НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

До захисту допущено:
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім'я, прізвище)
"___"_____2021 р.

Дипломний проєкт
на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: « *Інформаційна система з визначення близькості документів на основі стиснення текстів* »

Виконала: студентка *IV* курсу, групи *ІС-73*

Шеляхіна Владислава Владиславівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник *доц., к.т.н., доц. Фіногенов Олексій Дмитрович* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Сперкач Майя Олегівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент *доц., к.т.н., доц. Філіппова Марина В'ячеславівна* _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2021 року

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

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2021 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Шеляхіній Владиславі Владиславівні
(прізвище, ім’я, по батькові)

1. Тема проєкту « *Інформаційна система з визначення близькості документів на основі стиснення текстів* »,
керівник проєкту *Фіногенов Олексій Дмитрович, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*11*” *05* 2021 р. № 1139-с

2. Термін подання студентом проєкту “*04*”*червня* 2021 року

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання
4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів
5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна діяльності

2. Схема структурна варіантів використання

3. Схема структурна послідовності

4. Схема структурна компонентів

5. Креслення вигляду екранних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання *«7» квітня 2021 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*18.04.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*20.04.2021*	
3.	*Постановка та формалізація задачі*	*21.04.2021*	
4.	*Розробка інформаційного забезпечення*	*22.04.2021*	
5.	*Алгоритмізація задачі*	*26.04.2021*	
6.	*Обґрунтування використовуваних технічних засобів*	*29.04.2021*	
7.	*Розробка програмного забезпечення*	*10.05.2021*	
8.	*Налагодження програми*	*12.05.2021*	
9.	*Виконання графічних документів*	*13.05.2021*	
10.	*Оформлення пояснювальної записки*	*14.05.2021*	
11.	*Подання ДП на попередній захист*	*15.05.2021*	
12.	*Подання ДП на основний захист*	*04.06.2021*	
13.	*Подання ДП рецензенту*	*07.06.2021*	

Студент Владислава ШЕЛЯХІНА

Керівник Олексій ФІНОГЕНОВ

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 7325.00.000 ПЗ	Пояснювальна записка	80	
3	А4	ДП 7325.01.000 ТЗ	Технічне завдання	8	
4	А4	ДП 7325.02.000 ССД	Схема структурна діяльності	1	
5	А3	ДП 7325.03.000 ССВ	Схема структурна варіантів використань	1	
6	А3	ДП 7325.04.000 ССП	Схема структурна послідовності	1	
7	А3	ДП 7325.05.000 СК	Схема структурна компонентів	1	
8	А3	ДП 7325.06.000 КЕ	Креслення вигляду екранних форм	3	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 7325 00.000.00		
Розробн.		Шеляхіна В.В.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Фіногенов О.Д.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-73	
Н/контр.		Сперкач М.О.					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: *Інформаційна система з визначення близькості документів на основі стиснення текстів*

Київ – 2021 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 25 рисунків, 9 таблиць, 1 додаток, 18 джерел.

Дипломний проєкт присвячений розробці інформаційної системи з визначення близькості документів на основі стиснення текстів.

Призначенням розробки є підтримка процесу визначення близькості документів з попереднім їх стисненням.

Мета розробки полягає у підвищенні ефективності порівняння текстів за рахунок створення їх стиснутих копій (авторефератів).

У розділі інформаційного забезпечення здійснюється опис первісних реквізитів та вихідних сигналів системи, здійснюється детальний опис масивів інформації.

Розділ математичного забезпечення присвячений пошуку та обґрунтуванню методів вирішення сформульованої постановки задачі.

Програмне забезпечення описує обрані засоби розробки та спроектовану архітектуру програмного забезпечення.

У технологічному розділі надається керівництво користувача та випробування програмного продукту.

КЛЮЧОВІ СЛОВА: АВТОМАТИЧНЕ РЕФЕРУВАННЯ, АЛГОРИТМ, БЛИЗЬКІСТЬ, ДОКУМЕНТ, ІНФОРМАЦІЯ, ОБРОБКА, ПОДІБНІСТЬ, СТИСНЕННЯ, ТЕКСТ.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of five sections, contains 25 figures, 9 tables, 1 application and 18 sources.

The diploma project is devoted to the development of an information system for determining the similarity of documents based on the compression of texts.

The purpose of the development is to support the process of determining the similarity of documents with their prior compression.

The aim of the development is to increase the efficiency of comparing texts by creating their abstracts.

In the section of information support, the description of initial requisites and output signals of the system is carried out, the detailed description of arrays of information is carried out.

The section of mathematical support is devoted to the search and substantiation of methods for solving the formulated problem statement.

The software describes the selected development tools and the designed software architecture.

The technology section provides a user guide and software tests.

KEY WORDS: AUTOMATIC TEXT SUMMARIZTION, ALGORITHM, SIMILARITY, DOCUMENT, INFORMATION, PROCESSING, HOMOGENEITY, COMPRESSION, TEXT.

ЗМІСТ

ВСТУП

Поява перших електронно-обчислювальних машин у середині минулого століття та стрімкий розвиток таких розділів математики як теорія алгоритмів та математична логіка сприяли виникненню великої кількості нових наук. Однією з таких наук стала математична лінгвістика – науковий та інженерний напрямок, пов’язаний із розумінням природньої мови з обчислювальної точки зору та створенням математичних моделей [1]. Зацікавленість в лінгвістичній обробці текстів була зумовлена наступними значимими факторами: експоненційним збільшенням обсягів текстової інформації у повсякденному житті та бажанням автоматизації трудомістких процесів.

Визначення близькості текстів наразі є одним із підрозділів сучасної математичної лінгвістики. Через невпинний розвиток комп’ютерно-комунікаційних технологій і, насамперед, мережі Інтернет, використання об’єктів авторського права стає все більш розповсюдженим явищем. Враховуючи стрімкий ріст потоків інформації, проблема виявлення плагіату та захисту авторських прав набуває особливого значення. Так як час та складність опрацювання текстів прямо пропорційні до їх об’єму, дана проблема не може бути вирішена без використання обчислювальних потужностей.

Сьогодні алгоритмам NLP, які використовують комп’ютери для обробки природніх мов [2], приділяється значна увага розвинених країн Європи, Азії та Америки. Про це свідчить постійна розробка нових технологій та виділення коштів організаціям, які займаються науковою діяльністю у сфері комп’ютерної лінгвістики. Проте, не дивлячись на те, що розробка математичних алгоритмів виявлення близькості природномовних текстів триває вже протягом століття, методу, який даватиме кращий результат, досі не існує.

Практичне значення одержаних результатів. Результатом розробленого програмного забезпечення є сервіс, задача якого полягає у

використанні алгоритмів стиснення для подальшого порівняння завантажених текстових документів та формуванні звіту виявлення близькості документів (плагіату) з можливістю його збереження.

Публікації.

Результати роботи були опубліковані в тезах доповідей:

Шеляхіна В.В. Технологія визначення близькості документів на основі стиснення текстів. *VI всеукраїнська науково-практична конференція молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2021)* [3].

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Ефективне виявлення морфологічної та семантичної подібності масивів інформації має неабияку практичну цінність для людства. Наразі алгоритми визначення близькості текстів використовуються майже у всіх сферах людської діяльності, аби полегшити виконання таких задач, як: пошук інформації, її вилучення, семантичний аналіз, класифікація та впорядкування, надання рекомендацій, машинний переклад, обробка природніх мов, кластеризація документів, генерація запитань та відповідей, виявлення плагіату тощо [4, 5].

Визначення близькості базується на оцінці схожості токенів, або, іншими словами, слів, між собою. Виявлення семантично пов’язаних, тобто близьких за сенсом, слів під час порівняння текстів є комплексною проблемою [6], хоча і дозволяє підвищити якість обробки текстової інформації у порівнянні із визначенням лексичної схожості, яка означає схожу послідовність літер. Так, слова з високою оцінкою морфологічної близькості, можуть взагалі не мати семантичної схожості і навпаки.

Далі наведено три основні підходи для визначення близькості між словами, реченнями, параграфами та документами, які були описані Ваелем Гомаа у [7].

Методи, засновані на підході рядкової подібності, оцінюють схожість та несхожість двох текстових рядків для приблизного співпадіння рядків чи порівняння. Алгоритми, основані на обробці символів, трансформують та виділяють з оригінального тексту підпослідовності символів визначеної довжини, та застосовують до них різноманітні формули підрахунку степеню збігу. Алгоритми, які беруть за основу терміни, можуть використовувати евристику для обчислення схожості.

Група методів корпусної подібності встановлює близькість між словами за допомогою колекцій текстів.

Методи, в основі яких лежать знання, оцінюють семантичну схожість двох слів використовуючи інформацію про лексичні одиниці, отриманої з лексичних баз даних мов. Дані алгоритми задають чіткі межи встановлення смислових зв'язків між словами, так як зазвичай дані оцінки не можуть бути визначені однозначно.

Як правило, під час перевірки текстових документів на схожість, використовується повний обсяг тексту. Проте із збільшенням об'ємів масивів інформації знижується точність роботи алгоритму, а також підвищуються час обробки та вимоги до обчислювальних ресурсів машини. Це спричиняє неабиякі труднощі для проведення лінгвістичних аналізів.

В більшості документів автори додатково вказують анотацію – осмислену вибірку найбільш значущого тексту. Так постає питання про те, чи можливо використовувати для порівняння текстових документів фрагменти тексту, які передають головну суть в межах оригінального змісту.

Таким чином проведено дослідження, яке полягає у визначенні ефективності виявлення подібності між текстами із попереднім стисненням, використовуючи алгоритми автоматичного реферування.

Створена система може бути використана для завдань галузей, які потребують вирішення задач, що пов'язані із захистом об'єктів авторського права, у дослідженнях науковців у галузі математичної лінгвістики, пошукових системах тощо.

1.1.1 Опис процесу діяльності

Програмне забезпечення реалізовано у вигляді вебдодатку.

У розробленій системі присутній один актор – Користувач. Він же і приймає безпосередню участь у процесі діяльності системи. Процес

використання вебзастосунку Користувачем можна представити діаграмою діяльності, наведеної у графічному матеріалі.

У вебзастосуванні представлені три пункти меню: «Головна», «Про сервіс» та «Історія запитів».

Перша сторінка, з якою знайомиться користувач, є пункт меню «Про сервіс». На цій сторінці Користувач отримує коротку інформацію про призначення інформаційної системи та ознайомлюється із покроковою інструкцію роботи із програмою. Він має можливість одразу перейти до сторінки завантаження документів для їх обробки.

Пункт меню «Історія запитів» - це випадаючий список, що містить інформацію про останні запити на обробку. Користувач може переглянути історію запитів у будь-який момент роботи із сервісом, не покидаючи сторінку, на якій знаходиться.

У розділі меню «Головна» розміщений основний функціонал системи. Обробка документів проходить у чотири етапи.

На першому етапі Користувач має завантажити локальні документи (у форматі *.txt) чи ввести власноруч тексти. Завантаження документів відбувається через відкриття програми «Провідник» або функції «Drag and Drop». Введені власноруч тексти додаються при натисканні на кнопку «Завантажити».

Завантажені файли додаються чи видаляються поодинці. Для переходу на наступний етап необхідне завантаження двох текстів та натискання кнопки «Продовжити».

На другому етапі Користувач проводить налаштування майбутнього звіту. Передбачена можливість повернутися на перший етап, якщо були завантажені помилкові документи.

Після натискання кнопки «Продовжити» починається третій етап – обробка документів, час якої залежить від розміру вхідних даних. Починаючи з цього етапу, Користувач більше не може редагувати свій запит.

Останній етап передбачує отримання згенерованого звіту. Поля звіту можуть відрізнятися в залежності від обраних налаштувань. Якщо на етапі налаштування було ввімкнена «Перевірка на основі стиснення», то з'являється кнопка «Переглянути стиснені тексти». Натиснувши на цю кнопку, Користувач може переглянути інформацію, отриману під час роботи алгоритму автоматичного реферування. Якщо Користувач натискає «Завантажити звіт у PDF», то на вихід програма видає результати аналізу у вигляді файлу у форматі PDF.

1.1.2 Опис функціональної моделі

Особливості функціональної поведінки системи представлено у вигляді варіантів використання.

Визначимо авторів (дійових осів системи) для проектування структурної схеми варіантів використання та їх дії.

Актором системи є Користувач. Даний актор надає вхідні дані системі та обирає необхідні йому результати аналізу текстової інформації. Структурну схему варіантів використання представлено у графічному матеріалі.

Варіанти використання:

- перегляд інформації про систему;
- введення текстів власноруч;
- завантаження текстових документів;
- зміна способу надання текстів на обробку;
- видалення текстів;
- проведення налаштувань;
- отримання звіту;
- завантаження звіту;
- перегляд стиснених текстів;
- перегляд історії запитів.

Визначимо функціональні вимоги та пріоритети кожного з варіантів використання. У таблиці 1.1 наведений перелік усіх дій актора з коротким описом вимог та встановленим пріоритетом.

Таблиця 1.1 – Функціональні вимоги до варіантів використання

Актор	Варіант використання	Опис вимог до варіанту використання	Пріоритет
Користувач	Перегляд інформації про систему	RQ 001. Користувач повинен мати змогу ознайомитися з описом задачі сервісу та інструкцією.	Середній
	Введення текстів власноруч	RQ 002. Система має надавати можливість заповнити текстові поля назви тексту та його змісту.	Високий
	Завантаження текстових документів	RQ 003. Система має надавати можливість завантажити текстовий документ перетягуванням його у визначену область або використанням програми Провідник.	Високий

Продовження таблиці 1.1

Актор	Варіант використання	Опис вимог до варіанту використання	Пріоритет
Користувач	Зміна способу надання текстів на обробку	RQ 004. Користувач повинен бути спроможним обирати спосіб надання текстів на обробку.	Високий
	Видалення текстів	RQ 005. Система повинна надавати можливість видалити завантажені тексти	Середній
	Проведення налаштувань	RQ 006. Система має надавати змогу налаштувати обробку текстів.	Високий
	Отримання звіту	RQ 007. Користувач має мати можливість отримати та переглянути результат обробки текстів.	Високий
	Завантаження звіту	RQ 008. Користувач повинен бути спроможним завантажити згенерований звіт у форматі PDF.	Низький
	Перегляд стиснених текстів	RQ 009. Користувач повинен мати змогу отримати автореферати текстів.	Середній

Продовження таблиці 1.1

Актор	Варіант використання	Опис вимог до варіанту використання	Пріоритет
Користувач	Перегляд історії запитів	RQ 010. Користувач має отримувати доступ до останніх запитів, зробленим ним у системі.	Низький

1.2 Огляд наявних аналогів

Досліджень та програмних продуктів, які знаходять відсоток близькості текстів за допомогою алгоритмів автоматичного реферування не було знайдено. Крім цього, була виявлена відсутність у відкритому доступі якісних наборів необхідних даних українською мовою, а також бібліотек, які її підтримують. Тому у розділі були розглянуті програмні продукти, які окремо виконують функції автоматичного реферування та виявлення плагіату.

Система «Copyleaks»

Даний сервіс дозволяє виявляти невеликий відсоток схожості документів. Copyleaks не може виявити плагіат, якщо речення реструктуризовано або коли слово в реченні замінено.

Перевагами Copyleaks можна назвати широкий перелік підтримуваних форматів файлів, гнучкі налаштування, докладний звіт та невелику ціну. Недоліками є велика тривалість порівняння файлів, додавання документів у базу даних сервісу та некоректна робота з українськими текстами.

Представлений звіт програмою є добре зрозумілим. Інтерфейс представлений на рисунку 1.3.

Рисунок 1.3 – Інтерфейс вебсайту Copyleaks

Сервіс «Unicheck»

Unicheck є платним сервісом для виявлення плагіату з відкритих джерел мережі Інтернет чи внутрішньої бази даних. Використовується на рівні навчальних закладів та наукових установ.

Точність роботи алгоритму сервісу оцінюється високими шансами виявлення запозичених фрагментів за рахунок розпізнавання синонімів та знаходження перефразованих уривків.

Інтерфейс застосунку є дружнім та інтуїтивним (рис. 1.4).

Рисунок 1.4 – Інтерфейс застосунку Unicheck

Сервіс підтримує велику кількість форматів документів, дозволяє працювати із популярними хмарними сховищами і генерує вичерпний інтерактивний звіт. На жаль, тестування даного сервісу було ускладнене через занадто обмежений фунціонал безкоштовної онлайн-версії.

