МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет магістерської та аспірантської підготовки
Кафедра інформаційних технологій

КОМПЛЕКСНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розробка системи виявлення плагіату в наукових та студентських роботах»

Склад:

Частина 1. «Серверна частина»

Виконавець: Федючек О.В
(П.І.Б.)

Керівник: Терещенко Т.М.
(П.І.Б.)

Частина 2. «Клієнтська частина»

Виконавець: Щекотілін В.А
(П.І.Б.)

Керівник: Терещенко Т.М.
(П.І.Б.)

Староста роботи: Щекотілін В.А
(П.І.Б.)

Провідний керівник проекту: к.т.н., доц. Терещенко Т.М.
(П.І.Б.)

Рецензент: к.т.н., доц. Гнатовська Г.А.
(П.І.Б.)

ОДЕСА 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет магістерської та
аспірантської підготовки
Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка серверної частини

Виконав студент 2 курсу групи МК-61
спеціальності 8.05010101 Інформаційні
управляючі системи та технології,
Федючек Олег Володимирович

Керівник к.т.н., доцент
Терещенко Тетяна Михайлівна

Рецензент к.т.н., доцент
Гнатовська Ганна Арнольдівна

Одеса 2017

АНОТАЦІЯ

Актуальність даної магістерської роботи полягає в необхідності виявлення плагіату серед робіт різних рівней. На наш час вкрасти чужу роботу стало значно легше і простіше.

Об'єкт дослідження – системи виявлення плагіату.

Мета роботи – розробка і створення системи пошуку плагіату документів, ріноманітних робіт, створення публічного сервісу та зручного інтерфейсу користувача. Для досягнення поставленої мети в роботі були вирішені наступні завдання: проведено порівняльний аналіз існуючих систем; визначені функції, які повинна виконувати система; проаналізовані існуючі методи та засоби реалізації алгоритмів; розроблена система пошуку у Solr; розроблені рекомендації для тестування зручності користувача.

Представлена магістерська робота містить пояснювальної записки – 61 сторінки, малюнків – 23, таблиць – 1, посилань – 14.

Ключові слова: плагіат, анти плагіат, проектування

SUMMARY

The relevance of this master's work is the need to detect plagiarism among various levels of work. In our time, steal someone else's work became easier and easier.

Object of research – plagiarism detection.

Purpose–development and creation of search engines plagiarism documents, works, the creation of public service and convenient user interface. To achieve this goal have been resolved in the following tasks: a comparative analysis of existing systems; defined functions that must perform system; analyzed existing methods and means of implementing algorithms; developed search engine in Solr; recommendations for testing convenience.

Presented master thesis contains: notes – 61, pictures – 23, tables – 1, references–14.

Keywords: plagiarism, anti plagiarism, design

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Магістерської та аспірантської підготовки
Кафедра інформаційних технологій
Рівень вищої освіти магістр
Спеціальність 8.05010101 Інформаційні управляючі системи та технології
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____

"____" _____201__ року

З А В Д А Н Н Я
МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Федючек Олег Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка серверної частини

керівник роботи Терещенко Тетяна Михайлівна, к.т.н, доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від _____
2. Строк подання студентом роботи 1 лютого 2017 р.
3. Вихідні дані до роботи Ресурси мережі Інтернет, СУБД Solr/Lucene, мова програмування Scala

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предмету та цілей системи виявлення плагіату в наукових та студентських роботах. Аналіз предметної області. Постановка завдання. Вибір архітектури та програмних засобів. Програмна реалізація веб-системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада	Підпис, дата

	консультанта	завдання видав	завдання прийняв

7. Дата видачі завдання____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Оцінка виконання етапу	
			у %	за 4-х бальною шкалою
1	Дослідження предмета, цілей, особливостей реалізації			
	системи виявлення плагіату			
2	Аналіз існуючих систем. Визначення недоліків і переваг.			
	Визначення вимог до системи			
3	Основні принципи застосування фреймфорків при			
	створенні веб-систем			
4	Особливості архітектури подібних систем			
	Проміжкова атестація			
	Вибір архітектури і програмних засобів реалізації			
	системи пошуку плагіату			
	Проектування та розробка схеми БД			
	Розробка сервісу та методів доступу			
	Реалізація доступу до даних			
	Тестування системи			
	Оформлення пояснювальної записки. Створення			
	презентації			
	Інтегральна оцінка виконання етапів календарного плану (як середня по етапам)			

Студент _____ **Федючек О.В.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Терещенко Т.М.**
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення
SBT – Simple Build Tool – система збірки проектів для мови Scala

Терміни
Netty – NIO сервер.
Solr – система повнотекстового пошуку.
Scala Play – фрейморвк для написання Web-ситем на мові Scala.
Lucene – найвідоміший з пошукових движків.
Tika – бібліотека для роботи з різноманітними форматами документів.
Хостинг – послуга з надання простору для розміщення сайтів в Інтернеті.

ВСТУП

На сьогоднішній день одним з найбільш актуальних напрямків розвитку комп'ютерних технологій є розробка спеціалізованих інформаційних систем, використання яких є потужним засобом підвищення ефективності та продуктивності різних закладів.

Розвиток сучасних комп'ютерних технологій – розподілених систем, об'єктно-реляційних баз даних, Web-орієнтованих засобів рекламування товарів впливає на методологію і технологію організації рекламної компанії і передбачає реалізацію нового підходу до її інформаційного забезпечення.

З’являються нові проблеми для суспільства та можливості їх вирішення.

Одна із таких проблема це плагіат – привласнення авторства на чужий твір або на чуже відкриття, винахід чи раціоналізаторську пропозицію, а також використання у своїх працях чужого твору без посилання на автора . Також з розвитком технологій з’явилися нові можливості для пошуку плагіату. До візуального визначення плагіату шляхом порівняння текстів двох творів додались і технічні способи, які полягають в автоматичному порівнянні електронного тексту з базою даних інших текстів. Подібним програмним забезпеченням користується все більша кількість видавництв, юридичних фірм, Інтернет-компаній, державних органів. Такі системи дають змогу визначити справжнього власника тексту та запобігти подальшому плагіату.

Ціль цієї роботи – розробка системи виявлення плагіату.Для цього потрібно вирішити такі задачі:

– розробити архітектуру системи;
– обрати кращий алгоритм для реалізаціїї системи;
– реалізувати інтерфейс.

Об'єкт дослідження – це повноцінна система пошуку плагіату.

Предмет досліждення – процесс знаходження плагіату, алгоритми пошуку та визначення.

Результати отримані у цій роботі були докладені на конференції минулого року, та на студентській конференції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Дослідження предмета, цілей і особливостей систем виявлення плагіату в наукових та студентських роботах.

Настав той час, коли важко уявити життя без мобільних телефонів, швидкого інтернету, різноманітних засобів зв'язку. З розвитком технологій з'явилась проблема з плагіатом та його пошуком. Кожна людина має доступ до будь-якої інформації та здатна скопіювати її без жодних проблем. Саме для запобігання цьому з'явились системи анти плагіату.

Сотні алгоритмів, великі бази даних, системи на основі нейронних сітей – все це працює над виявленням та знаходженням плагіату

Система виявлення плагіату в наукових та студентських роботах – це Web-система, яка дозволяє перевірити та виявити плагіат у тексті, відсоток плагіату, або навіть знайти частини тексту зі зміненими словами або реченнями. Ці системи набули поширення за останні 20-30 років і на наш час використовуються практично усюди для перевірки оригінальності документі, текстів, наукових робіт.

В Україні існує багато таких систем. Вони відрізняються застосуванням різних алгоритмів, способів пошуку інформації, способами зберігання даних. Більшість таких систем це комерційні продукти з різними рівнями фінансування та закритими алгоритмами. Тому аналіз таких систем дещо ускладняється.

Головним плюсом цих систем є те, що частіше за все вони мають необхідну базу текстів яка налічує терабайти даних, за допомогою яких аналізуються вхідні тексті. Більшість використовує пошукові алгоритми Google або Яндекс, але це на так ефективно. Зазвичай аналіз та пошук дуже швидкий, якщо використовувати деякі алгоритми та внутрішню базу.

Такі системи дуже підходять для розширення. Можна безліч разів додавати потужні сервера для обробки і індексування даних та збільшувати пам'ять для сховищ даних.

Системи пошуку плагіату використовуються на державному рівні. Зокрема набули поширення у більшості університетів, при патентуванні. На даний час існує деякий застій у розвитку цих система. Майже усі працюють із старими алгоритмами з грубим порівняння або мішками слів та речень

1.2 Аналіз існуючих систем

Для визначення вимог, функцій, бізнес-логіки розроблюваної системи виявлення плагіату в наукових та студентських роботах проведений необхідний аналіз існуючих і функціонуючих в мережі Інтернет аналогічних систем. У ході аналізу були визначені основні переваги та недоліки таких систем.

Відмінності між такими системами визначаються за наявністю різноманітних додаткових способів пошуку, якості пошуку, наявності генерації звітів, ціни.

Розглянемо деякі спеціалізовані системи виявлення плагіату.

Quetext – безкоштовна система пошуку плагіату. Базується у США, та має велику кількість клієнтів від звичайних людей, до великих компаній та університетів. (рис. 1.1) [2]

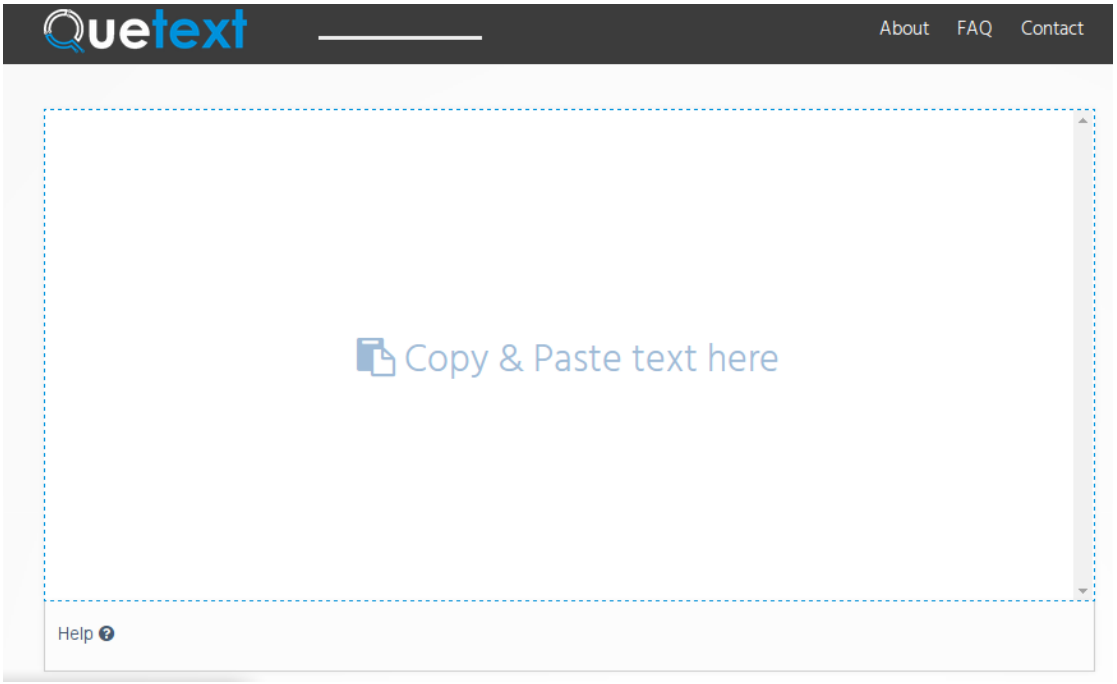

Рисунок 1.1 – Головна сторінка системи «Quetext»

Ця система має приємний дизайн сайту, великий вибір планів користування системою, різноманітні знижки. Можна сказати, що ця компанія має дуже великий досвід у використанні подібних систем. Також вони надають багато різноманітних пакетів послуг для юридичних та фізичних осіб.

