіт про виявлені недоліки. Практична цінність даного сервісу полягає в наданні користувачам зручного та ефективного інструменту для автоматичної перевірки дипломних робіт на наявність помилок.

Однак важливо зауважити, що подальший розвиток сервісу може охоплювати застосування штучного інтелекту для автоматизації додаткових завдань. Важливим аспектом може стати впровадження інструментів машинного навчання для аналізу структури аргументації та виявлення сильних та слабких сторін роботи. Інтеграція із засобами аналізу мовленнєвих особливостей може сприяти покращенню стилістики та загальної якості

написання. Можливість перевірки орфографії, відповідність тексту вимогам оформлення (абзацні відступи, вирівнювання за шириною, шрифт), а також стандартизація структури робіт (вирівнювання розділів та підрозділів) не лише підвищать ефективність сервісу, але й сприятимуть збільшенню якості наукових робіт.

Вищезгадані інновації сприятимуть не лише ефективності сервісу, але і його адаптації до змін вимог до наукових досліджень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Програмне забезпечення для перевірки наукових текстів на плагіат: інформаційний огляд / автори-укладачі: А. Р. Вергун, Л. В. Савенкова, С. О. Чуканова ; редколегія: В. С. Пашкова, О. В. Воскобойнікова-Гузева, Я. Є. Сошинська ; Українська бібліотечна асоціація. – Київ : УБА, 2016. – Електрон. вид. – 1 електрон. опт. диск (CDROM). – 36 с. – ISBN 978-966-97569-5-4.
2. Математичне та імітаційне моделювання систем. МОДС 2018 : тези доповідей Тринадцятої міжнародної науково-практичної конференції (Чернігів, 25 - 29 червня 2018 р.) / М-во освіти і науки України, Нац. Акад. наук України, Академія технологічних наук України, Інженерна академія України та ін. – Чернігів : ЧНТУ, 2018. – 392 с.
3. G. E. Moore, «Cramming More Components Onto Integrated Circuits,» in *Proceedings of the IEEE* // vol. 86, no. 1, pp. 82-85, Jan. 1998, doi:10.1109/JPROC.1998.658762.
4. Кучерук О. В., Желізко В.В., Савіцький Р.С. Система перевірки дипломних робіт як інструмент автоматизації роботи студента та викладача // редакційний журнал «Житомирська політехніка»(подано до друку) – Житомир.
5. Кучерук О. В., Вакалюк Т.А. Сучасні інформаційні системи та технології // тези V Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення», м. Житомир, 01–02 грудня
6. Желізко В. В., Кучерук О.В., Савіцький Р.С. Парсинг PDF файлів // Теоретичні та практичні дослідження молодих вчених : тези доповідей Міжнародної науково-практичної конференції, Online, 28 – 30 листопада 2023 / МОН України, Харківський політехнічний інститут, 2023. – С. 360.
7. JavaScript documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.

8. Аспекти статичної типізації [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://foxminded.ua/statychna-typizatsiia/>.
9. TypeScript for JS Programmers [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>.
10. Формат файлу PDF [Електронний ресурс] / документація – Режим доступу до ресурсу: <https://docs.fileformat.com/uk/pdf/>.
11. PDFKit documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://pdfkit.org/>.
12. PDFMAKE documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://pdfmake.github.io/docs/0.1/>.
13. jsPDF documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/parallax/jsPDF/blob/master/README.md>.
14. Puppeteer documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/puppeteer/puppeteer/blob/main/README.md>.
15. pdf-lib documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/pdf-lib?activeTab=readme#documentation>.
16. pdf-parse documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/pdf-parse>.
17. pdfjs-dist documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/pdfjs-dist>.
18. Shai S. Understanding Machine Learning: From Theory to Algorithms / S. Shai, B. Shai., 2014. – 449 с. – (Cambridge University Press).
19. Гаврюшин Д. Криптосемантичний метод захисту текстової інформації [Електронний ресурс] / Д. Гаврюшин, В. Чуприн. – 2023. – Режим доступу до ресурсу: https://dspace.nau.edu.ua/bitstream/NAU/60088/1/%D0%A4%D0%90%D0%95%D0%A2_172_2023_%D0%B4%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC

[%D0%93%D0%B0%D0%B2%D1%80%D1%8E%D1%88%D0%B8%D0%BD%20%D0%94.%D0%AE..pdf.](#)

20. Білоус Р.В., Крилов Е.В., Анікін В.К. The concept of deep learning: recognizing elements in cartographic images // Міжвідомчий науково-технічний збірник «Адаптивні системи автоматичного управління», 2020. – № 2(37). – с.117.

21. Основні поняття алгоритмістики [Електронний ресурс] – Режим доступу до ресурсу:
<https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/16312/1/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%96%D1%81%D1%82%D0%B8%D0%BA%D0%B0.%20%D0%9B%D0%B5%D0%BA%D1%86%D1%96%D1%97.pdf>.

22. Гранік Михайло Обґрунтування доцільності використання глибинного навчання для розв'язання задач розпізнавання неправдивих новин [Текст] / М. Гранік, В. Месюра // Proceedings of the eleventh international scientific-practical conference INTERNET-EDUCATION-SCIENCE - 2018, Vinnytsia, 22 - 25 May, 2018. – Vinnytsia : VNTU, 2018. – С. 202-203.