Сервіс «Resoomer»

Вебсайт та розширення для популярних браузерів під загальною назвою Resoomer дають змогу узагальнити тексти, статті, наукові тексти, а також структуровані художні твори, виділивши їх основні ідеї факти.

Сервіс стиснення Resoomer є частково безкоштовним, тому була змога протестувати лише власноруч введені тексти. З переваг можна виділити зрозумілий інтерфейс та можливість гнучкого налаштування узагальнення. Недоліком є підтримка лише англійської мови (рис. 1.5).

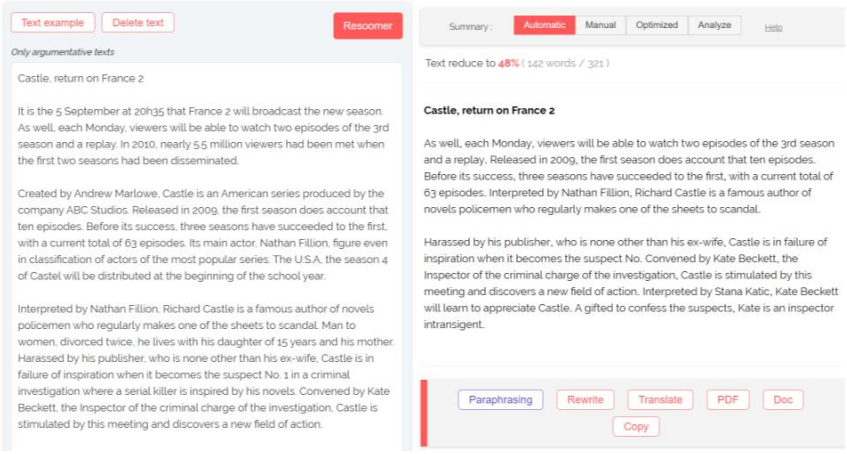


Рисунок 1.5 – Користувацький інтерфейс програми Resoomer

Система «Smmry»

Даний вебсайт має простий інтерфейс та велику кількість доступних налаштувань. Він дозволяє побудувати теплову карту отриманого на вхід тексту (який можна ввести власноруч, завантажити чи витягти з інтернет-ресурсу), у якій найважливіші речення виділяються найбільш насиченим кольором, а найменш важливі – навпаки (рис. 1.6).

Рисунок 1.6 – Інтерфейс вебсайту Smmry

Недоліком програми є підтримка файлів тільки із розширенням .doc, а також обробка лише англійських текстів.

Огляд аналогових рішень показав, що жоден з описаних вище програмних продуктів, які застосовують алгоритми стиснення текстової інформації, не підтримують обробку україномовних текстів.

В першу чергу розроблена система має на меті виправити головний недолік даних програм та додати підтримку української мови.

1.3 Постановка задачі

1.3.1 Призначення розробки

Даний проєкт може бути використаний для завдань галузей, які потребують вирішення задач, що пов’язані із захистом об’єктів авторського права, пошуком інформації, у дослідженнях науковців у галузі математичної лінгвістики, пошукових системах тощо.

Крім того, враховуючи недостатній функціонал наявних аналогів для роботи з текстами українською мовою, ця розробка може бути використана для створення авторефератів та перевірки україномовних текстів на близькість.

1.3.2 Цілі та задачі розробки

Цілі створення інформаційної системи з визначення близькості документів на основі стиснення текстів:

- підвищення швидкості виявлення схожості між текстами за рахунок зменшення обсягів текстів документів, які надходять на обробку;
- зниження вимог до необхідних обчислювальних ресурсів;
- автоматизація процесу створення авторефератів україномовних документів;
- автоматизація процесу перевірки україномовних документів на схожість;
- покращення ефективності алгоритмів, поєднуючи більш складні обчислювальні, статистичні та лінгвістичні гібридні методи виявлення.

Таким чином основною метою розробки системи є підвищення ефективності порівняння текстів за рахунок використання алгоритмів автоматичного реферування.

Для досягнення поставлених цілей необхідно реалізувати наступні задачі:

- проведення аналізу алгоритмів, що використовуються для стиснення та порівняння текстових документів;
- розробка логіки роботи програми;
- розробка користувацького інтерфейсу, що відповідає вимогам UI/UX;
- надання можливості формування звітів на основі аналізу документів.

Висновок до розділу

У ході написання даного розділу було проведено дослідження предметної області, доведена актуальність поставленої задачі, визначено цілі розробки сервісу та задачі, необхідні для їх досягнення та коректної роботи вебзастосунку з визначеними функціональними вимогами.

Надані опис предметного середовища, процесу діяльності, функціональної моделі системи. Додані діаграма варіантів використання та структурна схема діяльності користувача системи.

В ході аналізу існуючих аналогів, було досліджено ринок програмних продуктів, що використовують алгоритми визначення близькості та стиснення текстів, а також сформульовано переваги та недоліки кожного з них.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Для виконання автоматизованого створення авторефератів та перевірки їх на близькість, система використовує дані, які надходять від користувача. За допомогою графічного інтерфейсу він надає інформацію, яку можна поділити на чотири умовні групи:

- дані про тексти;
- параметри стиснення тексту;
- дані про ключові слова;
- параметри перевірки документів на близькість.

Дані про тексти відповідають текстовій інформації, які користувач надає для обробки.

Параметри стиснення тексту відповідають інформації про ступень стиснення оригінального документу.

Дані про ключові слова включає налаштування підрахунку та виводу найбільш релевантних слів у порядку спадання їх значущості.

Для налаштування виявлення ступеню подібності текстів користувач має ввести параметри перевірки документів на близькість.

З повним переліком реквізитів вхідних даних від користувача можна ознайомитися у таблиці 2.1.

Таблиця 2.1 – Перелік реквізитів первісних даних від користувача

Найменування	Кодове позначення	Тип, розрядність	Кодове позначення документу, який зберігає дані
Файл	InputFile	File	InputFile.jsx

Продовження таблиці 2.1

Найменування	Кодове позначення	Тип, розрядність	Кодове позначення документу, який зберігає дані
Назва тексту	InputFileHandle.name	String	InputFileHandle.jsx
Текст	InputFileHandle.text	String	InputFileHandle.jsx
Розмір стиснення	compCounter	Number, 8 байт	Settings.jsx
Виведення ключових слів	showKeyWords	Boolean, 1 байт	Settings.jsx
Кількість ключових слів	keyWordsCounter	Number, 8 байт	Settings.jsx
Перевірка на основі стиснення	checkingOnComp	Boolean, 1 байт	Settings.jsx
Виведення к-сті слів	showWordsCounter	Boolean, 1 байт	Settings.jsx
Виведення к-сті символів	showSignCounter	Boolean, 1 байт	Settings.jsx

Варто зауважити, що під час завантаження текстового файлу, сервіс додатково витягує з нього інформацію про назву, розмір та зміст документу. Введений користувачем власноруч текст також відразу оброблюється, та подає на вхід інформацію про розмір в байтах.

2.2 Вихідні дані

Вихідними даними системи є звіт, отриманий користувачем по закінченню обробки документів. Даний звіт включає ту чи іншу інформацію в залежності від того, які параметри були обрані під час етапу налаштувань. Передбачена функція завантаження звіту у форматі PDF.

Перелік усіх можливих вихідних сигналів наведена у таблиці 2.2.

Таблиця 2.2 – Перелік вихідних сигналів

Призначення	Найменування	Одиниці виміру	Діапазони зміни	Спосіб представл. інформації
Отримання даних про близькість	Ступень схожості текстів	відсоток	[0; 100]	Графічний інтерфейс користувача. IV етап обробки документів. Перша секція
Отримання даних про час перевірки	Дата перевірки	день	[1; 31]	Графічний інтерфейс користувача. IV етап обробки документів. Друга секція
		місяць	[1; 12]	
		рік	[1; ∞)	
	Час перевірки	година	[0; 23]	
		хвилина	[0; 59]	
		секунда	[0; 59]	
Отримання даних про документи	Назва документу	символ	[1; ∞)	Графічний інтерфейс користувача. IV етап обробки документів. Секція документів

Продовження таблиці 2.2

Призначення	Найменування	Одиниці виміру	Діапазони зміни	Спосіб представл. інформації
Отримання даних про документи	Кількість слів	шт.	[0; ∞)	Графічний інтерфейс користувача. IV етап обробки документів. Секція документів
	Кількість символів	шт.	[0; ∞)	
	Розмір файлу	байт	[0; ∞)	
	Перелік ключових слів	слово	[1; ∞)	
Отримання авторефератів	Стиснений текст документу	речення	[1; ∞)	Графічний інтерфейс користувача. Вкладка перегляду стиснених текстів

2.3 Структура масивів інформації

Після того, як користувач переходить на третій етап обробки завантажених текстів, первісні дані переносяться до оперативної пам’яті обчислювальної машини. Для зручного оперування інформації, що поступила, створюються об’єкти або масиви об’єктів. У даному випадку об’єкти використовуються для зберігання та доступу до колекцій значень і більш складних сутностей.

Отримані від користувача тексти записуються у масив об’єктів, який має структуру, описану у форматі JSON на рисунку 2.1.

```
[
  {
    "name": "Document #1",
    "size": 0,
    "value": "Text in the document #1"
  },
  {
    "name": "Document #2",
    "size": 0,
    "value": "Text in the document #2"
  }
]
```


Рисунок 2.1 – Структура масиву текстів для обробки

Налаштування, обрані користувачем у графічному інтерфейсі, записуються у єдиний об’єкт. Значення ключів об’єкта за замовчуванням вказані на рисунку 2.2. Пояснення до ключів були наведені вище (табл. 2.1).


```
{
  "compCounter": 7,
  "showKeyWords": false,
  "keyWordsCounter": 10,
  "checkingOnComp": false,
  "showWordsCounter": false,
  "showSignCounter": false
}
```


Рисунок 2.2 – Структура об’єкта налаштувань

Під час процесу стиснення створюється складний об’єкт, який складається з масиву параграфів документа. Кожен параграф, у свою чергу, ділиться на речення (рис. 2.3), а речення – на токени (рис. 2.4).

```
{
    "value": "Text of the sentence.",
    "isFirstLast": false,
    "isQuestionExclamation": false,
    "tokens": [],
    "relevance": 0
}
```


Рисунок 2.3 – Структура об’єкта речення


```
{
    "name": "Full-sized word",
    "stemmedName": "Stemmed word",
    "isFirst": false,
    "frequency": 0,
    "inFirstLast": false,
    "inQuestionExclamation": false
}
```


Рисунок 2.4 – Структура об’єкта токена

Таким чином об’єкт речення зберігає інформацію про свій зміст, чи є воно першим чи останнім у параграфі, є окличним чи питальним, про наявні токени у ньому, а також оцінку значущості.

Об’єкт токену має наступні ключі: слово, стемована форма слова, розташування першим у реченні, оцінка значущості, розташування у першому чи останньому, а також питальному чи окличному реченні.

Окрім того, система оперує трьома дата-сетами, які приймають вид ієрархічної структури папок із файлами з розширенням js. Набори даних розташовані у папці data, яка має структуру, що є наведеною у таблиці 2.3.

Інформація у файлах зберігаються у вигляді масивів, що дозволяє швидко їх оброблювати, не витрачаючи додатково час на зчитування.

Набір даних, що зберігає перелік стоп-слів української мови містить 1983 слова.

Таблиця 2.3 – Структура папки data

Найменування дата-сету	Кодове позначення документу
Колекція стоп-слів української мови [8]	stopWords.js
Частоти коренів слів української мови [9]	uaFreqs.js
Колекція абревіатур української мови [10]	abbreviations.js

У дата-сеті uaFreqs знаходяться стемовані версії 98931 словникових форм слів української мови. Кожний запис у масиві – це об’єкт, який має ключі, описані у таблиці 2.4.

Таблиця 2.4 – Перелік ключів об’єкта у дата-сеті uaFreqs

Найменування	Кодове позначення	Тип, розрядність
Корінь слова	stemmed_word	String, по 2 байти на кожний символ
Абсолютна частота слова, для якого обчислюється відносна частота	abs_freq	Number, 8 байт
Відносна частота слова	rel_freq	Number, 8 байт

Відносна частота приймає вигляд цілого числа від 1 до 21, де оцінка 1 – це найбільш поширене слово, а 21 – найменш. Для кожного слова вона розраховується як логарифм від різниці найбільшої абсолютної частоти та абсолютної частоти даного слова за основою 2.

Останній скрипт перетворює найпоширеніші абревіатури української мови до канонічного виду.

Висновок до розділу

У розділі надана інформація щодо форматів первісних даних та вихідних сигналів, які використовуються під час роботи інформаційної системи з визначення близькості документів на основі стиснення текстів.

Наведені описи масивів інформації, які є ключовими під час обробки та бібліотеки, що забезпечують попередню обробку тексту та відіграють важливу роль в аналізі україномовних текстів.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Цільовою аудиторією програмного продукту є учасники навчального процесу будь-якого спрямування, науковці галузі математичної лінгвістики, аналітики, спеціалісти галузей юриспруденції, журналістики, видавничої справи, лінгвістики, літератури тощо. Дані сфери пов’язані з великим обсягом інформації, яка потребує опрацювання та порівняння, тому вебзастосунок дозволить автоматизувати деякі монотонні процеси даної роботи та зекономити час.

Таким чином, програмна розробка забезпечує визначення схожості між заданими користувачем текстами українською мовою, порівнюючи лише найбільш важливі фрагменти заданого обсягу. Після виконання обчислень користувачу сервісу надається детальний звіт.

Окрім цього, для науковців, що проводять наукові дослідження, забезпечується можливість змінювати параметри вхідних даних для дослідження поведінки алгоритму.

Варто зазначити, що для визначення схожості двох документів на основі їх анотацій, необхідно мати широку лексичну базу для обробки наявних у зазначених текстах токенів.

Виходячи з усього вище сказаного, можна сформулювати та навести головну задачу даної інформаційної системи як задачу визначення близькості текстів на основі їх стиснення.

3.2 Математична постановка задачі

Нехай існують тексти $\boldsymbol{A_1}$ та $\boldsymbol{B_1}$, які необхідно порівняти між собою на близькість, попередньо проводячи над ними операцію стиснення. Знайти оцінку їх подібності із мінімальними витратами часу.

Дано:

- $\boldsymbol{A_1}$ – перший текст для порівняння;
- $\boldsymbol{B_1}$ – другий текст для порівняння;
- $\boldsymbol{Z}$ – словник стоп-слів української мови;
- $\boldsymbol{c}$ – параметр стиснення текстів.

Змінні:

- $\boldsymbol{A_2}$ – автореферат тексту A_1;
- $\boldsymbol{B_2}$ – автореферат тексту B_1;
- $\boldsymbol{t}$ – час обробки тексту;
- $\boldsymbol{p}$ – множина параграфів тексту;
- $\boldsymbol{k}$ – множина речень тексту;
- $\boldsymbol{w}$ – кількість унікальних токенів тексту;
- $\boldsymbol{s}$ – оцінка подібності текстів A_1 та B_1;
- $\boldsymbol{f}$ – оцінка значущості токена;
- $\boldsymbol{r}$ – оцінка значущості речення;
- $\boldsymbol{n}$ – кількість біграм, знайдених у тексті.

Цільова функція (3.1):

$$Z = t_a + t_b \rightarrow min \tag{3.1}$$

Таким чином цільова функція представляє собою мінімізацію сумарного часу обробки документів A_1 та B_1.

3.3 Обґрунтування методу розв’язання

Для вирішення поставленої задачі були обрані наступні методи стиснення та виявлення схожості текстів:

- класичний алгоритм шинглів;
- алгоритм Дайса;

– загальний алгоритм стиснення тексту.

Дані методи беруть за основу напрацювання галузі лінгвістики для виділення лінгвістичних одиниць, точність виявлення яких впливатиме на ефективність обчислень. Результатом роботи алгоритму стиснення в узагальненому вигляді має бути автореферат заданої довжини. Результатом роботи алгоритмів визначення близькості текстів є відсоток співпадіння.

Використання класичного алгоритму шинглів обумовлене тим, що даний метод є відносно простим та ефективним. У проведеному дослідженні [11] було виявлено, що він показує гарні результати у відношенні економії витрат часу, легкості застосування та ефективності визначення схожих за структурою фрагментів тексту.

Описаний метод для порівняння текстової інформації, який оснований на працях Поля Жаккара [12], написаних більше століття тому, є достатньо популярним на сьогоднішній день. Проте, не дивлячись на свою популярність, ефективність даного методу зменшується при порівнянні вхідних текстів, які є невеликими за обсягом.