Перевагою даної системи є:

– велика база даних текстів;
– доступ до тисяч організацій та їх сховищ даних;
– різноманіття планів та цін на послуги для різних людей.

Недоліками є:

– лише на англійській мові;
– є обмження на пошук;
– приватна система (приватні алгоритми пошуку).

Наступна розглянута система – це Text.ru. Дуже проста система пошуку плагіату. (рис. 1.2) [3]

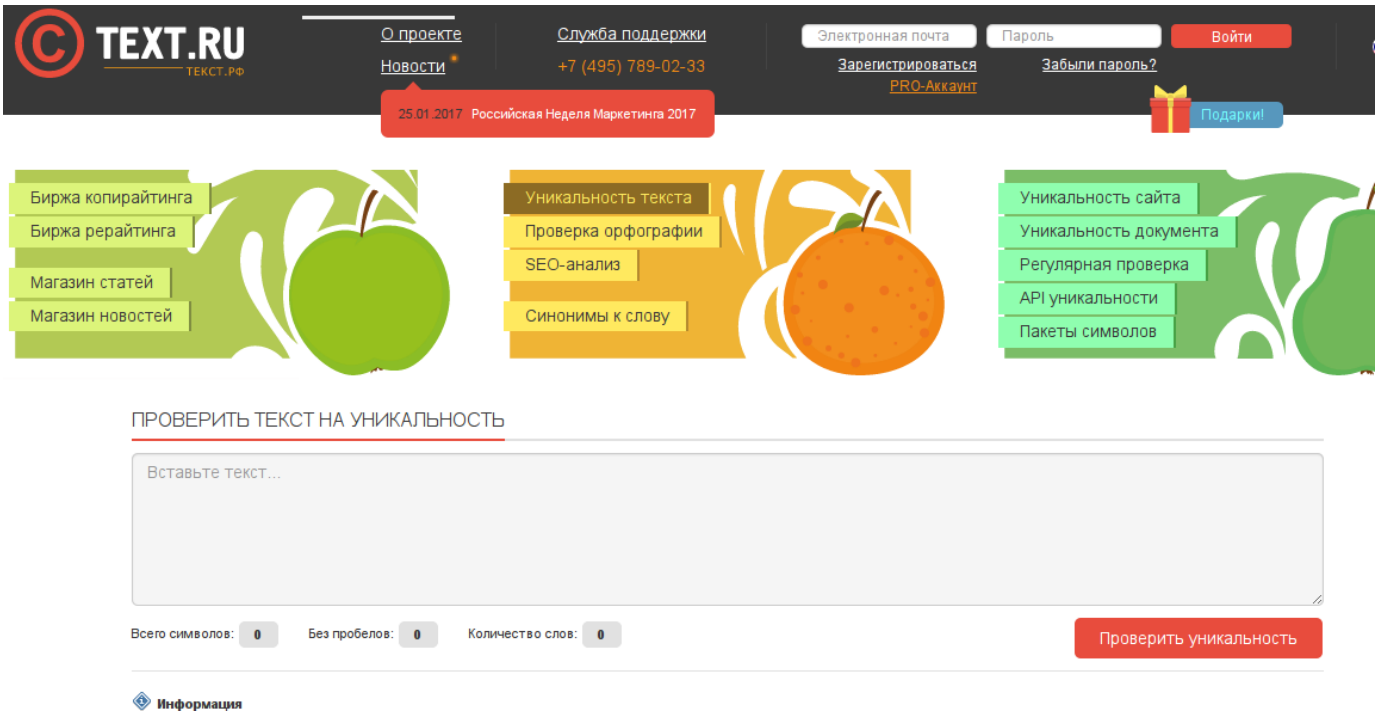

Рисунок 1.2 – Головна сторінка сайту системи «Text.ru»

Система позиціонується як просто та зручна для користувача. Відразу можна побачити поле вводу для тескту та перевірити його унікальність. Пошук здійснюється швидко, скоріше за все використовується зовнішня система пошуку.

Перевагою даної системи є:

– зручність у користуванні;
– декілька способів пошуку плагіату;
– доступність десятка мов для текстів;
– створення звітності за результатами пошуку;
– додаткові послуги для клієнта.

Недоліками є:

– приватність внутрішнього алгоритму пошуку;
– не дуже зручна звітність.

Наступна розглянута система – це PR.CY. (рис. 1.3) [4]

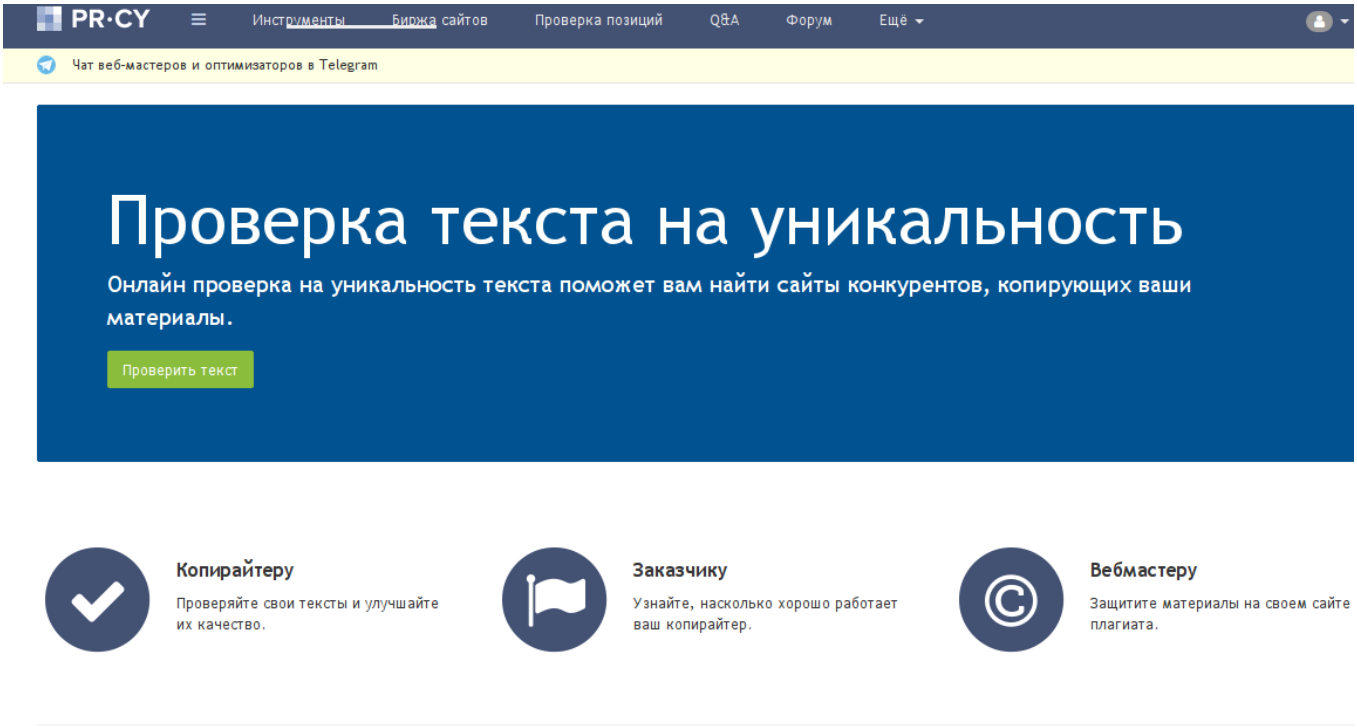

Рисунок 1.3 – Головна сторінка сайту системи «PR.CY»

Ця система має приємний дизайн сайту та трохи більший функціонал ніж інші системи.

Перевагою даної системи є:

– зручність у користуванні;

– простота системи;
– велика кількість форматів документі;
– швидкість роботи;
– можливість роботи без реєстрації.

Недоліками є:
– один спосіб пошуку плагіату;
– відсутність планів для фізичних або юридичних осіб.

Наступна розглянута система – це Plagiarisma. За функціоналом подібна до інших систем але також має суттєвий мінус.. (рис. 1.4) [5]

Рисунок 1.4 – Головна сторінка сайту системи «Plagiarisma»

При перевірці на унікальність використовуються зовнішні пошукові системи.

Перевагою даної системи є:

– є можливість подивитися, які частини тексту були знайдені на кожній з проаналізованих сторінок.

Недоліками є:

– Більша частина функціоналу стає доступна тільки після реєстраціїї.

У результаті проведеного аналізу існуючих систем, з множини реалізацій перевага віддається тій, яка найбільш проста у використанні та має найбільший функціонал по пошуку плагітау. Можна виділити головні якості, які необхідні для цієї системи: точність, простота, легкість у впровадженні.

1.3 Вимоги до програмних продуктів, призначених для впровадження системи

Ідеальним варіантом для реалізації системи виявлення плагіату в наукових та студентських роботах є мережа Інтернет. Через яку можна найбільш простіше та організувати пошук та обробку текстів. Така система буде мати оптимальне співвідношення чинників: ціна-якість.

Що потрібно для створення такої системи і її поширення:

– точність аналізу та пошуку плагіата – найважливіший компонент цієї системи. Від нього залежить ефективність цієї системи;
– простота інтерфейсів – теж дуже важливий компонент. Чим простіше буде керуватися система, тим легше буде її впровадження;
– швидкість – система повинна працювати швидко і без затримок. Швидко індексувати нові тексти та робити пошук по базі;
– конфіденційність – забезпечувати безпеку збереження даних та конфіденційність.