23. Основи алгоритмізації та програмування: курс лекцій. Частина 1 методичні рекомендації для студентів спеціальності 014 Середня освіта (Інформатика) Луцьк: Вежа-друк, 2022. 36 с.

24. Крєневич А.П. Алгоритми і структури даних. Підручник. – К.: ВПЦ "Київський Університет", 2021. – 200 с.

25. Костів О. В., Ярошко С. А. Методи розробки алгоритмів: Тексти лекцій. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2002. – 96 с.

26. Основи алгоритмізації та програмування: курс лекцій. Частина 1 методичні рекомендації для студентів спеціальності 014 Середня освіта (Інформатика) Луцьк: Вежа-друк, 2022. 36 с.

27. Бандоріна Л.М., Климкович Т.О., Удачина К.О. Основи алгоритмізації та програмування : навч. посібник. УДУНТ, 2022. 158 с.

28. Конспект лекцій з дисципліни «Архітектура та проектування програмного забезпечення» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інженерія програмного забезпечення» із спеціальності 121 – «Інженерія програмного забезпечення» / Укл. В.В. Завгородній, К.М.Ялова. – Кам'янське: ДДТУ, 2019. – 144с.

29. What is SOA (Service-Oriented Architecture) [Електронний ресурс] – Режим доступу до ресурсу: <https://aws.amazon.com/what-is/service-oriented-architecture/>.

30. Архітектура клієнт-сервер [Електронний ресурс] – Режим доступу до ресурсу: <http://inter.ptngu.com/kompyuterni-merezhi/arhitektura-kliiyent-server>.

31. Архітектура клієнт-сервер [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/>.

32. Цибульник, С. О., Барандич К.С. Технології розроблення програмного забезпечення. Частина 1. Життєвий цикл програмного забезпечення // підручник закладу вищої освіти – ред. Павловський О.М – Київ : КПІ ім. Ігоря Сікорського – 2022 – с. 270.

33. Kouraklis J. MVV as design pattern [Електронний ресурс] / John Kouraklis. – 2016. – Режим доступу до ресурсу: https://www.researchgate.net/publication/310493153_MVVM_as_Design_Pattern.

34. Григор'єв Д. О. Інформаційна система для роботи з тегами ID3v2 для mp3 файлів. Інтерфейс користувача з використанням технології WPF та патерну MVVM, робота з файловими ієрархіями. [Електронний ресурс] / Даниїл Олександрович Григор'єв. – 2019. – Режим доступу до ресурсу: <https://krs.chmnu.edu.ua/jspui/bitstream/123456789/628/1/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20%D0%93%D1%80%D0%B8%D0%B3%D0%BE%D1%80%D1%8C%D0%B5%D0%B2%20%D0%94.%D0%9E%20403.pdf>.

35. Токаренко О. В. Система збору експериментальних даних на основі мікросервісної архітектури [Електронний ресурс] / Олександр Володимирович Токаренко. – 2020. – Режим доступу до ресурсу: https://ela.kpi.ua/bitstream/12310789/38271/1/Tokarenko_magistr.pdf.
36. Мікросервісна архітектура для початківців. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/>.
37. Желізко В. В. Розробка сервісів первинної перевірки пояснювальних записок до дипломних робіт / Віктор Вікторович Желізко. – 2023.
38. ДСТУ гост 7.1:2006. Бібліографічний запис, бібліографічний опис. загальні вимоги та правила складання (33995).
39. Випускна кваліфікаційна робота(ІІЗ+УІТІІ). Рівень: магістр [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://learn.ztu.edu.ua/pluginfile.php/229117/mod_resource/content/1/%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BD%D1%96%20%D0%B2%D0%BA%D0%B0%D0%B7%D1%96%D0%B2%D0%BA%D0%B8%20%D0%B4%D0%BE%20%D0%B0%D1%82%D0%B5%D1%81%D1%82%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%BE%D1%97%20%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B8.pdf.
40. Дідковська М. В., Тимошенко Ю.О. Тестування: Основні визначення, аксіоми та принципи. Текст лекцій до курсу «Технології розробки і тестування програм» національного технічного університету України «Київський політехнічний інститут» – 2010. Частина І. С. 62.
41. Василевський В. О., Романюк О.В. Недоліки використання модульного тестування як основної технології тестування // Тези доповідей ХІ Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології – 2020 (ІКТ-2020)», м. Житомир, 09 - 11 квітня 2020 р. – Житомир: Житомирська політехніка, 2020. – 252 с.

42. Конспект лекцій з навчальної дисципліни «Методологія тестування програмного забезпечення» / Укладач Старух А.І. – Львів: Львівський національний університет імені Івана Франка, 2020. – с.3.

43. Кучерук О. В., Желізко В.В., Савіцький Р.С. Система перевірки дипломних робіт як інструмент автоматизації роботи студента та викладача // редакційний журнал «Житомирська політехніка»(подано до друку) – Житомир.

44. Кобилецький В. Р., Коефіцієнт рентабельності власного капіталу / В. Р. Кобилецький // Онлайн-журнал «Financial Analysis online» [Електронний ресурс] – Режим доступу до ресурсу: <https://analizua.com/slovník-ekonomichnikh-terminiv/338-pokaznik-rentabelnosti-vlasnogo-kapitalu>