Наразі існує велика кількість методів, які є більш ефективними для порівняння відносно малих текстів. Однак, для коректності отриманих результатів має бути застосований метод, який за принципом буде близьким до обчислення індексу Жаккара у класичному методі шинглів, а також даватиме оптимальний результат при порівнянні відносно невеликих фрагментів текстової інформації.

Для порівняння невеликих текстів найбільш доцільним є використання методу, що базується на порівнянні біграмів (алгоритм Дайса).

Для стиснення текстів, або, іншими словами, автоматичного реферування, був обраний метод, що заснований на напрацюваннях Гарольда Едмунсона [13]. Обрання даного методу пов’язане з відсутністю об’ємної лексичної бази, наявність якої є обов’язковою для ефективного використання

інших загальновідомих алгоритмів стиснення текстів. На жаль, даних напрацювань саме для української мови не було знайдено.

3.4 Опис методів розв'язання

Методи визначення близькості текстів

Ступень подібності двох документів між собою, використовуючи коефіцієнт Жаккара, обчислюється як відношення величин перетину та об'єднання їх елементів (3.2):

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.2}$$

де $|A|$ – розмір масиву A, елементами яких можуть бути слова, речення;

$|B|$ – розмір масиву B, елементами яких можуть бути слова, речення.

Перед обробкою приводимо слова до канонічного вигляду, прибираючи спеціальні символи, розділові знаки, зайві пробіли, а також стоп-слова. Після цього проводимо процес токенізації, або розбиття тексту на лексичні одиниці. В отриманих токенах відкидаємо частини слова, які не є їх коренем. Після цього текст розбивається на шингли. Хешуємо шингли. Контрольні суми шинглів тексту A порівнюємо із контрольними сумами тексту B.

Для порівняння невеликих текстів було обрано другий метод, що базується на порівнянні біграмів (3.3) та визначається як:

$$s = \frac{2n_t}{n_x + n_y} \tag{3.3}$$

де n_t – к-сть біграм, знайдених в обох текстах;

n_x – к-сть біграм, знайдених у тексті x;

n_y – к-сть біграм, знайдених у тексті y.

Метод автоматичного реферування

Стиснення текстів відбувається з допомогою оцінки значущості кожного речення.

Спочатку розбиваємо текст на параграфи, речення та токени. Викидаються токени, які відповідають стоп-словам. Виконуємо операцію стемінгу слів. Для нього характерне використання морфемного розбору слова. Позначимо значущість слова як оцінку показника TF-IDF (3.4):

$$TFIDF = \frac{n_i}{\sum_k n_k} \cdot \log \frac{|D|}{|d_i \supset t_i|} \tag{3.4}$$

де n_i – число входжень слова у досліджуваному тексті;

$\sum_k n_k$ – загальна кількість слів у тексті;

$|D|$ – кількість документів колекції;

$|d_i \supset t_i|$ – кількість документів, в яких зустрічається слово t_i.

Формуємо «мішок» унікальних токенів тексту. Обчислюємо додатковий коефіцієнт, якщо слово знаходиться у першому чи останньому реченні параграфу (3.5):

$$f_{new} = \frac{n_k}{n_m} \cdot f \tag{3.5}$$

де n_k – загальна кількість речень у тексті;

n_m – загальна кількість перших та останніх речень параграфу;

f – обчислена значущість слова.

Обчислюємо коефіцієнт за розташування токену у питальному чи окличному реченнях (3.6):

$$f_{new} = \frac{n_k}{n_q} \cdot f \tag{3.6}$$

де n_k – загальна кількість речень у тексті;

n_q – загальна кількість питальних та окличних речень;

f – обчислена значущість слова.

Обчислюємо значущість слова, якщо воно є власним ім'ям (3.7):

$$f_{new} = \log 15 \cdot f \tag{3.7}$$

де f – обчислена значущість слова.

Після отримання кінцевого значення значущості слова після додавання усіх додаткових коефіцієнтів, розраховуємо значущість речень як суму оцінок важливості усіх токенів у ньому (3.8):

$$r = \sum_{i=0}^{n} f_i \tag{3.8}$$

де n – загальна кількість усіх токенів у реченні;

f_i – значущість слова *i*.

В результаті роботи алгоритму отримуємо реферат заданої в умові задачі довжини, в якому речення є впорядкованими за їх розташуванням в оригінальному тексті.

Висновок до розділу

В даному розділі сформульовані змістовна та математична постановки задачі.

Проведено огляд доступних методів для розв’язання поставленої задачі та обґрунтовано їх вибір: алгоритму шинглів, алгоритму Дайса та загального алгоритму автоматичного реферування.

Наведено детальний опис алгоритмів обчислення значущості слів з урахуванням типу речення, його розташуванням у тексті, типу токену з наданням відповідних формул.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для розробки програмного забезпечення було обрано декларативну, ефективну та гнучку JavaScript бібліотеку React, яка є одним із найбільш популярних засобів на сьогоднішній день для створення користувацьких інтерфейсів [14].

Дана бібліотека була створена компанією Facebook у 2013 році для вирішення проблем, пов’язаних із масштабуванням вебсайтів, які були керованими даними. Через застосування новітніх підходів тоді дану бібліотеку сприйняли з певною долею скептицизму [15]. А вже сьогодні спеціалісти дискутують на тему, чи можна вважати React повноцінним фреймворком, як, наприклад, Angular, чи варто залишити йому статус невеликої бібліотеки з просунутим функціоналом.

Мінімалістичний та зрозумілий синтаксис робить React швидким та легким в опануванні, та дозволяє створювати вебсайти з блискавичною швидкістю.

Головною перевагою React є використання концепції віртуальної об’єктної моделі документу HTML [14]. Якщо під час розробки необхідно модифікувати елемент вебсторінки, то усі зміни спочатку вносяться у віртуальний DOM, а бібліотека знаходить найшвидший спосіб, що є достатнім для оновлення реального DOM до нового стану.

Окрім того, бібліотека використовує файли формату JSX, створеному на перетині синтаксису мови програмування JavaScript та розмітки XML. Така структура дозволяє розробнику зосередити увагу на створенні компонентів, які можуть бути використані повторно у різних програмах.

У зв’язці з React, як правило, використовують бібліотеку Redux для управління станом застосунку. Вона базується на використанні особливих

концепцій, які спрощують роботу із технологією. Офіційне визначення Redux [16] звучить як передбачуваний контейнер станів для JavaScript застосунків.

Для створення програмного забезпечення було включено метамову під назвою SCSS, яка є розширенням CSS [17]. Технологія не надає нових можливостей до каскадної таблиці стилів, проте значно підвищую швидкість написання коду за рахунок покращень синтаксичних конструкцій.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Даний програмний продукт представляється у вигляді вебзастосування для визначення схожості документів на основі стиснення їх змісту.

Мінімальні вимоги до системи є наступними:

- операційна система Windows 7, або OS X El Capitan (10.11), або Ubuntu 14.04;
- об’єм оперативної пам’яті 4 GB;
- графічний пристрій DirectX 9 з драйвером WDDM 1.0;
- розширення екрану SVGA 800×600;
- програмна платформа Node.js;
- підключення до мережі Інтернет;
- встановлений браузер.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма класів

Розроблена система не використовує структуру класів. Замість цього застосовуються React компоненти та JavaScript об’єкти, тому побудова діаграми класів є неможливою.

4.3.2 Діаграма послідовності

У графічному матерілі представлена схема структурна послідовності для відображення взаємодії між компонентами вебзастосунку для процесу обробки текстів. Діаграма складається із шести компонентів: «:Користувач», «:Main», «:Settings», «:Treatment», «:Results» та «:DocumentResult».

Об’єкт «:Користувач» завантажує документ до об’єкта «:Main» та проводить налаштування, записуючи їх у об'єкт «:Settings». Після цього ці об’єкти передають дані, отримані на вхід, до об’єкта «:Treatment». Проходить обробка даних. Отримані результати передаються до об’єкта «:Results», а після надаються об’єкта «:Користувач».

Представлена можливість надання запитів на перегляд згенерованих авторефератів та завантаження звіту об’єктом «:Користувач». В першому випадку для надання інформації об’єкт «:Results» додатково передає дані до об’єкта «:DocumentResult», який опрацьовує запит та повертає відповідь.

4.3.3 Діаграма компонентів

Розроблена система має Flux архітектуру. У цій архітектурі події оброблюються по черзі за кругообігом з кількома дійовими особами: диспетчер (або редуктор), сховище та дії [15].

Діаграма компонентів системи, що є наведеною у графічному матерілі, схематично описує взаємодію між компонентами.

Серверна частина складається з файлового та локального сховищ даних. Локальне сховище необхідне для збереження історії запитів, а файлове – для коректної обробки україномовних текстів.

Користувач напряму взаємодіє із графічним інтерфейсом програми, який складається з React-компонентів.

Графічний інтерфейс та серверна частина інформаційної системи обмінюються даними один з одним за допомогою протоколу HTTP.

Важливою є робота Redux модулю. Представлення застосовує метод dispatch() для з'єднання з редуктором. Варто зазначити, що кожна виконувана дія, що надсилається до сховища використовуючи зворотній зв'язок, реєструється в Dispatcher. Після того, як сховище оновлює дані у відповідь на викликану дію, воно за допомогою методу subscribe() публікує зміну події.

Контролери, які прослуховують зміни подій, отримують новий стан даних зі сховища та постачають нову інформацію до відповідних представлень.

4.3.4 Специфікація функцій

Наведемо у таблиці 4.1 специфікацію основних функцій програмного продукту.

Таблиця 4.1 – Специфікація функцій програми

Назва файлу	Назва функції	Призначення функції
analyzeTexts.js	analyzeTexts (settings, p, ut)	Створення анотації оригінального тексту, виявлення ключових слів
base.js	getUniqueTokens (paragraphs)	Отримання унікальних токенів із заданої вибірки
	concatTokens (paragraphs)	Об'єднання наявних у тексті токенів
	concatSentences (paragraphs)	Об'єднання наявних у тексті речень

Продовження таблиці 4.1

Назва файлу	Назва функції	Призначення функції
base.js	deleteDuplicates (tokens)	Видалення дублікатів стемованих версій слів
detectSimilarity.js	detectSimilarity(files, summary_1, summary_2, settings)	Обрання методу перевірки на близькість в залежності від налаштувань
	compareByShingle(text_1, text_2, shingleLength)	Порівняння текстів алгоритмом шинглів
	normalizeText (text)	Приведення тексту до нормалізованого вигляду
	createShingles (text, shingleLength)	Створення шинглів заданої довжини
	hashShingles (shingles)	Хешування заданих шинглів
	compareShingles (hashes_1, hashes_2)	Порівняння шинглів

Продовження таблиці 4.1

Назва файлу	Назва функції	Призначення функції
prepareFiles.js	prepareFiles (data)	Підготовка файлів до подальшої роботи: проведення процесу токенізації, встановлення границь для виведення ключових слів та проведення стиснення
	processFile (data)	Обробка файлу: проведення процесу токенізації, визначення унікальних токенів
stem.js	stem(word)	Проведення операції стемінгу
summarize.js	summarize (settings, paragraphs, uniqueTokens)	Опис алгоритму визначення стисненої версії оригінального тексту та пошуку ключових слів
	getKeywordValues (keywords)	Отримання масиву ключових слів
	getSummary (sentences, compSize)	Отримання анотації оригінального тексту

Продовження таблиці 4.1

Назва файлу	Назва функції	Призначення функції
summarize.js	getRelevantSentences (sentences, compSize)	Отримання релевантних речень для стиснення
	sortRelevantSentences (sentences)	Сортування списку релевантних речень для стиснення
	sortKeyWords (uniqueTokens)	Сортування списку унікальних токенів для формування списку ключових слів
	evaluateSentences (paragraphs)	Розрахунок значущості речень
tf_idf.js	calcTF_IDF (paragraphs, tokens)	Опис алгоритму розрахунку оцінки показника TF-IDF
	calcQuantitySentences (paragraphs)	Обчислення кількості питальних, окличних, перших та останніх речень параграфів
	calcBySpecialSentences (token, sentenceQuantity, questExclQuantity, firstLastQuantity)	Присвоєння додаткових коефіцієнтів токенам на основі їх розташування у тексті

Продовження таблиці 4.1

Назва файлу	Назва функції	Призначення функції
tf_idf.js	calcByTFIDF (token, quantity, tokenLength)	Обчислення показника TF-IDF
	getTokenFrequency (tokenStemmedName)	Отримання табличного значення відносної частоти стемованого токена
	calcByProperName (token)	Присвоєння додаткового коефіцієнта власним назвам
	findProperNames (tokens)	Пошук токенів, які є власними назвами
tokenize.js	findParagraphs (data)	Визначення складного об’єкта, який складається з масиву параграфів документа
	findSentences (data)	Виявлення речень у тексті
	makeTokens (paragraphs)	Обробка токенів тексту: проведення алгоритму токенізації, відкидання стоп-слів, додавання інформації про токен

Продовження таблиці 4.1

Назва файлу	Назва функції	Призначення функції
tokenize.js	makeToken (name, stemmedName, isFirst)	Створення об'єкта токена
	makeSentence (value)	Створення об’єкта речення
	extractTokens (tokens, sentence)	Пошук токенів у тексті
	addTokenInfoAboutSentences (paragraphs)	Оновлення значень ключів токена на основі їх розташування у тексті
	tokenize (data)	Проведення процесу токенізації
	findSpecialSentences (paragraphs)	Виявлення речень, які мають особливе розташування у тексті
	removeStopWords (dataArray, stopWords)	Видалення стоп-слів

4.4 Опис звітів

В результаті роботи програмного продукту на виході генерується детальний звіт роботи алгоритму. Перелік можливої інформації звіту в залежності від обраних налаштувань може містити: відсоток схожості документів, дата перевірки, назви документів, кількість слів та символів, розміри завантажених файлів, список ключових слів.

На рисунку 4.3 представлений результат обробки двох документів.

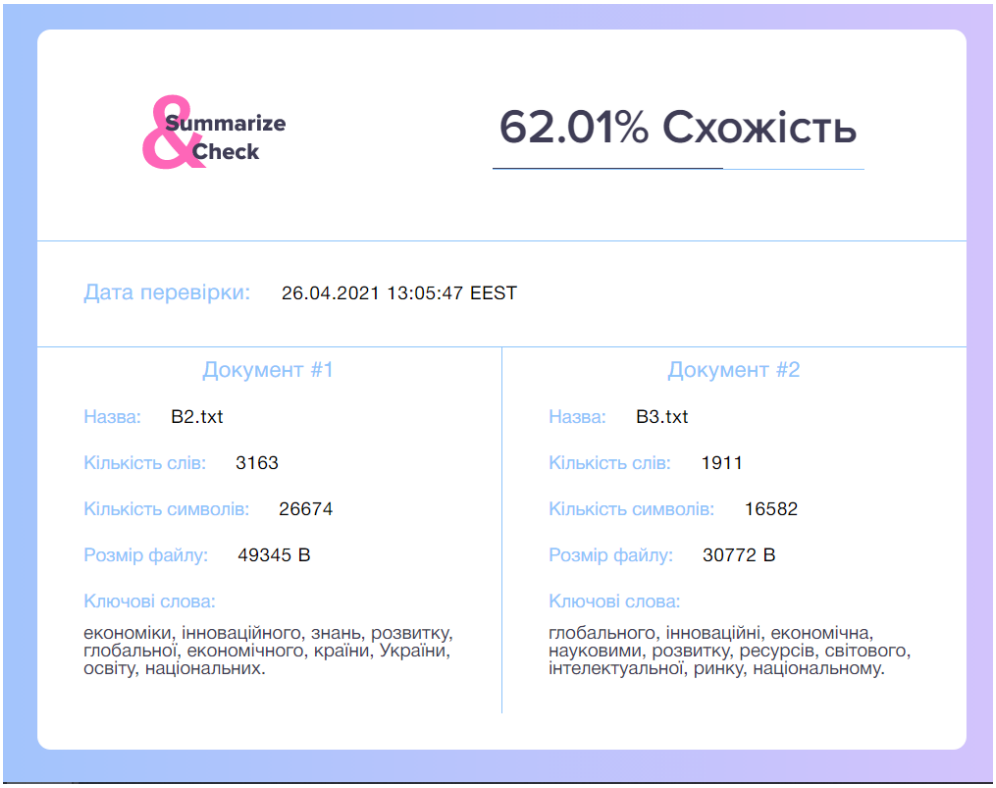


Рисунок 4.3 – Результат обробки документів

Користувач має змогу завантажити звіт у форматі PDF, який буде точною копією результату, представленого на рисунку 4.3.