Необхідні функціональні можливості системи:

– формування власної бази внутрішніх джерел (роботи учнів минулих років, підручники, художня література, методичні матеріали та інші тексти), як загальної для всіх факультетів, так і окремих для розширення області пошуку плагіату;
– побудова звітів про перевірку із зазначенням тих фрагментів аналізованого тексту, які були знайдені системою в базі джерел, з можливістю перегляду тексту цих джерел;

– видача короткого звіту, що містить список джерел та відсотка оригінальності документа, а також повного звіту про перевірку, плюс, можливість вибору найбільш підходящого для користувача з чотирьох варіантів його відображення.

1.4 Огляд існуючих рішень пошуку плагіату

Існує ряд методів для знаходження плагіату в багатомовних корпусах текстів.

Найбільш часто процес пошуку здійснюється за наступним алгоритмом:

– Евристичний пошук. З корпусу відбираються документи, в яких містяться фрагменти тексту, схожі на деякі частини тексту T0, що перевіряється на плагіат. На цьому кроці використовуються різні алгоритми для виявлення схожих текстів, в тому числі часто необхідно використовувати деякий перехід між мовами документів–переклад ключових слів, визначення жанру і теми.
– Докладний аналіз. Для кожного з відібраних документів аналізується ступінь подібності з перевіряється, якщо вона досить висока – передбачається випадок плагіату. У багатьох системах цей крок застосовується без евристичного пошуку: порівняння йде з кожним з текстів корпусу.
– Пост-обробка. Отримані результати перевіряються, найчастіше вручну, для запобігання хибним виявлень (наприклад, випадків, коли за плагіат приймається оформлена за всіма правилами цитація).

Розглянемо найбільш часто використовувані прийоми відбору текстів з корпусу:

– У тексті T0 виділяються ключові слова, які потім переводяться на мови кожного з розглянутих документів корпусу, проводиться пошук текстів по цими ключовими словами.
– Текст T0 перекладається на мову документа, з яким порівнюється, за допомогою машинного перекладу, після чого з нього виділяються ключові слова, відбувається пошук отриманих слів в документі. Ці два підходи можна порівняти за ефективністю.
– Текст представляється як невеликий набір чисел, званий відбитком. Ці числа обчислюються за допомогою hash-функції, що реалізує функцію

міри близькості і з високою ймовірністю находячи схожі документи однаковому hash-коду.

При використанні даного методу hash-код будується, як правило, з машинного перекладу тексту T0 на мову документа, з яким йде порівняння, проте деякі дослідження пропонують хеш-функції, за допомогою яких можна отримати однакові результати для схожих текстів на різних мовах.

Досить часто для пошуку плагіату в багатомовних корпусах використовується поєднання машинного перекладу і алгоритмів для текстів на одній мові. Розглянемо докладніше найбільш популярний з них.

Одним з найбільш часто використовуваних є метод «шинглів», запропонований в 1997 році. Він заснований на представленні документа у вигляді всіляких послідовностей фіксованої довжини k, що складаються з сусідніх слів. Такі послідовності назвали Шингли» (від англ. Shingles). Два документа вважаються схожими, якщо безлічі їх 8 шинглів істотно перетинаються. Оскільки число шинглів для кожного документа є досить великим, використовуються різні способи усічення їх безлічі, засновані на обчисленні їх дактілограмм. Дактілограмми документа включає всі
текстові підрядка фіксованої довжини, а її чисельне значення за допомогою
алгоритму випадкових поліномів Карпа-Рабіна.

У сучасній інтерпретації кожен документ представляється 84 шинглі, мінімізують статистичні функції, використані для обчислення дактілограмм. Ці шингли розбиваються на 6 груп по 14 шинглів в кожної–супершінглів. Якщо два документи мають схожість з ймовірністю p, то два їх супершінглів збігаються з ймовірністю p14. Для ефективної перевірки гіпотези про схожості кожний документ видається всілякими попарними поєднаннями з 6 супершінглів–мегашінглів (їх всього 15). Два документа вважаються схожими по змісту, якщо у них збігається хоча б один мегашінглів.

Методи на основі шинглів використовуються в основі багатьох алгоритмів визначення плагіату в одномовних корпусах, проте для випадку багатьох мов їх застосування недоцільно, тому що зміна порядку слів при перекладі не дозволяє абстрагуватися від мови, якою написаний текст.

Ще одне з цікавих рішень запропоновано Bruno Pouliquen. На першому етапі, кожному дескриптору з тезауруса EUROVOC (http://eurovoc.europa.eu/) ставиться у відповідність статистично визначається набір слів, що відносяться до нього. Далі для кожного з розглянутих текстів визначається деякий набір

відповідних йому дескрипторів з EUROVOC і ці набори порівнюються. Схожим текстів будуть відповідати схожі набори дескрипторів. Даний метод цікавий тим, що не вимагає машинного перекладу ні для перевіряється документа, ні для документів корпусу, крім того, набори дескрипторів для документів корпусу досить обчислити один раз. Однак використання такого алгоритму обмежується набором мов, для яких доступний тезаурус.

Розглянемо інший сигнатурний підхід, заснований вже не на синтаксичних, а на лексичних принципах. Основна ідея такого підходу полягає в обчисленні дактілограмми I-Match для подання змісту документів. З цією метою спочатку для вихідної колекції документів будується словник L, який включає слова з середніми значеннями IDF, оскільки такі слова забезпечують, як правило, більш точні результати при виявленні нечітких дублікатів. Слова з великими і маленькими значеннями IDF відкидаються. Потім для кожного документа формується безліч U різних слів, що входять в нього, і визначається перетин U і словника L. Якщо розмір цього перетину більше деякого мінімального порогу 9 (визначається експериментально), то список слів, які входять в перетин впорядковується, і для нього обчислюється I -Match сигнатура (hash-функція SHA1). Два документа вважаються схожими, якщо у них збігаються I-Match сигнатури (має місце колізія hash-кодів). Алгоритм має високу обчислювальну ефективність, що перевершує показники алгоритму шинглів. Іншою перевагою алгорітмаявляется його висока ефективність при порівнянні невеликих за розміром документів. Основний недолік–нестійкість до невеликих змін змісту документа. Для подолання зазначеного недоліку вихідний алгоритм був модифікований, і в нього була введена можливість багаторазового випадкового перемішування основного словника. Додатково до основного словника L створюються K різних словників L1-LK, одержуваних шляхом випадкового видалення з вихідного словника деякого невеликої фіксованої частини p слів, що становить близько 30% -35% від початкового об'єму L. Для кожного документа замість однієї обчислюється (K + 1) I-Match сигнатура за алгоритмом, описаним вище, тобто документ представляється у вигляді вектора розмірності (K + 1), і два документа вважаються дублікатами, якщо у них збігається хоча б одна з координат. Якщо документ піддається невеликим змінам (порядку n слів), то ймовірність того, що принаймні одна з K додаткових сигнатур залишиться незмінною дорівнює 0.

Як сигнатури документа використовується хеш-код рядка, отриманий зчепленням шести найбільш часто зустрічаємих у документі слів. Даний метод

можна вважати одним з випадків використання аналізу ключових слів текстів. Незважаючи на свою простоту, при використанні нормалізації і стоп-слів метод дає непогані результати.

Алгоритм MinHash розроблений як продовження алгоритму Shingles для зменшення часу витрачається на порівняння образів різних текстів і зменшення кількості займаної пам'яті. Після обробки тексту алгоритмом шінгліровання алгоритм MinHash здійснює обчислення хеш-функцій і відбір їх мінімальних значень. Алгоритм MinHash використовує h хеш-функцій. Кожній хеш-функції ставить у відповідність число–найменше її значення для всіх шинглів досліджуваної рядки. Отриманий в результаті вектор довжини h утворює сигнатуру або короткий числовий образ документа. На вхід алгоритму MinHash подається масив шинглів довжини N. На виході алгоритм видає одновимірний масив (вектор) хеш-кодів довжини h, т. Е. Сигнатуру або короткий числовий образ документа. Опис алгоритму MinHash наведено нижче (рис. 1.5)

Алгоритм MinHash

Вход: *shingles* – массив шинглов длины N

Выход: *hashes* – массив хеш-кодов длины h

Пусть $F \leftarrow$ массив хеш-функций алгоритма длины h

Для j **с** 1 **до** h

    $hashes[j] \leftarrow$ макс. значение

Для i **с** 1 **до** N

    Для j **с** 1 **до** h

        $hashes[j] \leftarrow \min\{hashes[j], F_j(shingles[i])\}$

Вывод *hashes*

Рисунок 1.5 – Алгоритм MinHash

Алгоритм MinHash використовує однаковий набір хеш-функцій в заданому порядку для обробки всієї множини текстів, тому список хеш-функцій зазвичай є частиною реалізації алгоритму. Порівняння документів здійснюється за значеннями векторів hashes1 і hashes2, побудованих алгоритмом MinHash з

використанням одного і того ж складу хеш-функцій. Побудовані вектори hashes1 і hashes2 завжди мають однакову довжину, рівну h. При виявленні дублікатів в алгоритм MinHash вбудовано 60 хеш-функцій.

Цей алгоритм відрізняється від інших своєю неймовірною швидкістю і здатен набагато пришвидшити пошук імовірних дублікатів. Приклад роботи цього алгоритму (рис. 1.6)

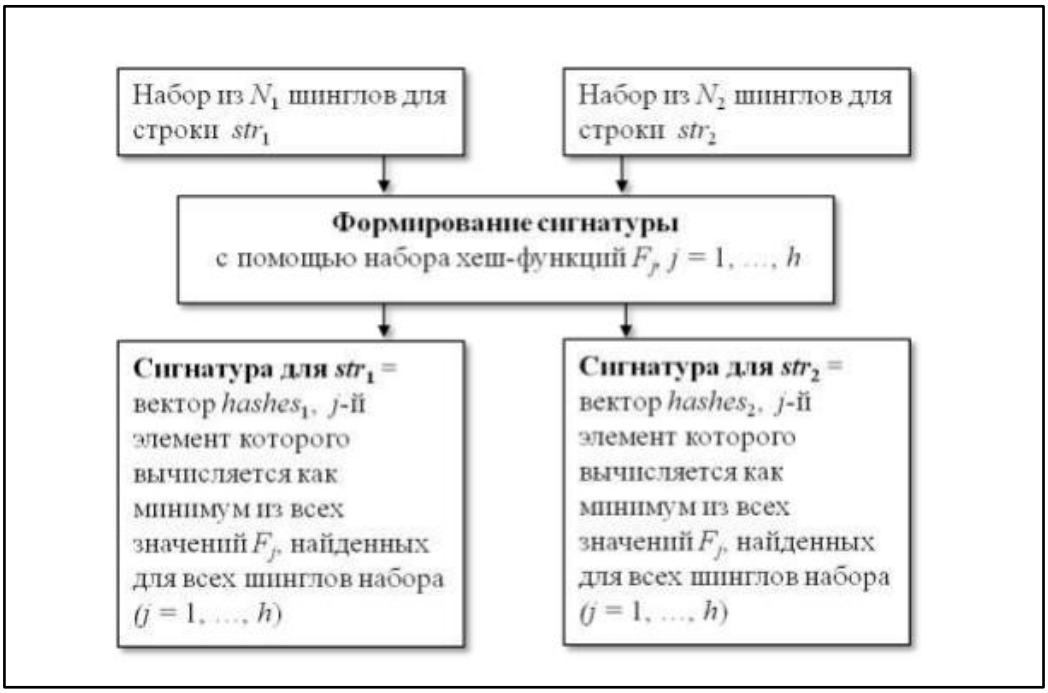

Рисунок 1.6 – Алгоритм MinHash. Signature

После применения алгоритма

У ході дипломного проектування розробити систему виявлення плагіату в наукових та студентських роботах.