Окрім цього, представлена можливість перегляду згенерованих автоматичних рефератів завантажених користувачем документів (рис. 4.4).

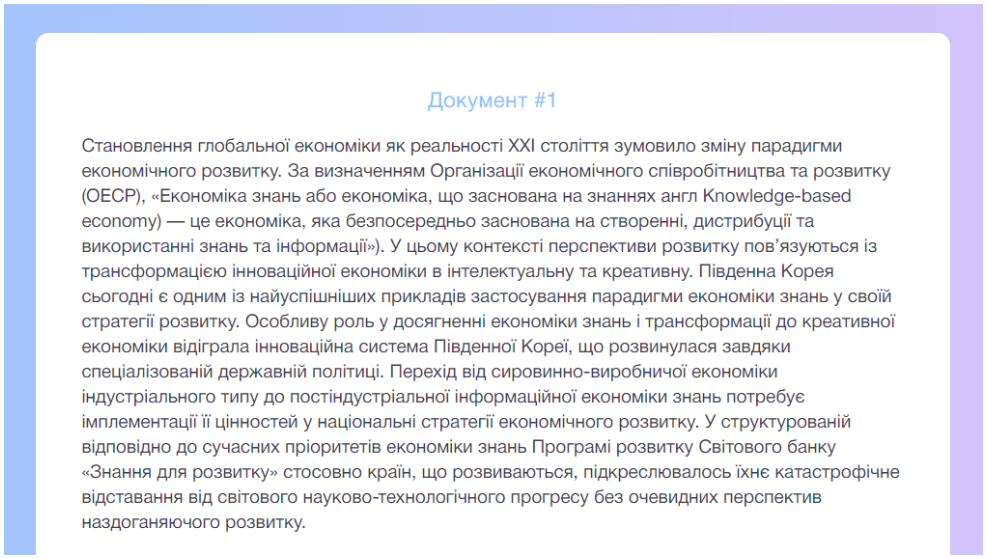


Рисунок 4.4 – Автоматичний реферат документа

Автореферат містить ті речення, які визначені як найбільш значущі згідно алгоритмів, наведених у розділі 3.

Висновок до розділу

Розглянуто засоби розробки програмного забезпечення, обґрунтовано обрання бібліотеки React у зв'язці з Redux, а також метамови SCSS як альтернативи каскадним таблицям стилів.

Наведено загальні вимоги до програмного та технічного забезпечення для коректної роботи програмного продукту.

Описано архітектуру вебзастосунку. Представлено схему структурну послідовності, що відображає взаємодію між компонентами системи для процесу обробки текстів. Спроектовано діаграму компонентів для опису взаємодій між компонентами. Наведено специфікацію функцій створеної інформаційної системи.

Окрім цього, описано звіти, які генеруються в результаті обробки завантажених документів, та наведено приклади.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Для забезпечення функціональності, яка відповідатиме описаній у розділі 1.1.2, було розроблено вебінтерфейс інформаційної системи, який складається із трьох розділів: «Головна», «Історія запитів» та «Про сервіс». Розділ «Головна», в свою чергу, ділиться на чотири етапи обробки текстів.

Розроблене вебзастосування має бути розгорнуто на локальному сервері та відкрито за допомогою веббраузера за адресою http://localhost:3000/.

Головною сторінкою інтерфейсу є розділ «Про сервіс». Він надає можливість ознайомитися з описом задачі сервісу та інструкцією. Окрім цього, користувач спроможний швидко перейти до функціональної частини програмного продукту, натиснувши відповідну кнопку.

Зовнішній вигляд вебсторінки «Про сервіс» представлений на рисунках 5.1-5.2.

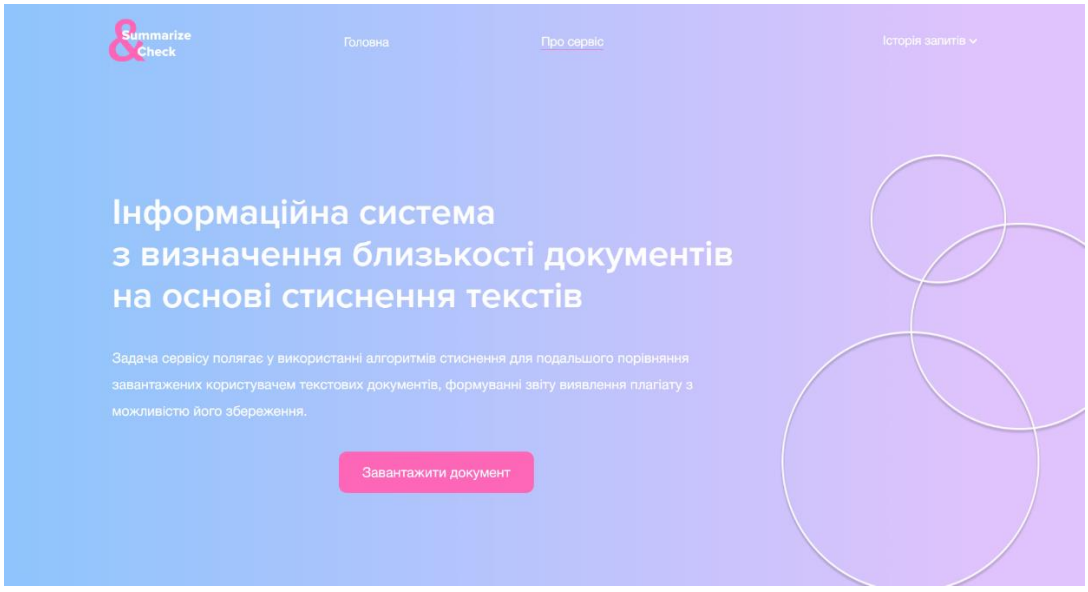


Рисунок 5.1 – Розділ «Про сервіс». Опис задачі сервісу

Рисунок 5.2 – Розділ «Про сервіс». Інструкція користувача

Розділ «Історія запитів» надає доступ до останніх запитів, зробленим користувачем у системі. За відсутності будь-яких запитів відображатиметься відповідне повідомлення (рис. 5.3).

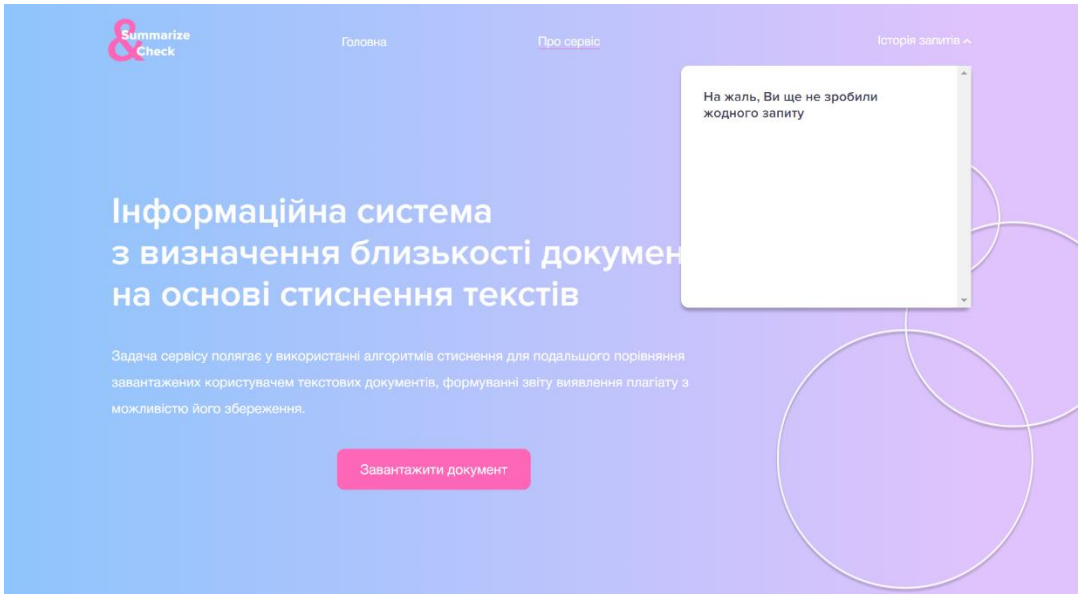


Рисунок 5.3 – Розділ «Історія запитів». Запити відсутні

На рисунку 5.4 представлений зовнішній вигляд розділу, якщо користувач системи вже надсилав документи для обробки та отримував звіти.

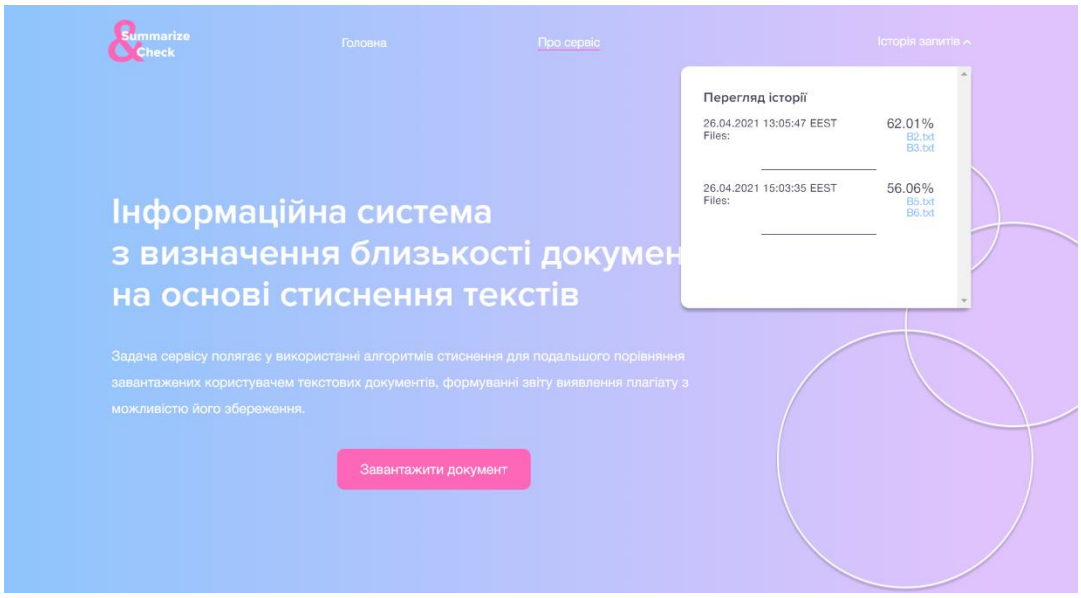


Рисунок 5.4 – Розділ «Історія запитів». Запити наявні

У розділі «Головна» розміщений основний функціонал інформаційної системи. Даний розділ містить наступні етапи обробки текстів:

– завантаження файлів;
– налаштування;

– обробка;

– отримання результатів.

На першому етапі обробки текстів користувач має змогу виконувати наступні функції:

– вибір способу надання текстів на обробку;

– завантаження текстових документів перетягуванням їх у визначену область;

– завантаження текстових документів за допомогою програми «Провідник»;

– заповнення текстових полів назви тексту та його змісту;

– видалення завантажених текстів.

На рисунку 5.5 зображений зовнішній вигляд вебсторінки, на якій користувач вводить тексти власноруч.

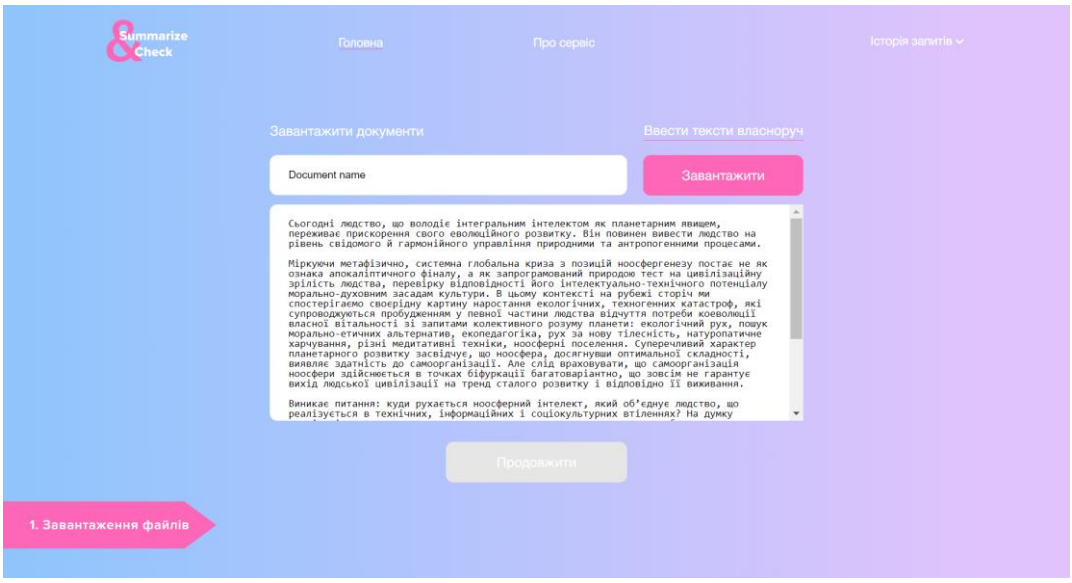


Рисунок 5.5 – Розділ «Головна». Введення текстів власноруч

На рисунку 5.6 представлена вебсторінка, на якій користувач завантажує текстові документи. Так як було додано два тексти для подальшої обробки, область завантаження нових файлів не є активною. Окрім цього, після завантаження хоча б одного тексту, з'являється панель «Завантажені тексти».

Справа від кожного з них наявна кнопка, яка дозволяє видалити небажаний документ.

Рисунок 5.6 – Розділ «Головна». Завантаження документів

Другий етап обробки текстів дозволяє змінювати параметри вхідних даних для дослідження поведінки алгоритму. Для зручності налаштування розбиті на чотири секції за тематикою.

Якщо обрано перевірку документів на схожість на основі стиснення, то користувач має обрати розмір очікуваного стиснення.

Незалежно від принципу порівняння текстів на близькість, активними є наступні налаштування:

– виведення ключових слів;
– виведення загальної кількості слів;
– виведення загальної кількості символів;
– обрання алгоритму (на основі обчислення коефіцієнту Дайса чи коефіцієнту Жаккара).

Якщо користувач бажає отримати ключові слова кожного документа, він має додатково обрати їх кількість.

Якщо обрано класичний алгоритм шинглів, додатково має обиратися довжина шинглу. За замовчуванням це значення дорівнює п’яти.

Користувач може повернутися на попередній етап, натиснувши стрілку у лівій частині екрану або на стрічку, розташованій унизу вебсторінки.

Інтерфейс даного етапу представлений на рисунку 5.7.

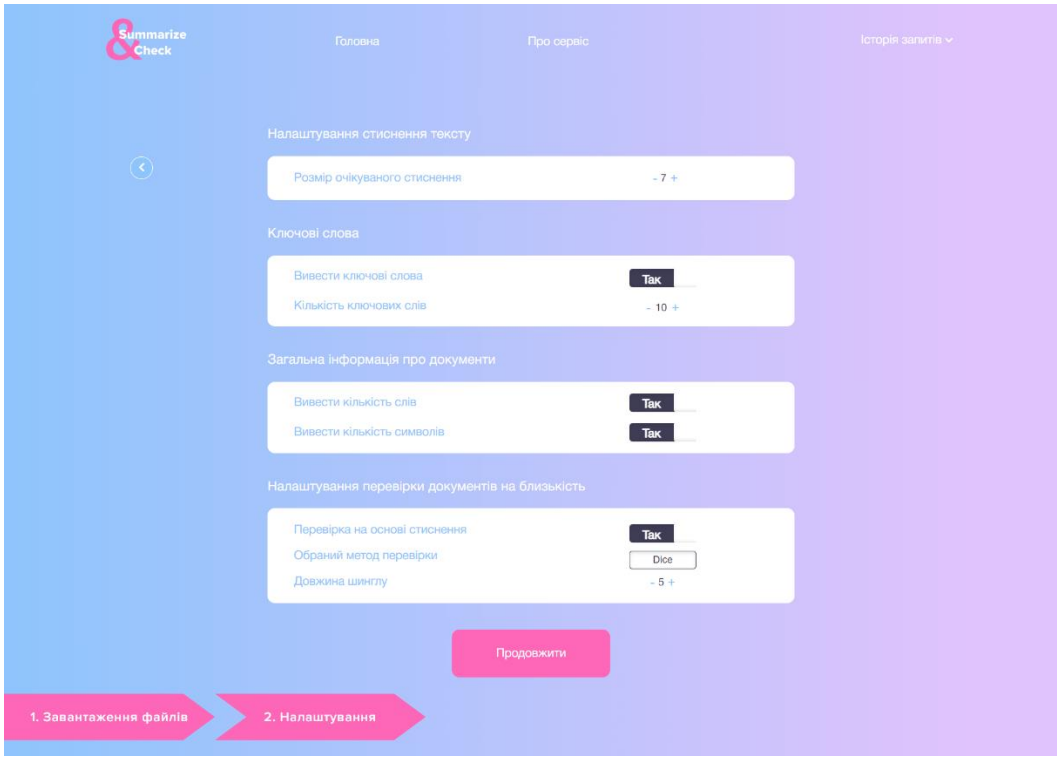


Рисунок 5.7 – Розділ «Головна». Налаштування

Під час очікування результатів виводиться відповідне повідомлення (рис. 5.8).

На четвертому етапі є доступними наступні функції:

– перегляд результатів обробки текстів;

– перегляд стиснених версій оригінальних текстів;

– завантаження звіту у форматі PDF.

Ознайомлення з автоматичними рефератами можливе тільки тоді, якщо під час налаштувань було обрано перевірку документів на схожість на основі стиснення.

Рисунок 5.8 – Розділ «Головна». Очікування завершення обробки

Опис згенерованого звіту наведено у розділі 4.4 пояснювальної записки дипломного проєкту.

На рисунку 5.9 зображена вебсторінка останнього етапу обробки текстів.

Рисунок 5.9 – Розділ «Головна». Згенерований звіт

При натисканні на кнопку перегляду стиснених текстів, відкривається вебсторінка з відповідними авторефератами, вигляд якої зображено на рисунку 5.10.

Рисунок 5.10 – Розділ Головна. Перегляд авторефератів

При завантаженні звіту програма робить знімок вебсторінки та автоматично зберігає на комп'ютері файл з назвою summarize_check.pdf. Вигляд отриманого документу представлено на рисунку 5.11.

Рисунок 5.11 – Завантажений звіт у форматі PDF

Звіт містить інформацію, отриману під час обробки завантажених документів.

5.2 Випробування програмного продукту

В цьому підрозділі наведено опис проведених тестів і порядок їх виконання для перевірки відповідності програмного забезпечення функціональним вимогам, представленим у технічному завданні на створення інформаційної системи з визначення близькості документів на основі стиснення текстів.

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій інформаційної системи з визначення близькості документів на основі стиснення текстів вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Для тестування програмного продукту було узято набір статей українських науковців з сайту бібліотеки Вернадського [18], що були написані за однією тематикою. Приблизний розмір кожного з текстів складає 8-10 сторінок у форматі видання.

У таблицях 5.1-5.3 наведено кілька випробувань основних функціональних можливостей інформаційної системи (ІС) та їх результати.

Таблиця 5.1 – Випробування 1

Назва тесту	Обробка текстів, завантажених різними методами
Початковий стан ІС	Локальний сервер запущений. Користувач знаходиться за посиланням http://localhost:3000/#/s-check/main.
Вхідні дані	Текст для введення власноруч, назва тексту для введення власноруч, текстовий документ.
Схема проведення тесту	Завантажити текстовий документ або за допомогою програми «Провідник», або перенесенням у відведену область. Перейти на сторінку введення текстів власноруч. Ввести назву тексту та його зміст. Натиснути кнопку «Завантажити». Натиснути кнопку «Продовжити».
Очікуваний результат	Завантаження інших документів не є можливим. Тексти завантажені у систему для подальшої обробки. Відкривається етап проведення налаштувань.
Стан ІС після проведення випробувань	Відкрита сторінка з налаштуваннями. Спосіб завантаження файлів не впливає на проходження наступних етапів обробки текстів.

Таблиця 5.2 – Випробування 2

Назва тесту	Завантаження звіту
Початковий стан ІС	Локальний сервер запущений. Користувач знаходиться за посиланням http://localhost:3000/#/s-check/results.
Вхідні дані	Результати обробки завантажених текстів.
Схема проведення тесту	Попередньо пройти усі етапи обробки документів. Натиснути кнопку «Завантажити звіт у PDF».
Очікуваний результат	На комп’ютер завантажено звіт у форматі PDF. Отриманий документ містить усю інформацію, представлену на сайті.
Стан ІС після проведення випробувань	Відкрита сторінка з результатами обробки текстів. Навігація по сайту працює без помилок.

Таблиця 5.3 – Випробування 3

Назва тесту	Видалення завантажених текстів
Початковий стан ІС	Локальний сервер запущений. Користувач знаходиться за посиланням http://localhost:3000/#/s-check/main.
Вхідні дані	Текст для введення власноруч, назва тексту для введення власноруч, текстовий документ.

Продовження таблиці 5.3

Схема проведення тесту	Завантажити текстовий документ або за допомогою програми «Провідник», або перенесенням у відведену область. Перейти на сторінку введення текстів власноруч. Ввести назву тексту та його зміст. Натиснути кнопку «Завантажити». Видалити завантажені тексти. Провести операцію повторно.
Очікуваний результат	Документи видаляються без помилок. Після операції видалення завантаження нових документів працює справно.
Стан ІС після проведення випробувань	Інформаційна система перебуває у своєму початковому стані. Видалення завантажених текстів не впливає на проходження наступних етапів обробки текстів.

Загалом були реалізовані наступні тестові сценарії:

- завантаження текстів методом перетягування документів в означену область;
- завантаження текстів методом відкриття програми «Провідник»;
- завантаження текстів методом ручного введення;
- обробка текстів, завантажених різними методами;
- видалення завантажених текстів;
- обрання алгоритмів під час проведення налаштувань;
- обрання критично малого та великого очікуваного розміру стиснення текстів під час проведення налаштувань;
- обрання максимальної кількості ключових слів під час проведення налаштувань;

– завантаження звіту;
– перегляд історії (запитів немає);
– перегляд історії (запити наявні);
– перегляд стиснутих текстів.

Тести були проведені методом ручного тестування.

В процесі проведення етапу випробувань було виявлено, що всі компоненти інформаційної системи працюють відповідно до функціональних вимог, наведених у технічному завданні.

Висновок до розділу

У розділі представлено керівництво користувача та наведено екранні форми інтерфейсу розробленої інформаційної системи. Описано дії, які можуть бути виконані користувачем під час взаємодії з вебзастосунком.

Також у розділі сформульована мета випробувань та визначені документи, які є фундаментом для формування випробувань програмного продукту.

Вказано перелік тестових сценаріїв для перевірки відповідності програмного забезпечення функціональним вимогам, представленим у технічному завданні та результати випробувань.

ЗАГАЛЬНІ ВИСНОВКИ

В результаті виконання дипломного проєкту було розроблено програмний продукт для проведення експериментів та загального користування у вигляді вебзастосунку. Дана розробка дозволяє виконати операції стиснення текстів і визначення близькості наданих документів. Програмний продукт створений з акцентуванням на здатності обробляти тексти українською мовою.

Під час аналізу предметної області було досліджено програми – аналоги з функціоналом визначення близькості текстів та механізмами стиснення текстів. Визначено, що існують істотні проблеми з аналізом україномовних текстів, відсутністю у єдиному процесі визначення подібності документів за їх скороченими аналогами.

Було розглянуто алгоритми стиснення даних, формування стиснутої форми (автореферату) із заданими параметрами, алгоритми визначення близькості текстів для текстів невеликого обсягу.

Сформульовано змістовна та математична постановки задачі, зроблено обґрунтований вибір методів для їх розв’язання. Наведено опис використаних алгоритмів для формування стиснутого варіанту тексту (автореферату), що враховують оцінку важливості елементів тексту на основі типу речення, розташуванням та параметрів токену.

Визначено вхідні та вихідні дані, порядок обробки текстової інформації із застосуванням словників стоп-слів, частот коренів слів та абревіатур.

Описано засоби розробки, архітектуру та функціональні можливості програмного продукту. Проведено тестування щодо відповідності програмного продукту функціональним вимогам. Визначено його цільову аудиторію. Наведені вимоги до технічного забезпечення.

Розроблена інформаційна система успішно пройшла всі випробування та готова до впровадження в експлуатацію.

Проведено експериментальне дослідження визначення відсотку близькості тексту щодо його стиснутого варіанту та поведінку коефіцієнту схожості текстів між собою та їх стиснутих аналогів. Результати дослідження були оприлюднені у науково-технічній конференції [3].

Готовий програмний продукт робить можливим проведення досліджень, які полягають у визначенні близькості документів на основі стиснення текстів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Computational Linguistics. *Stanford Encyclopedia of Pholosophy*. 2014. URL: https://plato.stanford.edu/entries/computational-linguistics/. (дата звернення: 15.05.2021).
2. Nature language processing. *Oxford Learner's Dictionaries.* URL: https://www.oxfordlearnersdictionaries.com/definition/english/natural-language-processing. (дата звернення: 15.05.2021).
3. Шеляхіна В.В. Технологія визначення близькості документів на основі стиснення текстів. *VI всеукраїнська науково-практична конференція молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2021)*: матеріали VI всеукр. наук.-практ. конф., м. Київ, 22-23 квіт. 2021 р. Київ, 2021.
4. Hidayat E. Y., Firdausillah F., Hastuti K., Dewi I. N., Azhari A. Automatic Text Summarization Using Latent Drichlet Allocation (LDA) for Document Clustering. *International Journal of Advances in Inteligent Informatics*. 2015. Vol. 1, No 3. P. 132. doi: https://doi.org/10.26555/ijain.v1i3.43.
5. Prasetya D.D., Wibawa A.P., Hirashima T. The performance of text similarity algorithms. *International Journal of Advances in Inteligent Informatics*. 2018. Vol. 4, No 1. P. 63-69. doi: https://doi.org/10.26555/ijain.v4i1.152.
6. Yunianta A., Barukab O. M., Yusof N., Dengen N., Haviluddin H., Othman M. S. Semantic data mapping technology to solve semantic data problem on heterogeneity aspect. *International Journal of Advances in Inteligent Informatics*. 2017. Vol. 3, No 3. P. 161-172. doi: https://doi.org/10.26555/ijain.v3i3.131.
7. Gomaa W. H., Fahmy A. A. A survey of text similarity approaches. *International Journal of Computer Applications.* 2013. Vol. 68, No 13. P. 13-18. doi: https://doi.org/10.5120/11638-7118.

8. Ukrainian Stopwords. URL: https://kuprienko.info/ukrainian-stopwords/. (дата звернення: 15.05.2021).
9. Частотний словник української мови. URL: http://u-mova.blogspot.com/2013/09/blog-post.html. (дата звернення: 15.05.2021).
10. ДСТУ 3582-2013. Видання. Бібліографічний опис скорочення слів і словосполучень українською мовою. [Чинний від 2014-01-01]. Вид. офіц. Київ, 2013. 15 с. (Інформація та документація).
11. Nishimura M. The Best Document Similarity Algorithm. *Towards Data Science.* 2020. URL: https://towardsdatascience.com/the-best-document-similarity-algorithm-in-2020-a-beginners-guide-a01b9ef8cf05. (дата звернення: 15.05.2021)
12. Jaccard P. Distribution de la flore alpine dans le Bassin des Dranses et dans quelques regions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*. 1901. Vol. 37, No 140. P. 241—272.
13. Edmundson H. P. New Methods in Automatic Extracting. *Journal of the ACM.* 1969. Vol. 16, No 2. P. 264-285. doi: https://doi.org/10.1145/321510.321519.
14. React Documentation. URL: https://ru.reactjs.org/tutorial/tutorial.html. (дата звернення: 15.05.2021).
15. Banks A., Porcello E. Learning React. Functional Web Development with React and Flux. 2017. United States of America, 2017. P. 1-437.
16. Dinkevich B., Gelman I. The complete Redux Book. 2017. P. 1-169.
17. SASS Documentation. URL: https://sass-scss.ru/. (дата звернення: 15.05.2021).
18. Національна бібліотека України імені В.І. Вернадського. URL: http://www.nbuv.gov.ua/.

Додаток А

Тексти програмного коду

Інформаційна система з визначення близькості документів на основі стиснення текстів

(Найменування програми (документа))

DVD-R

(Вид носія даних)

19 арк, 74 Кб

(Обсяг програми (документа) , арк., Кб)

Київ – 2021 року

analyzeTexts.js

```
import summarize from "./summarize";

export default function analyzeTexts(settings, p, ut) {
    let summary_1 = "",
        keywords_1 = [],
        summary_2 = "",
        keywords_2 = [];
    if (settings.checkingOnComp || settings.showKeyWords) {
        [summary_1, keywords_1] = summarize(settings, p[0], ut[0]);
        [summary_2, keywords_2] = summarize(settings, p[1], ut[1]);
    }
    return {
        summary_1 : summary_1,
        summary_2 : summary_2,
        keywords_1 : keywords_1,
        keywords_2 : keywords_2
    };
}
```

base.js

```
export default function getUniqueTokens(paragraphs) {
    let allTokens = concatTokens(paragraphs);
    allTokens = deleteDuplicates(allTokens);
    return allTokens;
}

export function concatTokens(paragraphs) {
    let allTokens = [];
    for (let i = 0; i < paragraphs.length; i++) {
        for (let j = 0; j < paragraphs[i].length; j++) {
            for (let k = 0; k < paragraphs[i][j].tokens.length; k++){
                allTokens = allTokens.concat(paragraphs[i][j].tokens[k]);
            }
        }
    }
```

```
    return allTokens;
}

export function concatSentences(paragraphs) {
    let sentences = [];
    for (let i = 0; i < paragraphs.length; i++) {
        for (let j = 0; j < paragraphs[i].length; j++) {
            sentences.push([i, j, paragraphs[i][j]]);
        }
    }
    return sentences;
}

function deleteDuplicates(tokens) {
    let seen = {};
    let out = [];
    let len = tokens.length;

    let j = 0;
    for(let i = 0; i < len; i++) {
        let item = tokens[i];
        if(seen[item.stemmedName] !== 1) {
            seen[item.stemmedName] = 1;
            out[j++] = item;
        }
    }
    return out;
}
```

detectSimilarity.js

```
import findParagraphs, {makeTokens} from "./tokenize";
import {concatTokens} from "./base";
import crc32 from 'crc/crc32';
import stringSimilarity from "string-similarity";

export default function detectSimilarity(files, summary_1, summary_2, settings) {
```

```
    if (settings.checkingOnComp === true && summary_1 !== "" && summary_2 !== "") {
        return settings.plagiarismMethod === 'dice'?
            stringSimilarity.compareTwoStrings(summary_1, summary_2) * 100 :
            compareByShingle(summary_1, summary_2, settings.shingleLength);

    } else {
        return settings.plagiarismMethod === 'dice'?
            stringSimilarity.compareTwoStrings(files[0].value, files[1].value) * 100 :
            compareByShingle(files[0].value, files[1].value, settings.shingleLength);
    }
}

function compareByShingle(text_1, text_2, shingleLength) {
    [text_1, text_2] = [normalizeText(text_1), normalizeText(text_2)];
    let [shingles_1, shingles_2] = [createShingles(text_1, shingleLength), createShingles(text_2,
shingleLength)];
    return compareShingles(hashShingles(shingles_1), hashShingles(shingles_2));
}

function normalizeText(text) {
    let paragraphs = findParagraphs(text);
    makeTokens(paragraphs);
    let textArray = concatTokens(paragraphs);
    let normalizedText = [];
    for (let i = 0; i < textArray.length; i++) {
        normalizedText.push(textArray[i].stemmedName);
    }
    return normalizedText;
}

function createShingles(text, shingleLength) {
    let shingles = [];
    let textLength = text.length;
    while(shingles.length !== (textLength - shingleLength + 1)) {
        shingles.push(text.slice(0, shingleLength).join(' '));
        text = text.slice(1);
    }
```

```
    return shingles;
}

function hashShingles(shingles) {
    let hashes = [];
    for(let i = 0, n = 1; i < n; i++) {
        let hashedArr = [];
        for(let j = 0, k = shingles.length; j < k; j++) {
            hashedArr.push(crc32(shingles[j]));
        }
        hashes.push(hashedArr);
    }
    return hashes;
}

function compareShingles(hashes_1, hashes_2) {
    let count = 0;
    hashes_1[0].forEach(function(item) {
        if(hashes_2[0].indexOf(item) !== -1) {
            count++;
        }
    });

    return count*2/(hashes_1[0].length + hashes_2[0].length)*100;
}
```