Саму систему необхідно реалізувати у вигляді сервісу, який повинен надавати зовнішнє API користувачу. Потрібно реалізувати такі можливісті: пошуку плагіату, індексування та завантаження тексту, вигрузку тексту з локального сховища, редагування текстів, збереження форматування текстів.

Також система повинна мати можливість масштабуватись, бути гнучкою та модульною. Для цього потрібно обрати найоптимальніше архітектурне рішення

2 ВИБІР І ОБГРУНТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Загальна архітектура системи

Сучасні програмні додатки та інформаційні системи досягли високого рівня розвитку і термін або поняття «архітектура» у застосуванні до них дозволяє грамотно побудувати і сконструювати інформаційну систему в цілому, забезпечуючи її ефективне і надійне функціонування.

Архітектура інформаційної системи – концепція, визначальна модель, структуру, виконувані функції і взаємозв'язок компонентів інформаційної системи.

У міру розвитку програмних систем все більшого значення набуває їх інтеграція один з одним з метою побудови єдиного інформаційного простору підприємства. Для того щоб побудувати правильну і надійну архітектуру і грамотно спроектувати інтеграцію програмних систем необхідно чітко слідувати сучасним стандартам в цих областях. Без цього велика ймовірність, створити архітектуру, яка нездатна розвиватися і задовольняти зростаючим потребам користувачів IT.

Класифікація програмних систем за їх архітектурою представляється таким чином:

- централізована архітектура;
- архітектура «файл-сервер»;
- дворівнева архітектура «клієнт-сервер»;
- багаторівнева архітектура «клієнт-сервер»;
- архітектура розподілених систем;
- архітектура Web-додатків;
- сервіс-орієнтована архітектура.

Дана система виявлення плагіату в наукових та студентських роботах розроблена як сервіс-орієнтована система. Сервіс-орієнтована архітектура – архітектурний шаблон програмного забезпечення, модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабко пов'язаних замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколам. Дуже часто становлення того чи іншого підходу супроводжується появою невірних або

хибних трактувань, як це було, наприклад, з концепцією федеративного сховища даних. Не оминуло стороною це і сервіс-орієнтовану архітектуру. Так вважає представник компанії BEA Джерімі Уестерман (Jeremy Westerman). Саме тому в одній із своїх статей, присвячених SOA, він спеціально зупиняється на «проблемних місцях» і наводить докладні пояснення:

- SOA не є чимось новим: IT-відділи компаній успішно створювали і розгортали застосунки, що підтримують сервіс-орієнтовану архітектуру, вже багато років – задовго до появи XML і Web-сервісів;
- SOA – це не технологія, а спосіб проектування і організації інформаційної архітектури та бізнес-функціональності;
- купівля найновіших продуктів, що реалізують XML і Web-сервіси, не означає побудову застосунків згідно з принципами SOA.

Джерімі Уестерман дає наступне визначення SOA: це парадигма, призначена для проектування, розробки та управління дискретних одиниць логіки (сервісів) в обчислювальному середовищі. Застосування цього підходу вимагає від розробників проектування застосунків як набору сервісів, навіть якщо переваги такого рішення відразу неочевидні. Розробники повинні вийти за межі своїх застосунків і подумати, як скористатися вже наявними сервісами, або вивчити, як їх сервіси можуть бути використані їх колегами.

SOA підштовхує до використання альтернативних технологій і підходів (таких як обмін повідомленнями) для побудови застосунків за допомогою зв'язування сервісів, а не за допомогою написання нового програмного коду. У цьому випадку, при належному проектуванні, застосування повідомлень дозволяє компаніям своєчасно реагувати на зміну ринкових умов – настроювати процес обміну повідомленнями, а не розробляти нові програми.

Ще до недавніх пір термін «сервіс-орієнтована архітектура» був синонімом «Web-сервіс». SOA – виклик Web-сервісів за допомогою засобів і мов управління бізнес-процесами. SOA – це термін, який з'явився для опису виконуваних компонентів, таких як Web-сервіси, які можуть викликатися іншими програмами, які виступають клієнтами або споживачами цих сервісів. Ці сервіси можуть бути повністю сучасними – або навіть застарілими – прикладними програмами, які можна активізувати як чорний ящик. Від розробника не потрібно знати, як працює програма, необхідно лише розуміти, які вхідні та вихідні дані потрібні, і як викликати ці програми для виконання. У найзагальнішому вигляді SOA припускає наявність трьох основних учасників:

постачальника сервісу, споживача сервісу та реєстру сервісів. Взаємодія учасників виглядає досить просто: постачальник сервісу реєструє свої сервіси в реєстрі, а споживач звертається до реєстру із запитом.

Для використання сервісу необхідно дотримуватися угоди про інтерфейс для звернення до сервісу – інтерфейс повинен не залежати від платформи. SOA реалізує масштабованість сервісів – можливість додавання сервісів, а також їх модернізацію. Постачальник сервісу і його споживач виявляються непов'язаними – вони спілкуються за допомогою повідомлень. Оскільки інтерфейс повинен не залежати від платформи, то і технологія, використовувана для визначення повідомлень, також повинна не залежати від платформи. Тому, як правило, повідомлення є XML-документами, які відповідають XML-схемі.

Дійсно, відкриті стандарти, що описують XML і Web-сервіси, дозволяють застосовувати SOA до всіх технологій і застосунків, встановлених в компанії. Як відомо, Web-сервіси базуються на широко поширених і відкритих протоколах: HTTP, XML, UDDI, WSDL і SOAP. Саме ці стандарти реалізують основні вимоги SOA – по-перше, сервіс повинен піддаватися динамічному виявленню і викликом (UDDI, WSDL і SOAP), подруге, повинен використовуватися незалежний від платформи інтерфейс (XML). Нарешті, HTTP забезпечує функціональну сумісність.

Сервіс-орієнтована архітектура заснована на наступних базових принципах, що дозволяють зняти найбільш гострі проблеми інтеграції додатків.

Слабка зв'язаність. З точки зору реалізації сервіси в SOA незалежні один від одного: вони виконують певні дії за запитами, отриманими від інших сервісів, і повертають результати. Всі деталі цього виконання повністю приховані: в концепції SOA сервіси – це "чорні ящики". Звідси слідує що зміни в реалізації сервісу ніяк не вплинуть на прикладної компонент, який цей сервіс використовує, і навпаки. Слабка зв'язаність забезпечує просту і швидку адаптацію системи до змін в структурі і принципах реалізації сервісів.

"Грубозерниста" структура сервісів. Сервіси в SOA є модулі бізнес-логіки досить високого рівня, завдяки чому взаємодія між ними зводиться до обмеженого числа повідомлень за змістом бізнес-логіки замість безлічі низькорівневих викликів, які враховують деталі реалізації сервісів. Такий підхід знижує навантаження на мережу і сприяє більш високій продуктивності системи. На практиці перехід до SOA може починатися з подання до вигляді сервісів бізнес- або системних функцій низького рівня, які потім завдяки ще одній властивості сервісів

– модульности будуть компонуватись в високорівневі сервіси, які реалізують певні етапи бізнес або обчислювального процесу або весь процес цілком. Важливо, що з точки зору архітектури сервіс, незалежно від внутрішньої структури і мови реалізації, виглядає як єдине ціле. Сервіс-орієнтована архітектура заснована на наступних базових принципах, що дозволяють зняти найбільш гострі проблеми інтеграції додатків.

Реалізація слабкою зв'язності може ґрунтуватися на поєднанні різних засобів і методів:

1. Контракти. Поняття контракту в SOA трактується дещо ширше, ніж зазвичай. Контракт визначає умови надання і споживання сервісів, включаючи функціональні, технічні (не тільки протокол, а й, наприклад, спосіб авторизації) та інші (вартість, гарантії) параметри і служить основою угоди між споживачем і провайдером сервісу на всіх етапах його життєвого циклу. Чи не даром одне з поширених визначень провайдера і споживача сервісів базується на тому, хто з них контракт надає (провайдер), а хто приймає (Споживач).

Як і в будь-якому іншому архітектурному стилі, контракт дозволяє розділити компонент на дві частини – публічну, більш стабільну, і приховану, потенційно мінливу. Оскільки спосіб організації компонентів в SOA відповідає способу організації бізнесу або обчислювального процесу, контракт окреслює можливості не тільки програмного компонента, скільки бізнес або обчислювальної функціональності, розділяючи сервіс на бізнес інтерфейс і бізнес (обчислювальний) процес. При цьому процес може змінюватися в рамках даного контракту, оскільки для споживачі сервісу взаємодіють з тільки з інтерфейсом.

Контракти не тільки послаблюють зв'язки між елементами архітектури, але, що не менш важливо, роблять їх явними, упредметненими об'єктами управління. Постійна зміна різних ділянок SOA, таким чином, стає можливо саме завдяки контрактам.

2. Композиція. Основною перевагою композиції, цілком самостійної риси SOA, вважається забезпечення можливості багаторазового використання раніше реалізованої функціональності. Однак композиція сприяє і адаптивності, тому що дозволяє швидше проводити багато змін шляхом перестиковки зв'язків між сервісами. Такі зміни здійснювати набагато легше, ніж міняти весь сервіс в цілому.

3. Динамічне зв'язування. Спираючись на стабільність контракту споживач може знаходити необхідні екземпляри сервісів, не перериваючи роботи. Це означає, що система стійка до змін не тільки в адресації доступних сервісів (Переїзд на інший сервер), а й в їх складі, який може змінитися за час експлуатації. Наприклад, можуть з'явитися нові партнери, або відповідальність за надання необхідної бізнес-функції може перейти до іншого провайдера, в тому числі зовнішнього, проте це не вплине на роботу споживача.