prepareFiles.js

```
import findParagraphs, {makeTokens} from "./tokenize.js";
import getUniqueTokens, {concatSentences} from "./base.js";


export default function prepareFiles(data) {
    let result = [];
    for (let i = 0; i < data.length; i++) {
        result.push(processFile(data[i].value));
    }
```

```
    return {
        "sentenceLimit": result[0].numSentences < result[1].numSentences ? result[0].numSentences :
result[1].numSentences,
        "wordsLimit": result[0].numUnique < result[1].numUnique ? result[0].numUnique :
result[1].numUnique,
        "paragraphs_1": result[0].paragraphs,
        "paragraphs_2": result[1].paragraphs,
        "uniqueTokens_1": result[0].uniqueTokens,
        "uniqueTokens_2": result[1].uniqueTokens
    };
}

function processFile(data) {
    let paragraphs = findParagraphs(data);
    makeTokens(paragraphs);
    let sentences = concatSentences(paragraphs);
    let uniqueTokens = getUniqueTokens(paragraphs);

    return {
        "numSentences": sentences.length,
        "numUnique": uniqueTokens.length,
        "paragraphs": paragraphs,
        "uniqueTokens": uniqueTokens
    };
}
```

stem.js

```
const PERFECT_GROUND = /(ив|ивши|ившись|ыв|ывши|ывшись(в|вши|вшись))$/;
const REFLEXIVE = /(с[яьи])$/;  //Рефлексивне_дієслово
const ADJECTIVE = /(ими|ій|ий|а|е|ова|ове|ів|є|їй|єє|еє|я|ем|им|ім|их|іх|ою|йми|іми|у|ю|ого|ому|ої)$/;
const PARTICIPLE = /(ий|ого|ому|им|ім|а|ій|у|ою|і|йми|их)$/; //Дієприкметник
const VERB = /(сь|ся|ив|ать|ять|у|ю|ав|али|учи|ячи|вши|ши|е|ме|ати|яти|є)$/;
const NOUN =
/(а|ев|ов|е|ями|ами|еи|и|ей|ой|ий|й|иям|ям|ием|ем|ам|ом|о|у|ах|иях|ях|ы|ь|ию|ью|ю|ия|ья|я|і|ові|ї|ею|єю|о
ю|є|еві|єм|ів|їв|'ю)$/;
```

```
const VOWELS = /^(.*?[аеиоуюяіїє])(.*)$/;
const DERIVATION = /[^аеиоуюяіїє][аеиоуюяіїє]+[^аеиоуюяіїє]+[аеиоуюяіїє].*сть?$/;

export default function stem (word) {
    if (word == null || !word.length) {
        return word;
    }
    let stem = word.toLowerCase();
    do {
        let v = word.match(VOWELS);
        if (!v) break;

        let start = v[1];
        let RV = v[2];
        if (!RV) break;



        let m = RV.replace(PERFECT_GROUND, '');
        if (m === RV) {
            RV = RV.replace(REFLEXIVE, '');

            m = RV.replace(ADJECTIVE, '');
            if (m === RV) {
                RV = RV.replace(PARTICIPLE, '');
            } else {
                RV = m;
                m = RV.replace(VERB, '');
                if (m === RV) {
                    RV = RV.replace(NOUN, '');
                } else {
                    RV = m;
                }
            }
        } else {
            RV = m;
        }
```

```
        RV = RV.replace(/и$/, '');



        if (DERIVATION.test(RV)) {
            RV = RV.replace(/ость?$/, '');
        }



        m = RV.replace(/ь$/, '');
        if (m === RV) {
            RV = RV.replace(/ейше?/, '');
            RV = RV.replace(/нн$/, 'н');
        } else {
            RV = m;
        }

        stem = start + RV;
    } while (false);
    return stem;
}
```

summarize.js

```
import calcTF_IDF from "./tf_idf.js";
import {concatSentences} from "./base.js";

export default function summarize(settings, paragraphs, uniqueTokens) {
    let summary = "";
    let keywords = [];
    calcTF_IDF(paragraphs, uniqueTokens);
    evaluateSentences(paragraphs);

    if (settings.checkingOnComp === true) {
        summary = getSummary(concatSentences(paragraphs), settings.compSize);
    }
    if (settings.showKeyWords === true){
```

```
        keywords = sortKeyWords(uniqueTokens).slice(0, settings.keyWordsCounter);
        keywords = getKeywordValues(keywords);
    }
    return [summary, keywords];
}

function getKeywordValues(keywords) {
    let keywordsValues = [];
    for (let i = 0; i < keywords.length; i++) {
        keywordsValues.push(keywords[i].name);
    }
    return keywordsValues;
}

function getSummary(sentences, compSize) {
    sentences = getRelevantSentences(sentences, compSize);
    sentences = sortRelevantSentences(sentences);
    let summary = "";
    for (let i = 0; i < sentences.length; i++) {
        summary += sentences[i][2].value + " ";
    }
    return summary;
}

function getRelevantSentences(sentences, compSize) {
    let sentenceArray = [];
    let counter = 0;
    while (compSize > counter) {
        let highest = [0, 0, 0];
        for (let i = 0; i < sentences.length; i++) {
            if (sentences[i][2].relevance > highest[1]) {
                highest = [i , sentences[i][2].relevance, sentences[i]];
            }
        }
        sentenceArray.push(highest[2]);
        sentences.splice(highest[0], 1);
        counter++;
```

```
    }
    return sentenceArray;
}

function sortRelevantSentences(sentences) {
    return sentences.sort(function (x, y) {
        let n = x[0] - y[0];
        if (n !== 0) {
            return n;
        }
        return x[1] - y[1];
    });
}

function sortKeyWords(uniqueTokens) {
    return uniqueTokens.sort(function (x, y) {
        if ( x.frequency > y.frequency ){
            return -1;
        }
        if ( x.frequency < y.frequency ){
            return 1;
        }
        return 0;
    });
}

function evaluateSentences(paragraphs)  {
    for (let i = 0; i < paragraphs.length; i++) {
        for (let j = 0; j < paragraphs[i].length; j++) {
            let mark_relevance = 0;
            for (let k = 0; k < paragraphs[i][j].tokens.length; k++) {
                mark_relevance += paragraphs[i][j].tokens[k].frequency;
            }
            paragraphs[i][j].relevance = mark_relevance/paragraphs[i][j].tokens.length;
        }
    }
}
```

tf_idf.js

```
import {concatTokens} from "./base";
import { frequencies } from '../data/ua_freq.js';

export default function calcTF_IDF(paragraphs, tokens) {
    const properNames = findProperNames(tokens);
    let [sentenceQuantity, questExclQuantity, firstLastQuantity] = calcQuantitySentences(paragraphs);

    tokens = concatTokens(paragraphs);
    for (let i = 0; i < tokens.length; i++) {
        let quantity = 0;
        for (let j = 0; j < tokens.length; j++) {
            if (tokens[i].stemmedName === tokens[j].stemmedName) {
                quantity++;
            }
        }
        calcByTFIDF(tokens[i], quantity, tokens.length);

        if (paragraphs.length >= 3 && sentenceQuantity >= 10) {
            calcBySpecialSentences(tokens[i], sentenceQuantity, questExclQuantity, firstLastQuantity);
        }

        if (tokens[i].name === properNames) {
            calcByProperName(tokens[i]);
        }
    }
}

function calcQuantitySentences(paragraphs) {
    let sentenceQuantity = 0;
    let questExclQuantity = 0;
    let firstLastQuantity = 0;
    for (let i = 0; i < paragraphs.length; i++) {
        for (let j = 0; j < paragraphs[i].length; j++) {
            if (paragraphs[i][j].isQuestionExclamation) {
                questExclQuantity += 1;
```

```
            }
            if (paragraphs[i][j].isFirstLast) {
                firstLastQuantity += 1;
            }
            sentenceQuantity += 1;
        }
    }
    return [sentenceQuantity, questExclQuantity, firstLastQuantity];
}

function calcBySpecialSentences(token, sentenceQuantity, questExclQuantity, firstLastQuantity) {
    if (token.inQuestionExclamation) {
        token.frequency *= sentenceQuantity/questExclQuantity;
    }
    if (token.inFirstLast) {
        token.frequency *= sentenceQuantity/firstLastQuantity;
    }
}

function calcByTFIDF(token, quantity, tokenLength){
    // TF in TF-IDF
    token.frequency = quantity/tokenLength;
    // IDF in TF-IDF
    let dict_freq = getTokenFrequency(token.stemmedName);
    if (dict_freq !== 0) {
        token.frequency *= Math.log10(dict_freq);
    }
}

function getTokenFrequency(tokenStemmedName) {
    const freqDict = frequencies;
    for (let i = 0; i < freqDict.length; i++){
        if (freqDict[i].stemmedWord === tokenStemmedName){
            return freqDict[i].rel_freq;
        }
    }
    return 0;
```

```
}

function calcByProperName(token) {
   token.frequency *= Math.log10(15);
}

function findProperNames(tokens) {
   const capital = /[ЙЦУКЕНГШЩЗХЇҐФІВАПРОЛДЖЄЯЧСМИТЬБЮА-
Z$]+[йцукенгшщзхїґфівапролджєячсмитьбюа-z]*/g
   let properNames = [];
   for (let i = 0; i < tokens.length; i++) {
      if (i !== 0 && !tokens[i].name.search(capital) && tokens[i].isFirst === false) {
         properNames.push(tokens[i]);
      }
   }
   return properNames;
}
```

tokenize.js

```
import stem from "./stem.js";
import {stop_words} from "../data/stopWords";
import replaceAbbreviations from "../data/abbreviations";

export default function findParagraphs(data) {
   data = replaceAbbreviations(data);
   data = data.split("\n\r| [\n]*");
   let paragraphs = [];
   for (let i = 0; i < data.length; i++) {
      while (data[i].charAt(0) === "\n") {
         data[i] = data[i].slice(1);
      }
      if (data[i] !== null) paragraphs.push(findSentences(data[i]));
   }
   findSpecialSentences(paragraphs);
   return paragraphs;
}
```

```
export function findSentences(data) {
    let sentences = data.match(/[^.!?\n\r]+[.\n\r!?]+/g);
    let result = [];
    if (sentences !== null) {
        for (let i = 0; i < sentences.length; i++) {
            while (sentences[i].charAt(0) === " ") {
                sentences[i] = sentences[i].slice(1);
            }
            sentences[i] = sentences[i].replace(/\r\n/g, "")
            let sentence = makeSentence(sentences[i]);
            if (sentence.value !== "") {
                result.push(sentence);
            }
        }
    }
    return sentences !== null ? result : [makeSentence(data + '.')];
}

export function makeTokens(paragraphs) {
    for (let i = 0; i < paragraphs.length; i++) {
        for (let j = 0; j < paragraphs[i].length; j++) {
            let tokens = [];
            tokens = tokenize(paragraphs[i][j].value);
            tokens = removeStopWords(tokens, stop_words);
            extractTokens(tokens, paragraphs[i][j]);
        }
    }
    addTokenInfoAboutSentences(paragraphs);
}

function makeToken(name, stemmedName, isFirst) {
    return {
        name,
        stemmedName,
        isFirst,
        frequency: 0,
```

```
        inFirstLast: false,
        inQuestionExclamation: false
    };
}

function makeSentence(value) {
    return {
        value,
        isFirstLast: false,
        isQuestionExclamation: false,
        tokens: [],
        relevance: 0
    };
}

function extractTokens(tokens, sentence) {
    const capital = /[ЙЦУКЕНГШЩЗХЇҐФІВАПРОЛДЖЄЯЧСМИТЬБЮА-
Z$]+[йцукенгшщзхїґфівапролджєячсмитьбюа-z]*/g
    let isFirst = false;
    for (let i = 0; i < tokens.length; i++) {
        i === 0 && !tokens[i].search(capital) ? isFirst = true : isFirst = false;
        tokens[i] = makeToken(tokens[i], stem(tokens[i]), isFirst);
        sentence.tokens.push(tokens[i]);
    }
}

function addTokenInfoAboutSentences(paragraphs) {
    for (let i = 0; i < paragraphs.length; i++) {
        for (let j = 0; j < paragraphs[i].length; j++) {
            if (paragraphs[i][j].isFirstLast && paragraphs[i][j].isQuestionExclamation) {
                for (let k = 0; k < paragraphs[i][j].tokens.length; k++) {
                    paragraphs[i][j].tokens[k].inFirstLast = true;
                    paragraphs[i][j].tokens[k].inQuestionExclamation = true;
                }
            }
            if (paragraphs[i][j].isFirstLast) {
                for (let k = 0; k < paragraphs[i][j].tokens.length; k++) {
```

```
                paragraphs[i][j].tokens[k].inFirstLast = true;
            }
        }
        if (paragraphs[i][j].isQuestionExclamation) {
            for (let k = 0; k < paragraphs[i][j].tokens.length; k++) {
                paragraphs[i][j].tokens[k].inQuestionExclamation = true;
            }
        }
    }
  }
}

function tokenize(data) {
  return data.match(/[^,.;:!?—«»"()\s]+[^.,;:!?—«»"()\s]*/g);
}

function findSpecialSentences(paragraphs) {
  for (let i = 0; i < paragraphs.length; i++) {
    if (paragraphs[i].length > 1) {
      paragraphs[i][0].isFirstLast = true;
      paragraphs[i][paragraphs[i].length-1].isFirstLast = true;
    }
    for (let j = 0; j < paragraphs[i].length; j++)
    {
      if (!paragraphs[i][j].value.search(/[^.!?\r\n]+[!?]+/g)) paragraphs[i][j].isQuestionExclamation =
true;
    }
  }
}

function removeStopWords(dataArray, stopWords) {
  dataArray = dataArray.filter((el) => !stopWords.includes( el.toLowerCase() ));
  return dataArray;
}
```