Динамічне зв'язування підтримується реєстром, який виконує функції облікової системи SOA. Зазвичай він містить посилання на контракти, конкретні екземпляри сервісів, інформацію про їх провайдерів і іноді також про їх споживачах. Все це разом, крім інших не менш важливих функцій, служить також і конфігурацією SOA, що визначає параметри часу виконання використовуваних сервісів для споживачів сервісів (внаслідок композиції часто є також їх провайдерами).

4. Масштабованість інтерфейсів. При всій тіпізірованія інтерфейсів, хорошою практикою вважається залишати можливість для додавання заздалегідь непередбачених елементів даних в передані повідомлення. Таким чином, контракт заздалегідь допускає деяку гнучкість, якщо вона можлива в конкретної ситуації.

Практичні аспекти сервісно-орієнтованої технології дозволяють вирішити проблеми масштабованості, інтегрувати мережі передачі даних і голосу, спростити процедури проектування і управління мережами, а також створити інші розподілені додатки, прозоро взаємодіють з ресурсами систем за допомогою прикладних програмних інтерфейсів і відкритих стандартів.

Систем виявлення плагіату в наукових та студентських роботах представляє собою трирівневу архітектуру (three-tier), що припускає наявність наступ-них компонентів програми: клієнтський додаток («тонкий клієнт» або термінал), підключений до сервера додатків, який в свою чергу підключений до сервера бази даних (рис. 2.1).

Рис. 2.1–Багаторівнева архітектура «клієнт-сервер»

Термінал – це інтерфейсний, зазвичай графічний, компонент, який представляє перший рівень, власне додаток для кінцевого користувача. Перший рівень не повинен мати прямих зв'язків з базою даних (за вимогами безпеки), бути навантаженим основною бізнес-логікою (за вимогами масштабованості) і зберігати стан додатків (за вимогами надійності). На перший рівень може бути винесена і зазвичай виноситься найпростіша бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка введених значень на допустимість і відповідність формату, нескладні операції (сортування, угруповання, підрахунок значень) з даними, вже завантаженими на термінал.

Сервер додатків розташовується на другому рівні. На другому рівні зосереджена велика частина бізнес-логіки. Поза його залишаються фрагменти, що експортуються на термінали, а також занурені в третій рівень збережені процедури і тригери.

Сервер бази даних забезпечує зберігання даних і виноситься на третій рівень. Зазвичай це стандартна реляційна або об'єктно-орієнтована СУБД. Якщо третій рівень являє собою базу даних разом з збереженими процедурами, тригерами і схемою, яка описує додаток в термінах реляційної моделі, то другий

рівень будується програмний інтерфейс, що зв'язує клієнтські компоненти з прикладної логікою бази даних.

У простій конфігурації фізично сервер додатків може бути поєднаний з сервером бази даних на одному комп'ютері, до якого по мережі підключається один або декілька терміналів.

2.2 Вибір сервера додатків

Було розглянуто декілька серверів додатків.

GlassFish – сервер застосунків з відкритим сирцевим кодом, який реалізує специфікації Java EE. Спочатку розроблений Sun Microsystems, після поглинання цієї компанії спонсорується корпорацією Oracle. Актуальна версія платформи називається Oracle GlassFish Server (або GlassFish Server Open Source Edition) Код GlassFish поширюється під двома ліцензіями: CDDL v1.0 і GPL v2.

Порівняно з простим використанням Apache Tomcat, використання цього серверу непотрібне, через його надлишкову функціональність.

JBoss Application Server або JBoss AS скорочено JBoss – сервер застосунків на платформі Java EE з відкритим сирцевим кодом, розроблений однойменною компанією JBoss. Як і багато програм з відкритим кодом, розроблених комерційними організаціями, JBoss можна вільно завантажувати та використовувати, проте підтримка та консультації здійснюються за окрему платню. Достатньо якісна реалізація принципів J2EE робить JBoss конкурентом для аналогічних власницьких програмних рішень, таких як WebSphere або WebLogic.

JBoss використовує Apache Tomcat в якості контейнеру сервлетів.

У випуску JBoss AS 6 підтримує Java Enterprise Edition 6; в JBoss AS 7 реалізовано підтримку додаткового профілю Java EE 6 "Web Profile", що являє собою легковагову і переносну підмножину Java EE 6, створену для розробки і поширення інтерактивних веб-застосунків. У JBoss AS 7 архітектура проекту перероблена з метою відходу від монолітної до модульної структури, що є важливим кроком у напрямку підтримки хмарних і мобільних технологій. Ключовим досягненням версії 7.1 є отримання сертифікату повної сумісності з Java EE 6.0 "Full Profile" на додаток до раніше досягнутої відповідності профілю Java EE 6 "Web Profile".

Причини, через які він не був обраний тіж самі, що і вище.

Apache HTTP-сервер – відкритий веб-сервер Інтернет для UNIX-подібних, Microsoft Windows, Novell NetWare та інших операційних систем.

Apache передусім використовується для передачі через HTTP статичних та динамічних веб-сторінок у всесвітній павутині. Багато веб-застосунків спроектовано, зважаючи на середовище і можливості, які надає цей веб-сервер.
Продукт може працювати в якості кешувального проксі-сервера, що дозволяє істотно підвищити продуктивність роботи користувачів локальної мережі при роботі з документами, розташованими в Інтернет. Можна задавати такі параметри і налаштування проксі-сервера

Apache зіграв ключову роль у початковому зростанні всесвітньої павутини, і продовжує бути найпопулярнішим у світі веб-сервером, де-факто платформою, на яку орієнтуються інші веб-сервери.

Відповідно до статистики Netcaft за червень 2008 року, Apache є найпоширенішим серверним програмним забезпеченням в Мережі: на цей веб-сервер припадала частка близько 49 % відповідного сегменту ринку (майже 85 мільйонів сайтів). Друге місце за популярністю займають програмні платформи Microsoft – 35,4 % (61 мільйон сайтів).

Netty – NIO server framework

С допомогою Netty можна швидко і просто написати будь-який клієнт-серверний додаток, яке буде легко розширюватися і масштабироваться. Якщо для обробки клієнтів не вистачає одного потоку, слід всього лише передати необхідну кількість потоків в конструктор EventLoopGroup. Якщо на якійсь стадії розвитку проекту знадобиться додаткова обробка даних, не потрібно переписувати код, досить додати новий обробник в ChannelPipeline, що значно спрощує підтримку програми.

Netty дозволяє передавати дані через TCP або UDP, підтримує безліч протоколів, таких як HTTP, FTP, SMTP, WebSockets, SSL / TLC, SPDY. API добре документований, на гітхабе викладено безліч прикладів з докладними коментарями. А все зростаюче співтовариство свідчить про те, що продукт вийшов хороший, гідний уваги і активного використання.

Цей сервер був обраний для розробки цієї системи через свою легкість та швидкість.

2.3 Вибір системи управління базами даних

Ця система потребує використання СУБД. Головним чином для збереження текстів та їх індексування. Використовується нереляційна модель даних.

NoSQL (зазвичай розшифровується як "non SQL", "non relational" or "not only SQL"–англ. не тільки SQL) – база даних, що забезпечує інший механізм зберігання та видобування даних, ніж звичний підхід таблиць-відношень в реляційних базах даних. Подібні бази даних існували вже в другій половині 1960тих, але тоді вони ще не здобули гучне ім'я "NoSQL" , одержане після сплеску популярності на початку 21-ого століття, що був спричинений потребами Web 2.0 компаніями, такми як Facebook, Google, та Amazon.com. NoSQL бази даних все більше і більше використовуються в задачах із застосуванням big data та real-time web застосунках. NoSQL системи також називають "Not only SQL" (англ. *not only SQL* – не тільки SQL) для підкреслення того, що вони можуть підтримувати SQL-подібну структуру та мову запитів.

Мотиви цього підходу включають: простоту дизайну схеми БД, значно спрощене горизонтальне масштабування на кластери машин (що є проблемою для реляційних баз даних), і тонкий контроль над доступністю. Структури даних, що використовуються в NoSQL (до прикладу ключ-значення, граф, документ) є відмінними від тих, що використовуються за замовчуванням в реляційних базах, що робить тим самим деякі операції над даними значно швидшими на NoSQL. Точна відповідність використання NoSQL бази даних залежить від проблем, які треба вирішити. Іноді структури даних, використовувані в NoSQL базах можуть розглядатись як більш гнучкі ніж таблиці реляційних моделей.

Багато NoSQL сховищ нехтують узгодженістю даних на противагу доступності, толерантності до партиціонування, та звісно швидкості. Бар'єрами прийняття парадигми NoSQL сховищ є використання низькорівневої мови запитів (замість добре розвиненого та стандартизованого SQL), брак стандартизованих інтерфейсів і значні інвестиції в існуючі реляційні бази. Більшість NoSQL сховищ далеко не забезпечують добре знані ACID транзакції, однак декілька баз, такі як MarkLogic, Aerospike, FairCom c-treeACE,

Google Spanner (що технічно теж NewSQL база даних), Symas LMDB, та OrientDB зробили на цьому додатковий акцент.

Натомість більшість NoSQL баз даних пропонують концепцію випадкового узгодження даних, в якому зміни в базі продубльованно на всі вузли "випадковим чином" (зазвичай така дія займає мілісекунди), що запити даних можуть не повернути оновлені дані моментально, або ж прочитані дані будуть не точними–давно знана проблема читання станів. На додаток, деякі NoSQL системи можуть містити дротову або іншої форми втрату даних. На щастя, деякі NoSQL забезпечують принцип write-ahead logging для уникнення втрати даних. Для розподіленої обробки транзакцій, поверх множинних баз даних, узгодженість даних, як наслідок, навіть більше завдання ніж воно постає при реляційному підході. навіть поточні реляційні бази не гарантують цілісність посилань для розширених баз даних. Це всього декілька систем що підтримують як і ACID транзакції так і X/Open XA стандарти для підтримки обробки транзакцій на розподілених базах даних.

Типи та приклади NoSQL баз даних

Відомо декілька способів класифікації NoSQL баз даних, кожен зі своїм набором категорій, деякі з них частково збігаються. Розглянемо базову класифікацію за моделями даних:

- Колонка: Accumulo, Cassandra, Druid, HBase, Vertica.
- Документ: Apache CouchDB, Clusterpoint, Couchbase, DocumentDB, HyperDex, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB.
- Ключ-значення: Aerospike, Couchbase, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, Berkeley DB.
- Граф: AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso, Stardog.
- Мульти-модель: Alchemy Database, ArangoDB, CortexDB, Couchbase, FoundationDB, MarkLogic, OrientDB.

Ключ-значення

Ключ-значення (від англ. Key-value (KV)) сховище використовує асоціативний масив (знаний як карта або словник) я основну модель даних. В цій моделі дані представляються як колекція з пар типу ключ-значення, при тому що кожен можливий ключ з'являється в колекціях не більше одного разу.

Модель такого типу є однією із найпростіших, і багатші моделі зазвичай реалізовані як їх розширення. Модель типу ключ-значення може бути розширена до скінченно впорядкованої, що підтримує ключі в лексикографічному порядку. Це розширення є обчислювально потужним, в тому, що він може ефективно видобувати селективні діапазони ключів.

Моделі типу ключ-значення можуть використовувати різні рівні узгодженості, починаючи від випадкової і завершуючи серіалізованими даними. Деякі бази даних підтримують впорядковані ключі. Існують різні реалізації апаратного забезпечення, і деякі користувачі підтримують дані в пам'яті (RAM), в той час як інші використовують HDD накопичувачі або класичні обертові диски.

Приклад: Oracle NoSQL Database, Redis, and dbm.

Документні сховища

Центральним поняттям такої моделі даних є "документ". У той час кожна документно-орієнтована реалізація бази даних відрізняється від деталей цього визначення, в загальному, всі вони припускають, що документи і інкапсулюють та шифрують збережені дані в деяких стандартних форматах або кодуваннях. Кодування на практиці включає XML, YAML і JSON, а також бінарні форми, як BSON. Документи адресуються в базі даних за допомогою унікального ключа.

Різноманітні способи реалізації пропонують різноманітні підходи і (або) способи групування документів:

- колекції;
- теги;
- невидимі мета-дані;
- ієрархія директорій.

У порівнянні з реляційними, колекції, до прикладу, можна розглядати як аналоги таблиць а документи–аналоги до записів. Однак існує відмінність: кожен запис в таблиці має сталу послідовність атрибутів, в той час як документні бази можуть містити в колекції набори з абсолютно відмінними атрибутами

Для даної системи використовується система повнотекстового пошуку Apache Solr на основі Lucene. Ця система здатна зберігати свої дані фізично у будь-якій СУБД. Також можливе зберігання у своїй реалізації (nosql подібна). Її ми і використали для нашої системи.

2.4 Вибір мови програмування та допоміжних засобів

Scala – мультипарадигмова мова програмування, що поєднує властивості об'єктно-орієнтованого та функційного програмування. Назва Scala утворена зі слів "scalable" (маштабовна) та "language" (мова), для того щоб задекларувати, що мова може рости разом з вимогами користувачів.

Програми мовою Scala виконуються на віртуальній машині Java за умови приєднання до дистрибутиву файлу scala-library.jar. Scala сумісна із існуючими програмами мовою Java, тобто код Scala може викликатися із Java-програм і навпаки.

Scala-програми багато в чому схожі на Java-програми, і можуть вільно взаємодіяти з Java-кодом. Мова включає одноманітну об'єктну модель–в тому сенсі, що будь-яке значення є об'єктом, а будь-яка операція–викликом методу. При цьому є також функціональним мовою в тому сенсі, що функції–це повноправні значення.

У Scala включені потужні і одноманітні концепції абстракцій як для типів, так і для значень. Зокрема, мова містить гнучкі симетричні конструкції домішок для композиції класів і типажів. Можливо дозволяє виробляти декомпозицію об'єктів шляхом порівняння зі зразком; зразки і вирази при цьому були узагальнені для підтримки природної обробки XML-документів. В цілому, ці конструкції дозволяють легко висловлювати самостійні компоненти, які використовують бібліотеки Scala, не користуючись спеціальними мовними конструкціями.

Мова допускає зовнішні розширення компонентів з використанням уявлень (views). Можливості узагальненого програмування реалізуються за рахунок підтримки узагальнених функцій (generics), в тому числі вищого типажу (generics of a higher kind). Крім різних класичних структурних типів даних, в мову включена підтримка екзистенціальних типів.

2.4.1 Додаткові засоби реалізації

Apache Tika – це рішення з відкритим вихідним кодом, що використовується для читання електронних документів різних типів, включаючи документи Word, файли PDF і файли в розширеному текстовому форматі, а також для вилучення з них тексту та інших метаданих.

Solr – платформа повнотекстовий пошук з відкритим вихідним кодом, заснована на проекті Apache Lucene. Її основні можливості: повнотекстовий пошук, підсвічування результатів, Фасетноє пошук, динамічна кластеризація, інтеграція з базами даних, обробка документів зі складним форматом (наприклад, Word, PDF). Так як в Solr є можливість розподіленого пошуку та реплікації, Solr добре масштабується. Станом на травень 2016 року Solr є другим за популярністю пошуковим движком.

Solr написаний на Java і запускається як окремий веб-додаток повнотекстового пошуку (починаючи з версії 5.0 запускається, як самостійний додаток, а не всередині будь-якого контейнера сервлетів). Solr використовує Lucene в якості основи для реалізації індексації та пошуку. У Solr є HTTP / XML і JSON API, що робить можливим використовувати Solr з усіх більш-менш популярних мов програмування. Також Solr можна дуже гнучко налаштовувати і підключати до нього зовнішні модулі.

Apache Lucene – це бібліотека, що дозволяє організувати повнотекстовий пошук по безлічі документів, тобто пошук з використанням заданих ключових слів. Основним джерелом даних, що використовуються в Lucene в процесі пошуку, є індекс. Індекс являє собою сховище, яке включає в себе безліч документів. Основною складовою документів є поля з даними.

У Lucene основним типом даних є String. Існує також підтримка числових типів і типів Date з різною точністю збереження, але за замовчуванням Lucene надає досить небагато коштів для підтримки цих типів даних.

Для побудови пошукового індексу потрібно створити безліч документів з відповідними значеннями полів і додати створені документи в індекс.

Основні можливості

- масштабна і високошвидкісна індексація;
- понад 95GB в годину на сучасному обладнанні;
- потрібно малий обсяг RAM–«heap» всього 1MB;
- розмір індексу приблизно 20-30% від розміру початкового тексту;
- потужний, точний і ефективний пошуковий алгоритм;
- ранжируваних пошук–кращі результати показуються першими;

- безліч потужних типів запитів: запит фрази, wildcard запити, пошук інтервалів;
- пошук, заснований на «полях» (таких як заголовок, автор, текст);
- можливість сортувати по різних полях;
- multiple-index пошук з можливістю об'єднання результатів;
- можливість одночасного пошуку та оновлення індексу;
- кроссплатформенноє рішення;
- вихідний код повністю написаний на Java;
- наявність портів на інші мови програмування.

Перевага Apache Lucene в його простоті, високій швидкості роботи і низьких вимогах до ресурсів.

Lucene–найвідоміший з пошукових движків, споконвічно орієнтований саме на вбудовування в інші програми. Зокрема, його широко використовують в Eclipse (пошук по документації) і навіть в IBM (продукти з серії OmniFind). В плюсах проекту–розвинені можливості пошуку, хороша система побудови і зберігання індексу, який може одночасно поповняться, втечуть документи і проводиться оптимізація разом з пошуком, а також паралельний пошук по безлічі індексів з об'єднанням результатів. Сам індекс побудований із сегментів, проте для поліпшення швидкості рекомендується його оптимізувати, що часто означає майже ті ж витрати, що і на переіндексацію. Спочатку присутні варіанти аналізаторів для різних мов, включаючи російську з підтримкою стемінг (приведення слів до нормальної формі). Однак мінусом є все ж низька швидкість індексації (особливо в порівнянні з Sphinx), складність роботи з базами даних і відсутність API (крім рідного Java). І хоча для досягнення серйозних показників Lucene може кластерізіроваться і зберігати індекси в розподіленої файлової системи або базі даних, для цього потрібно сторонні рішення, так само як і для всіх інших функцій–наприклад, спочатку він вміє індексувати тільки звичайний текст. Але саме в плані використання в складі сторонніх продуктів Lucene «попереду планети всієї»–ні для якого іншого движка немає стільки портів на інші мови і використання. Одним з чинників такої популярності є і дуже вдалий формат файлів індексів, який використовують сторонні рішення, тому цілком можна будувати рішення, що працюють з індексом і виробляють пошук, але не мають власного індексатора (це легше реалізувати і вимоги набагато нижче).

Саме через такі широкі можливості ми використовуємо цю бібліотеку у нашій системі. Вона займає центральну роль у пошуку плагіату.

SBT – це сучасний інструмент для збірки програм. Хоча він написаний на Scala і надає безліч зручних можливостей Scala, але він може використовуватися і як інструмент для збірки загального призначення.

Akka–це бібліотека для мов Java і Scala, в якій реалізовані агенти, актори, взаємодія між акторами по мережі і багато інше. У першому наближенні це нагадує Erlang або Cloud Haskell, але насправді Akka куди могутніше. Наприклад, за допомогою Akka можна легко об'єднати декілька машин в кластер, в якому буде відслідковуватися зникнення і поява машин, а актори будуть автоматично переміщатися з машини на машину в міру зміни розмірів кластера. І це тільки один з прикладів.

Наша система, а конкретно фреймворк Play активно використовуе акторну модель у своему функціоналі для роботи з базою, запитами до серверу, або обробкою інформації.

Nginx – веб-сервер і поштовий проксі-сервер, що працює на Unix-подібних операційних системах (тестувалася збірка і робота на FreeBSD, OpenBSD, Linux, Solaris, Mac OS X, AIX і HP-UX). Починаючи з версії 0.7.52 з'явилася експериментальна бінарна збірка під Microsoft Windows.

Основний функціонал:

- обслуговування незмінних запитів, індексних файлів, автоматичне створення списку файлів, кеш дескрипторів відкритих файлів;
- акселерірованное проксінг без кешування, простий розподіл навантаження і відмовостійкість;
- підтримка кешування при акселерірованном проксінг і FastCGI;
- акселерірованная підтримка FastCGI і memcached серверів, простий розподіл навантаження і відмовостійкість;
- модульність, фільтри, в тому числі стиснення (gzip), byte-ranges (докачка), chunked відповіді, HTTP-аутентифікація, SSI-фільтр;
- експериментальна підтримка вбудованого Perl.

У nginx робочі процеси обслуговують одночасно безліч з'єднань, мультіплексіруя їх викликами операційної системи select, epoll (Linux) і kqueue (FreeBSD). Робочі процеси виконують цикл обробки подій від дескрипторів (див. Подієво-орієнтоване програмування). Отримані від клієнта дані розбираються за допомогою кінцевого автомата. Розібраний запит послідовно

обробляється ланцюжком модулів, що задається конфігурацією. Відповідь клієнту формується в буферах, які зберігають дані або в пам'яті, або вказують на відрізок файлу. Буфери об'єднуються в ланцюжки, що визначають послідовність, в якій дані будуть передані клієнтові. Якщо операційна система підтримує ефективні операції введення-виведення, такі як writev і sendfile, то nginx застосовує їх якнайшвидше.