App.js

```
import React, {useEffect, useState} from "react";
import About from "./sections/About";
import {Route, useLocation} from "react-router";
import Main from "./sections/Main";
import Header from "./components/Header";
import './scss/index.scss'
import Settings from "./sections/Settings";
import Treatment from "./sections/Treatment";
import Results from "./sections/Results";
import NavigationDesk from "./components/NavigationDesk";



const App = () => {

    const location  = useLocation();

    useEffect(() => {
        !localStorage.getItem('history') && localStorage.setItem('history',JSON.stringify([]));
    });

    const [filesStore, setFilesStore] = useState([]);
    const [paragraphsStore, setParagraphsStore] = useState([]);
    const [uniqueTokensStore, setUniqueTokensStore] = useState([]);
    const [sentenceLimit, setSentenceLimit] = useState(0);
    const [wordsLimit, setWordsLimit] = useState(0);
    const [newTexts, setNewTexts] = useState([","]);
    const [settings,setSettings] = useState({
        compSize: 7,
        showKeyWords: true,
        keyWordsCounter: 10,
        checkingOnComp: true,
        showWordsCounter: false,
        showSignCounter: false,
        plagiarismMethod: 'dice',
        shingleLength: 5,
```

```
  });
  const [percent, setPercent] = useState(-1);
  const [keyWords, setKeyWords] = useState([
    ['lorem','ipsum','dolor','sit','amet'],
    ['consectetur', 'adipiscing', 'elit', 'suspendisse', 'quis']
  ]);

  return (
    <>
      <Header />
      <div className="app">
        <Route exact path={'/'}>
          <About />
        </Route>
        <Route exact path={'/main'}>
          <Main setFilesStore={setFilesStore} setSentenceLimit={setSentenceLimit}
setWordsLimit={setWordsLimit}
              setParagraphsStore={setParagraphsStore}
setUniqueTokensStore={setUniqueTokensStore}/>
        </Route>
        <Route exact path={'/settings'}>
          <Settings setSettings={setSettings} sentenceLimit={sentenceLimit}
wordsLimit={wordsLimit}/>
        </Route>
        <Route exact path={'/treatment'}>
          <Treatment setNewTexts={setNewTexts} setPercent={setPercent}
setKeyWords={setKeyWords}
                files={filesStore} settings={settings} paragraphsStore={paragraphsStore}
uniqueTokensStore={uniqueTokensStore}/>
        </Route>
        <Route exact path={'/results'}>
          <Results newTexts={newTexts} keyWords={keyWords} percent={percent}
files={filesStore} settings={settings} />
        </Route>
        {location.pathname !== '/' && <NavigationDesk />}
      </div>
    </>
```

```
    );
};

export default App;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import reportWebVitals from './reportWebVitals';
import {Provider} from "react-redux";
import store from "./store/store";
import {HashRouter} from "react-router-dom";


ReactDOM.render(
  <React.StrictMode>
     <HashRouter
        hashType={"slash"}
        basename={"s-check"}
     >
       <Provider store={store}>
         <App />
       </Provider>
     </HashRouter>
  </React.StrictMode>,
  document.getElementById('root')
);

reportWebVitals();
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО	**ЗАТВЕРДЖУЮ**
Керівник проєкту	**В.о. завідувача кафедри**
_____ *Олексій ФІНОГЕНОВ*	_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)	(підпис) (вл. ім’я, прізвище)
“5” квітня 2021 р.	“6” квітня 2021 р.

Інформаційна система з визначення близькості документів на основі стиснення текстів

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 7325.01.000 ТЗ*

на 8 сторінках

Київ – 2021 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи: «Інформаційна система з визначення близькості документів на основі їх стиснення».

Коротке найменування системи: «Summarize & Check».

1.2 Найменування організації-замовника та організацій учасників робіт

Замовником є кафедра автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут» ім. Ігоря Сікорського (далі за текстом — Замовник).

Адреса замовника: м. Київ, п. Перемоги 37, корпус №18 ФІОТ.

Виконавець – студентка групи ІС-73 кафедри АСОІУ НТУУ «КПІ» ім. Ігоря Сікорського Шеляхіна Владислава Владиславівна.

1.3 Перелік документів, на підставі яких створюється система

Під час розробки системи «Summarize & Check» і створення проектно-експлуатаційної документації Розробник сервісу має оперувати вимогами наступних нормативних документів:

- ДСТУ 3973-2000 «Система розроблення та поставлення продукції на виробництво»;
- ДСТУ 34.601-90 «Інформаційна технологія. Комплекс стандартів на автоматизовані системи».

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку робіт – 12 квітня 2021 року.

Плановий термін закінчення робіт – 15 травня 2021 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Система визначення близькості документів на основі стиснення текстів призначена для проведення досліджень, які базуються на виявлення ступеню подібності документів на основі їх найбільш важливих фрагментів.

Даний проєкт може бути використаний для завдань галузей, які потребують вирішення задач, що пов’язані із захистом об’єктів авторського права, пошуком інформації, а також у дослідженнях науковців у галузі математичної лінгвістики.

Крім того, враховуючи проблематичну роботу наявних аналогів із текстами українською мовою, ця розробка може бути використана для створення авторефератів та перевірки українських текстів на близькість.

2.2 Цілі створення системи

Цілі створення інформаційної системи визначення близькості документів на основі стиснення текстів:

- підвищення швидкості виявлення схожості між текстами за рахунок зменшення обсягів текстів документів, які надходять на обробку;
- зниження вимог до необхідних обчислювальних ресурсів;
- автоматизація процесу створення авторефератів україномовних документів;
- автоматизація процесу перевірки україномовних документів на схожість.

Головною метою розробки системи є підвищення ефективності порівняння текстів за рахунок використання алгоритмів автоматичного реферування.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Об'єктами автоматизації є процеси генерації анотацій та перевірки на схожість текстів українською мовою.

Таким чином, користувачі даної системи мають отримувати доступ до наступного функціоналу:

- завантаження документів для обробки та порівняння за допомогою області drag'n'drop або відкриттям програми «Провідник»;
- введення текстів для обробки та порівняння власноруч;
- видалення документів;
- обрання бажаного розміру очікуваного стиснення тексту;
- додавання чи видалення зі звіту інформації про кількість ключових слів, отриманих під час процесу обробки;
- надання кількості потрібних ключових слів, якщо дана функція ввімкнена;
- можливість додавання до звіту інформації про кількість слів та кількість символів завантажених документів;
- ввімкнення та відключення налаштування, яке стикає документи перед визначенням їх коефіцієнту схожості;
- перегляд згенерованого звіту;
- завантаження звіту;
- ознайомлення зі стиснутими версіями завантажених документів;
- перегляд інформації про методи користування застосунком;
- перегляд історії запитів до системи.

4 ВИМОГИ ДО СИСТЕМИ

4.1 Вимоги до системи в цілому

Розроблена система має відповідати наступним вимогам:

- оброблювати великі об’єми даних;
- бути спроектованою за архітектурою Flux;
- бути придатною для введення нового функціоналу.

4.2 Вимоги до функціональних характеристик

Функції розробленого програмного продукту мають працювати справно, а помилки повинні оброблюватися без незапланованого завершення роботи програми.

4.3 Вимоги до видів забезпечення

Даний програмний продукт представляється у вигляді веб-застосування для визначення схожості документів на основі стиснення їх змісту.

Склад технічних засобів визначається наявністю ЕОМ, що має задовольняти наступні вимоги:

- операційна система Windows 7, або OS X El Capitan (10.11), або Ubuntu 14.04 та вище;
- наявність оперативної пам’яті місткістю не менше 4 GB;
- встановлений графічний пристрій DirectX 9 з драйвером WDDM 1.0 або вище;
- розширення екрану не менше SVGA 800×600;
- встановлений Node.js;
- підключення до мережі Інтернет;
- наявність сучасного браузеру;
- мережеве підключення.

5 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

У таблиці 5.1 наведено календарний план виконання робіт та терміни їх виконання.

Таблиця 5.1 – Календарний план виконання робіт

№	Етап створення програмного продукту	Термін виконання	Результат виконання
1	Ознайомлення із постановкою задачі	12.04.2021	Постановка завдання проаналізована
2	Збір та огляд спеціалізованої літератури	15.04.2021	Предметна область досліджена
3	Огляд існуючих програмних аналогів	17.04.2021	Виявлено переваги і недоліки конкурентів
4	Огляд методів вирішення поставленої задачі	20.04.2021	Досліджено алгоритми для вирішення задачі
5	Пошук колекцій слів української мови	22.04.2021	Необхідні колекції слів знайдено
7	Розробка макету продукту	27.04.2021	Розроблено макет та узгоджено з замовником
6	Розробка програмного продукту	10.05.2021	Розробка сервісу завершена
7	Тестування та налагодження програми	12.05.2021	Застосунок успішно пройшов випробування
8	Розробка експериментів для дослідження роботи алгоритмів	13.05.2021	Експерименти показали ефективність алгоритмів
9	Формування звіту	14.05.2021	Звіт оформлено
10	Надання на перевірку готового програмного продукту	15.05.2021	Готовий програмний продукт надано замовнику

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

6.1 Види випробувань

Випробування узгоджуються із замовником до початку їх проведення. Надання результатів робіт виконується згідно з робочою програмою та календарним планом.

Готовий програмний продукт має пройти перевірки випробуваннями двох типів:

- тестові випробування – демонструються замовнику; показують, як в цілому працює програма;
- експериментальні випробування – проводяться на ЕОМ замовника для проведення аналізу наданої текстової інформації.

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008107309

Дата перевірки:
31.05.2021 17:38:50 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2021 00:00:18 EEST

ID користувача:
77149

Назва документа: **Shelyakhina_bachelor_is73**

Кількість сторінок: **81** Кількість слів: **10644** Кількість символів: **89723** Розмір файлу: **4.09 MB** ID файлу: **1008191045**

8.43%
Схожість

Найбільша схожість: **1.66%** з джерелом з Бібліотеки (ID файлу: **1008178208**)

5.59% Джерела з Інтернету	171	Сторінка 83
6.93% Джерела з Бібліотеки	349	Сторінка 85

0% Цитат

Не знайдено жодних цитат

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	4

Графічний матеріал
до дипломного проєкту

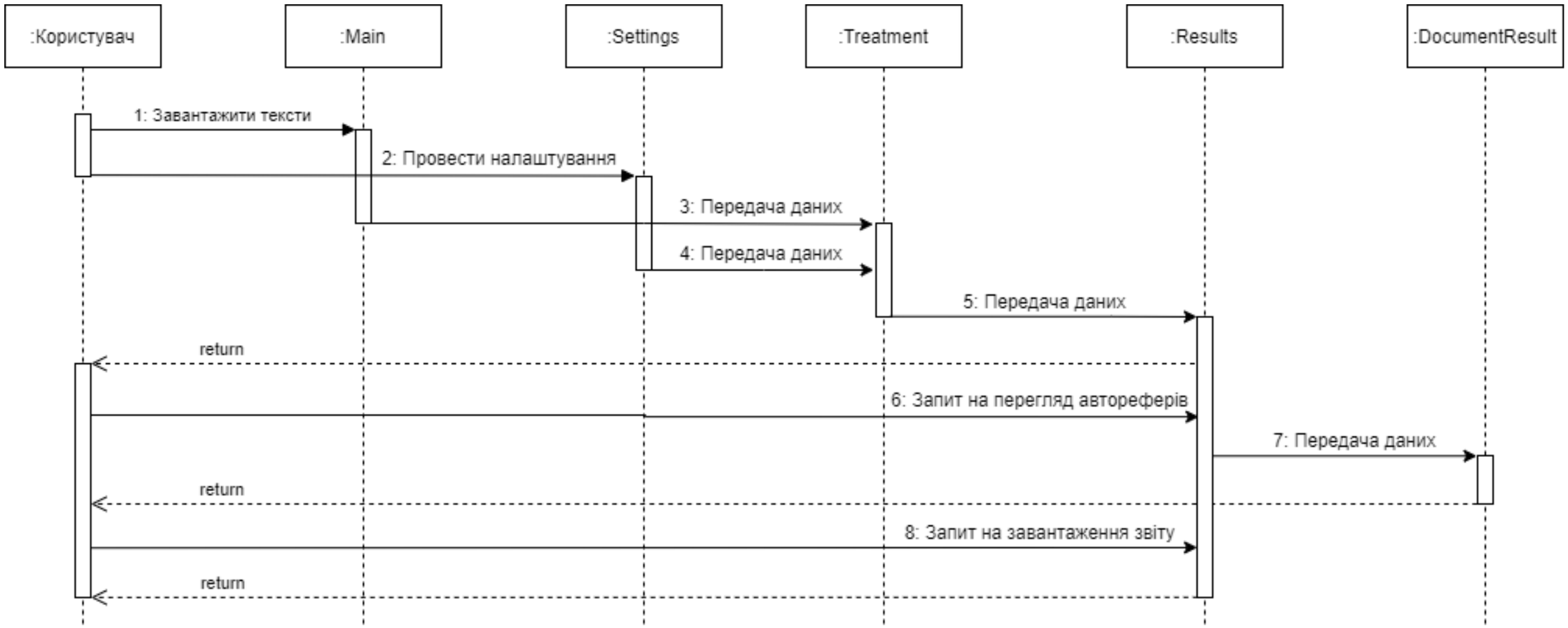
на тему: *Інформаційна система з визначення близькості документів на основі стиснення текстів*

Київ – 2021 року

					ДП 7325.02.000 ССД		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера	Маса / Масштаб
Розробив		Шеляхіна В.В.					
Перевірив		Фіногенов О.Д.				Аркуш 1	Аркушів 1
Т. кон.					Інформаційна система з визначення близькості документів на основі стиснення текстів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73	
Н. кон.		Сперкач М.О					
Затвердив		Фіногенов О.Д.					

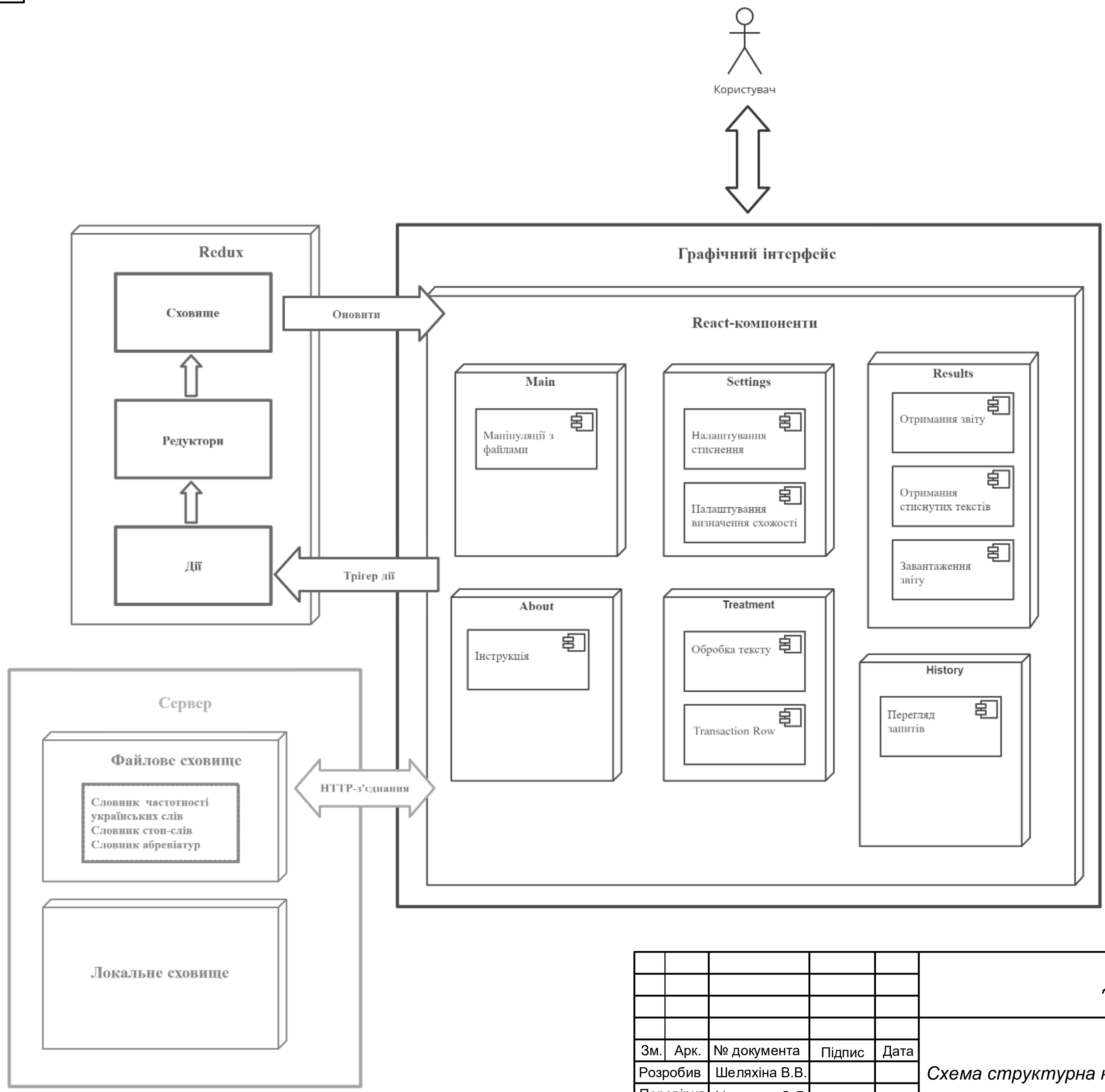
ДП 7325.03.000 ССВ

					ДП 7325.03.000 ССВ		
					Схема структурна варіантів використань	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Шеляхіна В.В.				Аркуш 1	Аркушів 1
Перевірив		Фіногенов О.Д.					
Т. кон.					*Інформаційна система з визначення близькості документів на основі стиснення текстів*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73*	
Н. кон.		Сперкач М.О.					
Затвердив		Фіногенов О.Д.					



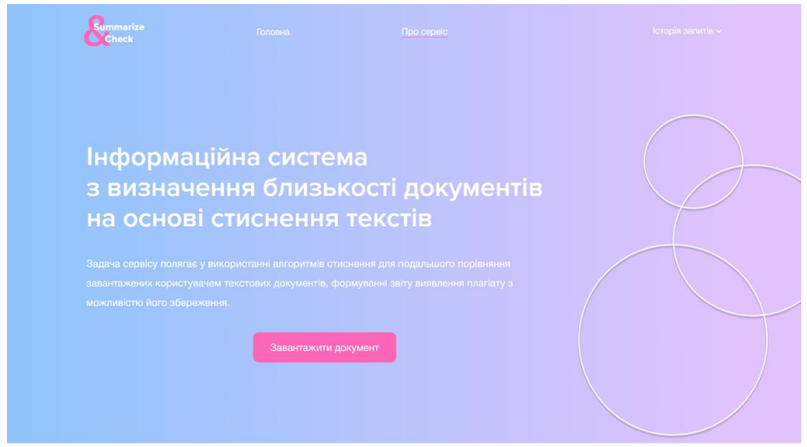
					ДП 7325.04.000 ССП		
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	*Схема структурна послідовності*			
Розробив		Шеляхіна В.В.				Аркуш 1	Аркушів 1
Перевірив		Фіногенов О.Д.					
Т. кон.							
					Інформаційна система з визначення близькості документів на основі стиснення текстів	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73*	
Н. кон.		Сперкач М.О.					
Затвердив		Фіногенов О.Д.					

ДП 7325.05.000 СК

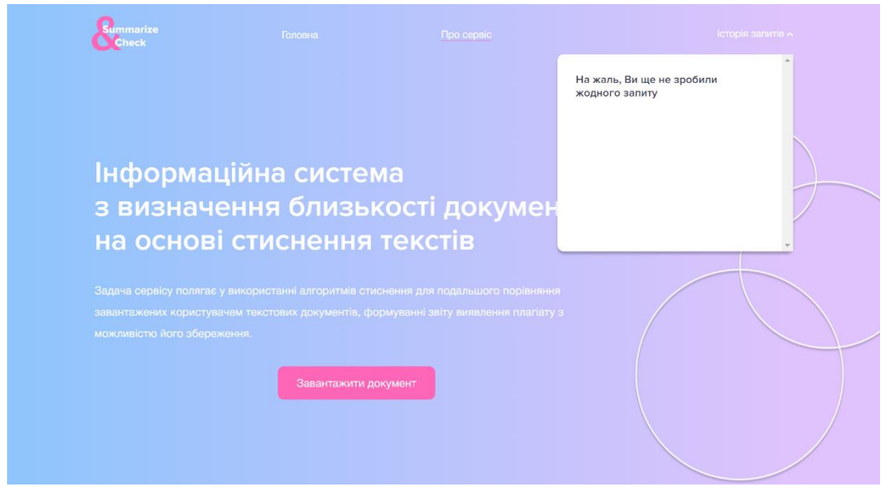


					ДП 7325.05.000 СК		
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Шеляхіна В.В.			Схема структурна компонентів		
Перевірив		Фіногенов О.Д.					
Т. кон.						Аркуш 1	Аркушів 1
					Інформаційна система з визначення близькості документів на основі стиснення текстів	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73	
Н. кон.		Сперкач М.О.					
Затвердив		Фіногенов О.Д.					

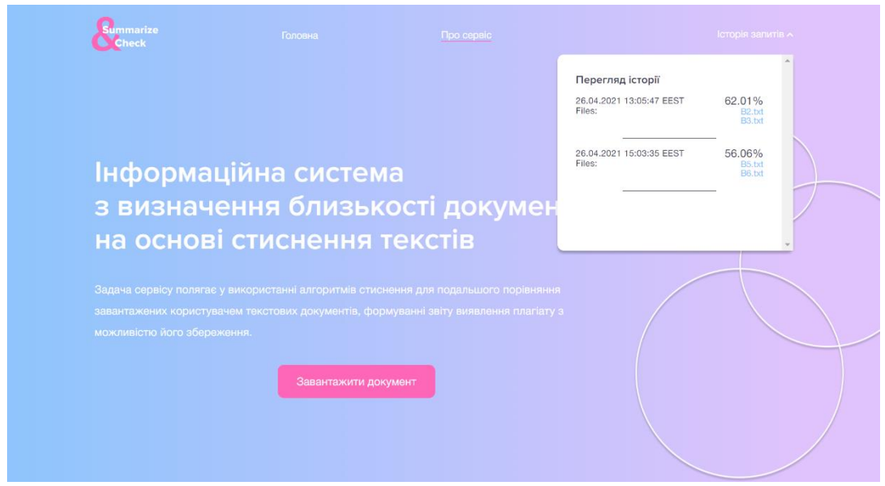
Екранна форма сторінки «Про Сервіс»



Екранна форма незаповненого меню «Історія запитів»

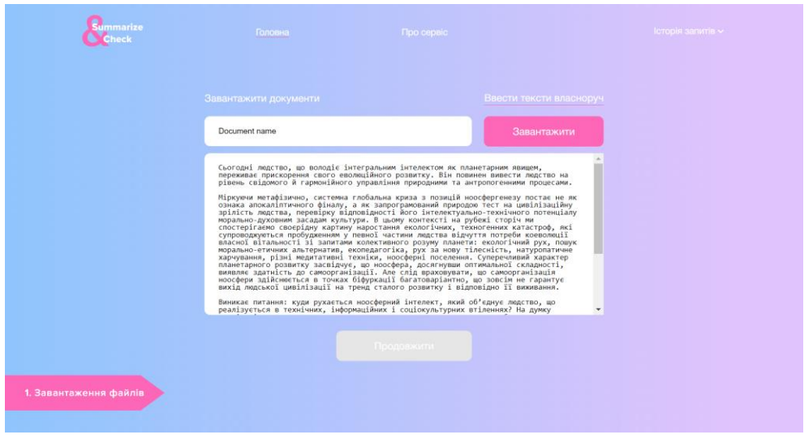


Екранна форма заповненого меню «Історія запитів»

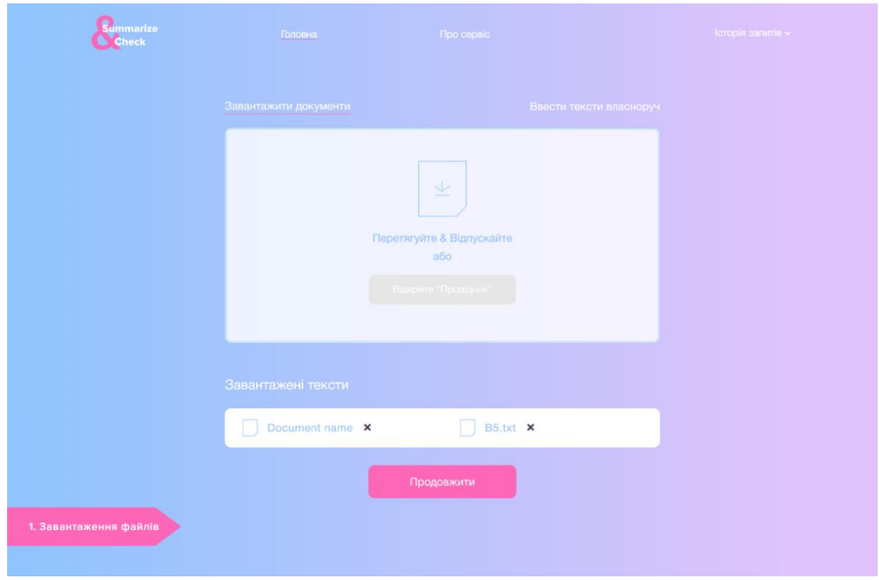


					ДП 7325.06.000 КЕ		
Зм.	Арк.	№ документа	Підпис	Дата	*Креслення вигляду екранних форм*	Літера	Маса / Масштаб
Розробив		Шеляхіна В.В.					
Перевірив		Фіногенов О.Д.				Аркуш 1	Аркушів 3
Т. кон.							
Н. кон.		Сперкач М.О.			*Інформаційна система з визначення близькості документів на основі стиснення текстів*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73*	
Затвердив		Фіногенов О.Д.					

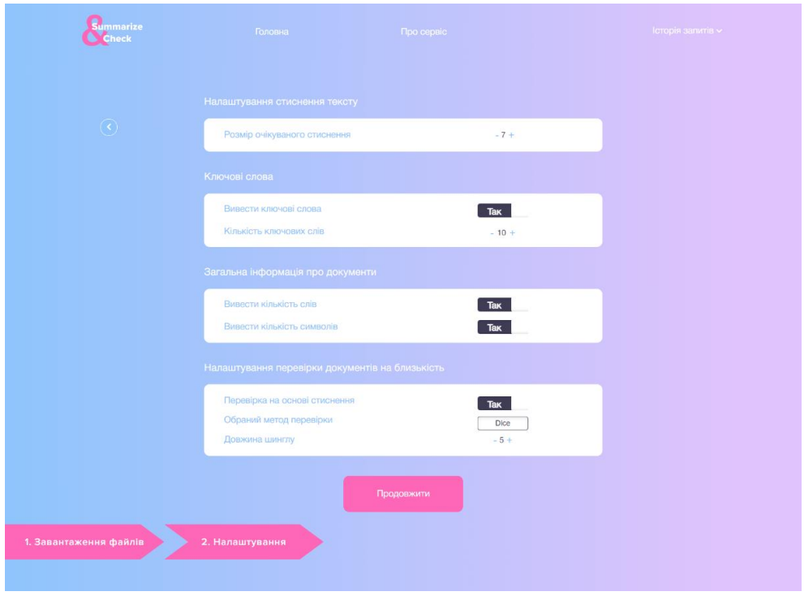
Екранна форма сторінки введення текстів власноруч



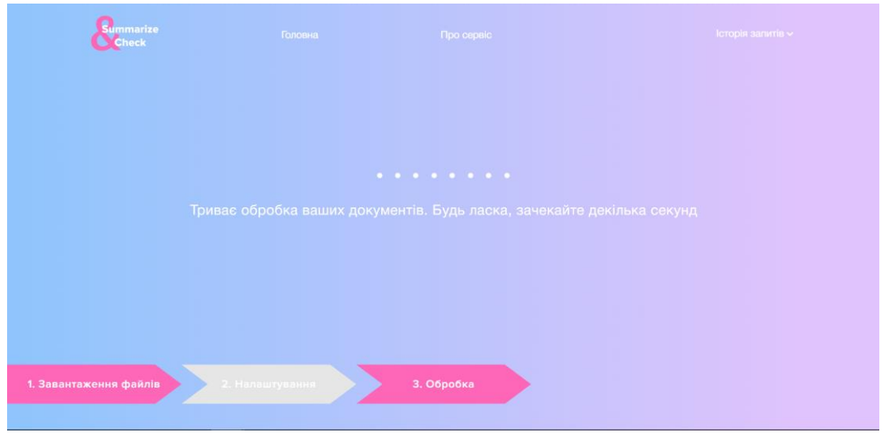
Екранна форма сторінки завантаження документів



Екранна форма сторінки налаштувань

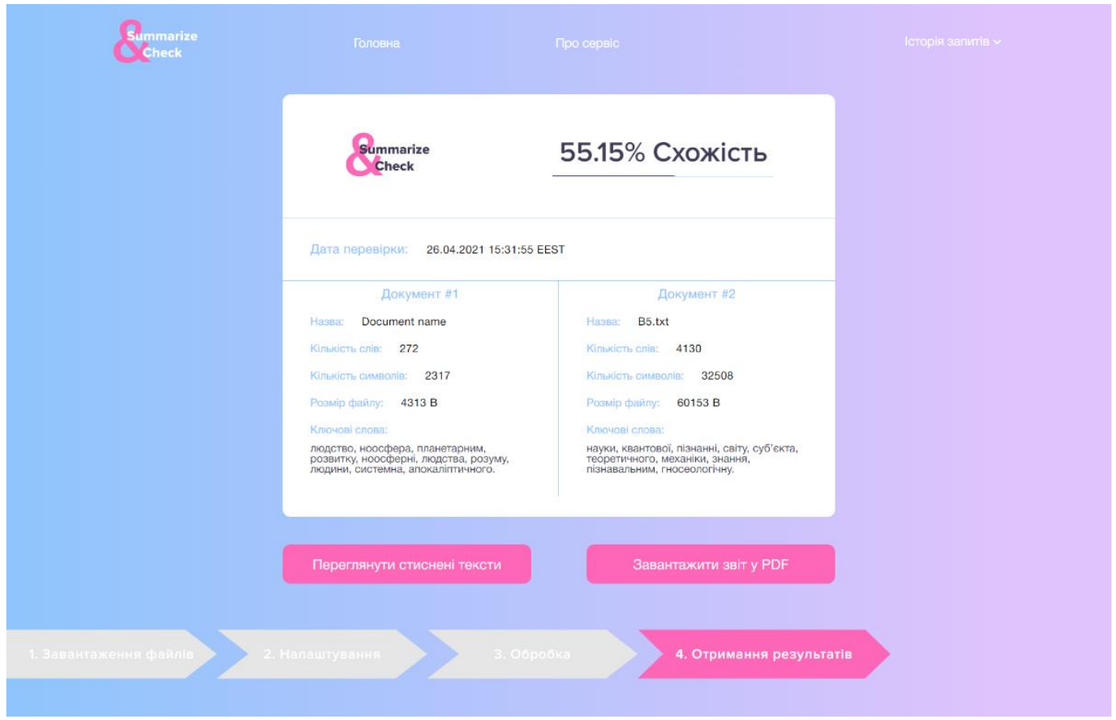


Екранна форма сторінки очікування закінчення обробки

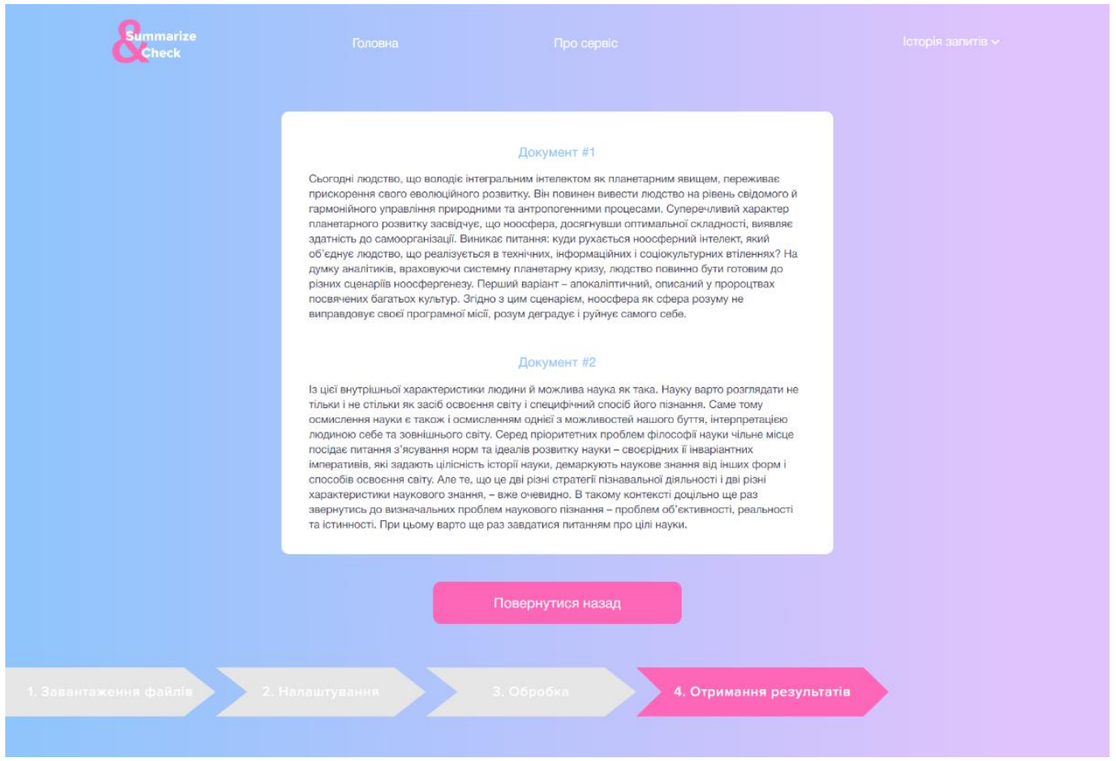


					ДП 7325.06.000 КЕ	
Зм.	Арк.	№ документа	Підпис	Дата	*Креслення вигляду екранних форм*	Літера / Маса / Масштаб
Розробив		Шеляхіна В.В.				
Перевірив		Фіногенов О.Д.				Аркуш 2 / Аркушів 3
Т. кон.						
Н. кон.		Сперкач М.О.			*Інформаційна система з визначення близькості документів на основі стиснення текстів*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73*
Затвердив		Фіногенов О.Д.				

Екранна форма сторінки отримання результатів



Екранна форма сторінки перегляду авторефератів



					ДП 7325.06.000 КЕ			
Зм.	Арк.	№ документа	Підпис	Дата	*Креслення вигляду екранних форм*	Літера	Маса	Масштаб
Розробив		Шеляхіна В.В.						
Перевірив		Фіногенов О.Д.				Аркуш 3	Аркушів 3	
Т. кон.								
Н. кон.		Сперкач М.О.			*Інформаційна система з визначення близькості документів на основі стиснення текстів*	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-73*		
Затвердив		Фіногенов О.Д.						