Конфигурация HTTP-сервера nginx разделяется на виртуальные серверы (директива «server»). Виртуальные серверы разделяются на location’ы («location»). Для виртуального сервера возможно задать адреса и порты, на которых будут приниматься соединения, а также имена, которые могут включать «*» для обозначения произвольной последовательности в первой и последней части, либо задаваться регулярным выражением.

location’ы могут задаваться точным URI, частью URI, либо регулярным выражением. location’ы могут быть сконфигурированы для обслуживания запросов из статического файла, проксирования на fastcgi/memcached сервер.

Для эффективного управления памятью nginx использует пулы. Пул это последовательность предварительно выделенных блоков динамической памяти. Длина блока варьируется от 1 до 16 килобайт. Изначально под пул выделяется только один блок. Блок разделяется на занятую область и незанятую. Выделение мелких объектов выполняется путём продвижения указателя на незанятую область с учётом выравнивания.

Nginx содержит модуль географической классификации клиентов по IP-адресу. В его основу входит база данных соответствия IP-адресов географическому региону, представленная в виде radix tree (сжатое префиксное дерево или сжатый бор) в оперативной памяти. nginx предварительно распределяет первые несколько уровней дерева, таким образом, чтобы они занимали ровно 1 страницу памяти. Это гарантирует, что при поиске IP-адреса для первых нескольких узлов при трансляции адреса всегда найдётся запись в TLB.

Його ми використовуємо як проксі-сервер для віддачи статичних файлів, документів.

2.4.2 Вибір головного фреймворку серверної частини

Lift вважається одним з найскладніших і в той же час потужних web-фреймворків існуючих на даний момент (хоч і мало відомим), багато в чому тому що активно використовує функціональні можливості мови Scala. Що б його вивчити потрібно докласти чимало зусиль.

Напевно найголовніша особливість – це концепт View First (рис 2.2).

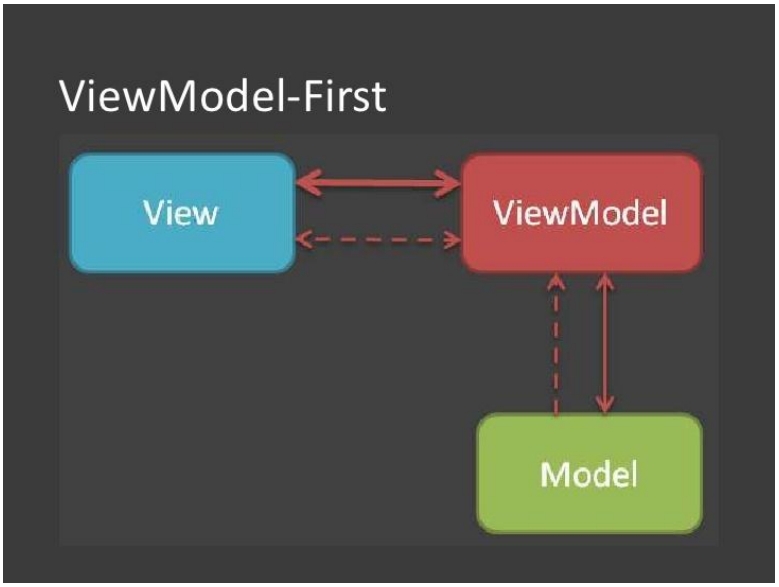

Рис. 2.2 – концепт View First

Це означає що View (XML-шаблон) – точка входу в ваш додаток (на відміну від традиційних способів, де контролер – вхідна точка). Scala-код не вбудований у View (в Lift – неможливо писати логіку на сторінці), але викликається з шаблону. Завдяки концепту View First, кожен елемент (форма, поле для пошуку, чат, список чого-небудь і т.д.) може бути повністю ізольований від інших в окремому сніпеті. Це дуже сильно зменшує зв'язаність компонентів і дозволяє легко їх перевикористати в інших частинах сайту.

Вся конфігурація робиться на Scala. Це означає що у вас немає необхідності реалізовувати 40% вашого коду в XML-файлах, що б створити гнучку конфігурацію. Так само це означає що компілятор вам буде підказувати якщо ви десь вказали невалідний параметр в конфігурації. Це дуже зручно коли у проекту велика і довга конфігурація. Ліфт гарантує захист від багатьох поши-

рених вразливостей за версією OWASP. Найцікавіше що вам не потрібно нічого для цього робити, досить просто писати код на Lift.

Деякі речі з Lift, які в інших фреймворків робляться значно складніше або взагалі не реалізовані:

- відкладене завантаження: можна помітити частині сторінки, які генеруються довго, Lift подбає про те, щоб інші частини завантажилися відразу, а повільні – як тільки будуть готові.
- Паралельно генерація сторінки: можна помітити кілька ділянок сторінки для Паралельно генерації, для них автоматично будуть виділені нові потоки.
- Дружні дизайнерам шаблони: в останній версії стали ще красивіше – інструкції для Lift передаються через другий атрибут class. У моєму проекті, наприклад, шаблони можна відкривати як звичайний HTML і вони виглядають в точності як сторінка додатки, тільки без динаміки.
- Багатосторінкові форми (Wizard): декларативним способом описані форми, що підтримують кнопку Back, які можна запускати або багатосторінкові, або через Ajax. Раніше ці форми були фішкою JBoss Seam, тепер в Lift це ще зручніше.
- Безпека: завдяки грамотному використанню статичної типізації та функціонального програмування, Lift гарантує захист від більшості найбільш небезпечних для інтерактивних веб додатків вразливостей.

Play Framework – це MVC (рис. 2.3) веб-фреймворк для мов програмування Java і Scala від компанії Typesafe. З одного боку, Play має гнучкість і простотою у використанні фреймворків типу Django або Mojolicious, з іншого – в ньому реалізовані багато ідей, наприклад, модульна, а отже і висока продуктивність, сувора статична типізація і тд, які ми можемо спостерігати, скажімо, в Yesod.

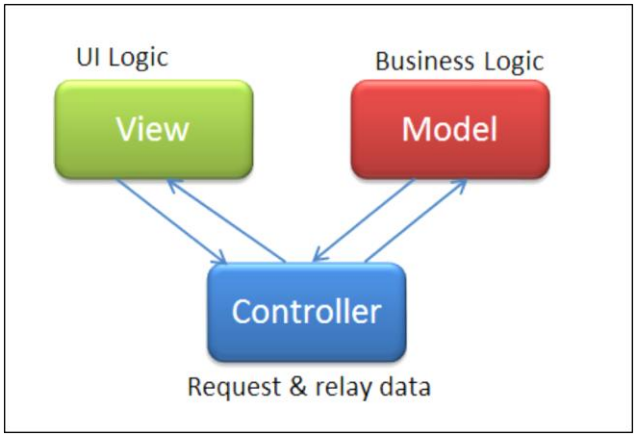

Рис. 2.3 – концепт MVC

Фреймворк заточений на те щоб якомога швидше почати розробляти і при цьому відразу бачити проміжний результат

Усередині фреймворка знаходиться embeded web-сервер і кастомний classloader. В першу чергу запуск проекту без попереднього налаштування Томкет итд + перекомпиляция вихідних кодів на льоту. Тобто додали новий обробник, оновили сторінку – він підхоплюється. Винятки треба зробити для прекомпіленних ресурсів: плагінів, бібліотек, итд. При додаванні бібліотеки, додаток необхідно перезапускати

Фреймворк вже включає в себе інструменти першої необхідності і дозволяє не витрачати часу на їх початкові установки.

Саме через ці плюси був обраний цей фреймворк. Але попередній фреймворк дійсно кращий за багатьма пунктами.

2.4.3 Вибір системи керування версіями проекту

Git – розподілена система керування версіями файлів та спільної роботи. Git, на відміну від Subversion і подібних до неї систем, не зберігає інформацію як список змін (патчів) для файлів. Замість цього Git зберігає дані набором зліпків. Кожного разу при фіксації поточної версії проекту Git зберігає зліпок того, як виглядають всі файли проекту. Але якщо файл не змінювався, то дається посилання на раніше збережений файл (рис. 2.4).

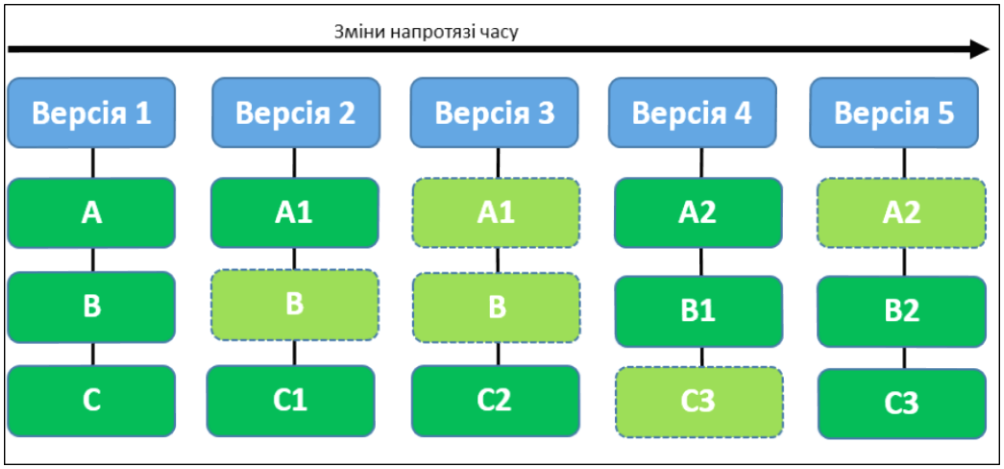

Рис. 2.4 – концепція роботи Git

Git схожий на своєрідну файлову систему з інструментами, які працюють поверх неї. Для кожного відстежуваного файлу Git зберігає розмір, час створення і останньої зміни. Ці дані зберігаються у файлі index, який знаходиться у теці .git Вся база даних Git зберігається в теці з назвою .git.

В Git файли можуть знаходитися в одному із 3-х станів: зафіксованому (файл вже збережено в локальній базі даних), зміненому (файл було змінено, але зміни не зафіксовано) і підготовленому (файли було змінено і відмічено для фіксації).

Більшість дій можна виконувати на локальній файловій системі без використання інтернет підключення. Вся історія змін зберігається локально і при необхідності вивантажується у віддалений репозиторій. На відміну від Subversion, де без підключення до інтернету можна лише редагувати файли, але зберегти зміни в вашу базу даних неможливо (оскільки вона відключена від репозиторія). Будь-який коміт спочатку робиться локально, а потім вивантажується у віддалений репозиторій.

В своїй базі Git зберігає все по хешам файлів. Як хешуюча функція використовується SHA-1. Перед кожним збереженням файлів Git обчислює SHA-1 хеш файлу і отриманий хеш стає індексом файлу в Git. Використовуючи хеш Git легко відслідковує зміни в файлах.

Галуження — це розмежування від основної лінії розробки. Git дозволяє створити декілька гілок і перемикатися між ними. Це корисно, оскільки

дозволяє працювати декільком розробникам над своїм функціоналом не заважаючи іншим і не псуючи основу гілку. За замовчуванням, Git створює гілку з назвою master. Гілка в Git просто являє собою вказівник на одну із фіксацій. При кожній новій фіксації гілка в Git рухається автоматично (тобто перемикається на фіксацію). Гілка є простим файлом, який містить 40 символів контрольної суми SHA-1 фіксації. Створення нової гілки дуже швидке, оскільки це однаково запису в файл 41 байта (40 символів + символ нового рядка).

При роботі з Git ви повсюди зустрічатимете такі хеші, адже Git постійно їх використовує. Фактично, Git зберігає все не за назвою файлу, а саме за значенням хешу його змісту.

Цей функціонал вбудовано в систему на найнижчих рівнях і є невід’ємною частиною її філософії. Ви не можете втратити інформацію при передачі чи отримати пошкоджений файл без відома Git.

Git підтримує два способи для інтеграції змін з гілки в гілку: merge (зливання) та rebase (перебазування). Основна різниця полягає в тому, що rebase запам'ятовує фіксації у вигляді патчів, перемотує гілку і застосовує патчі у вигляді фіксацій на відміну від merge, який зливає дві гілки в одну.

Для цього проекту він виявися дуже зручним так як для нього були знайдені плагіни для швидкого деплою проекту на сервер хостера. Git вже став стандартом розробки у наш час і використовується майже усюди

3 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Проектування інформаційної системи за допомогою методології функціонального моделювання SADT (Стандарт IDEF0)

При проектуванні системи виявлення плагіату в наукових та студентських роботах була обрана методологія функціонального моделювання SADT (Structured Analysis and Design Technique) – (стандарт IDEF0).

Методологія SADT являє собою сукупність методів, правил і процедур, призначених для побудови функціональної моделі об'єкта будь-якої предметної області. IDEF0 – методологія та стандарт функціонального моделювання. За допомогою графічної мови IDEF0, система для виявлення плагіату постає у вигляді набору взаємопов'язаних функціональних блоків. Моделювання засобами IDEF0, як правило, є першим етапом вивчення системи.

Контекстна діаграма є вершиною деревовидної структури діаграм і являє собою саме загальний опис системи та її взаємодія з зовнішнім середовищем. Проектування починається з представлення системи як єдиного цілого – одного функціонального блоку з граничними стрілками, які простираються за межі аналізованої області. Контекстна діаграма інформаційної системи наведена на рис. 3.1.

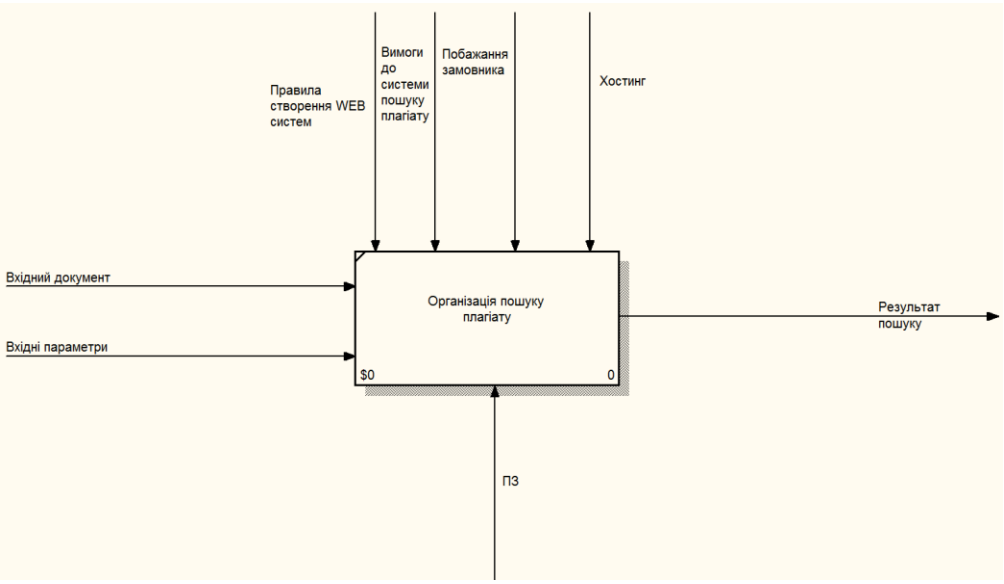

Рисунок 3.1 – Контекстна діаграма інформаційної системи

На контекстній діаграмі відображена головна робота системи «Організація пошуку плагіату». На вхід подається інформація про текст та початкові параметри. Головна робота керується: правилами створення Web-системи, бажаннями замовника, особливостями хостингу і вимогами до системи. Виходом є: результат пошуку.

Після опису системи в цілому проводиться розбиття її на великі фрагменти. Цей процес називається функціональна декомпозиція, а діаграми, які описують кожен фрагмент і взаємодію фрагментів, називаються діаграмами декомпозиції .

Після декомпозиції контекстної діаграми проводиться декомпозиція кожного великого фрагмента системи на більш дрібні і так далі, до досягнення потрібного рівня подробиці опису. Після кожного сеансу декомпозиції проводяться сеанси експертизи – експерти предметної області вказують на відповідність реальних процесів створеним діаграмам. Знайдені невідповідності виправляються, і тільки після проходження експертизи без зауважень можна приступати до наступного сеансу декомпозиції. Так досягається відповідність моделі реальним процесам на кожному рівні декомпозиції моделі. Синтаксис опису системи в цілому і кожного її фрагмента однаковий у всій моделі.

Після декомпозиції контекстної діаграми отримуємо 3 блоку – роботи. Ці блоки представляють основні під функції початкової функції.

Функція «Розробка Web-системи» включає в себе повну розробку інформаційної системи на локальному комп'ютері. Включає в себе розробку інтерфейсу, програмних додатків і баз даних. Входом у неї є вхідні параметри такої системи, так як вони потрібні для наповнення БД. Управляється за допомогою правил створення Web-системи та вимогами до такої системи. Механізмом є програмне забезпечення, яке потрібне для розробки. І результатом роботи є готова Web-система.

Функція «Розміщення Web-системи» служить для можливості отримати доменне ім'я, а потім помістити Web-систему на хостинг. Входом для роботи є готова Web-система для розміщення. Управляється робота бажаннями замовника, особливостями хостингу та Інтернет провайдером. Механізмом є – програмне забезпечення.

Функція «Реалізація пошуку плагіату» призначена для того, щоб розробити все те, що потрібно для аналізу документа, його індексування та знаходження подібних до нього. У даної роботи є три входи, це: вхідні парметри, вхідний до-

кумент і доменне ім'я Web-системи, розміщеної на хостингу. Управляється вимогами до створення системи виявлення плагіату і за допомогою Інтернет провайдера. Механізмом є – програмне забезпечення. Виходом даної роботи є результат пошуку(рис. 3.2).

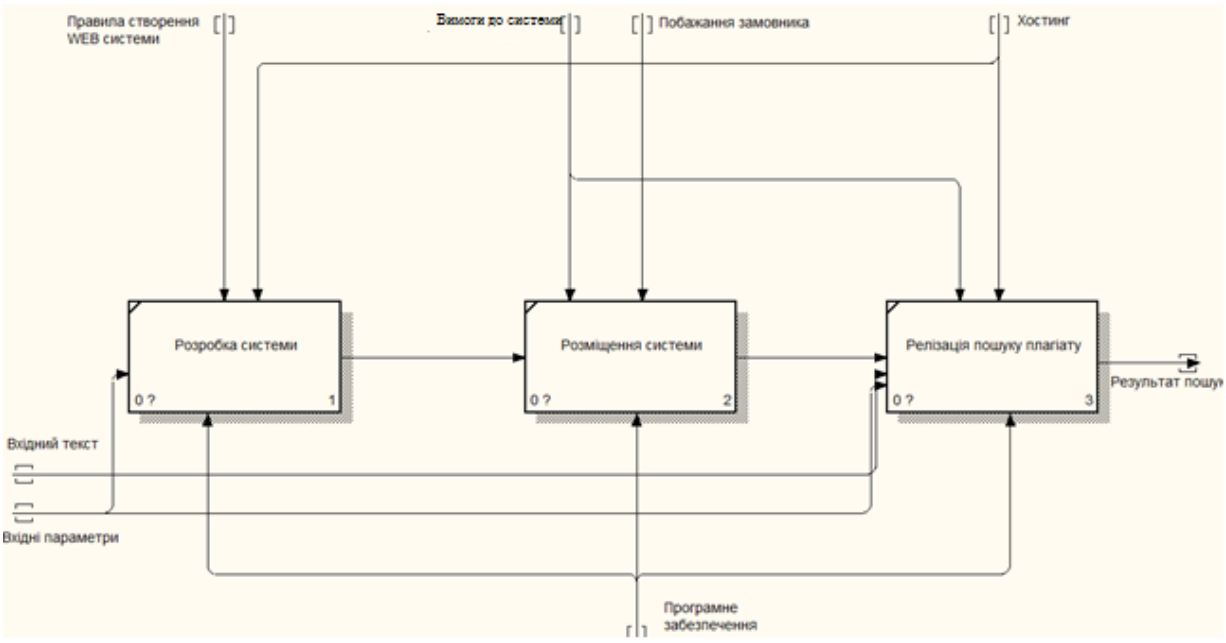

Рисунок 3.2 – Діаграма декомпозиції організації допомоги пацієнтам

При декомпозиції першого А1 блоку виділяються наступні блоки:

«Створення сервісу» – входу у даної роботи немає, управління – правила створення Web-системи; механізм – програмне забезпечення; вихід – сервіс Web-системи.

«Розробка програмної частини» – входом є сервіс – вихід попередньої роботи; управлінням є, як і в попередньому блоці, правила розробки WEB-систем, але додалися рекомендації створення систем антиплагіат; механізм, як і у всіх роботах, є – програмне забезпечення; вихід робоча Web-система.

«Створення БД» – даний блок управляється правилами розробки БД; робота виконується за допомогою програмного забезпечення; виходом є – готова БД.

«Наповнення БД» – робота має три входи: готова БД, робоча Web-система, документи; керується рекомендаціями до створення систем

антиплагіат; механізмом є програмне забезпечення. Вихід з даної роботи є готова Web-система, яка переходить входом на наступну роботу.

Діаграма декомпозиції першого A1 блоку представлена на рисунку 3.3.

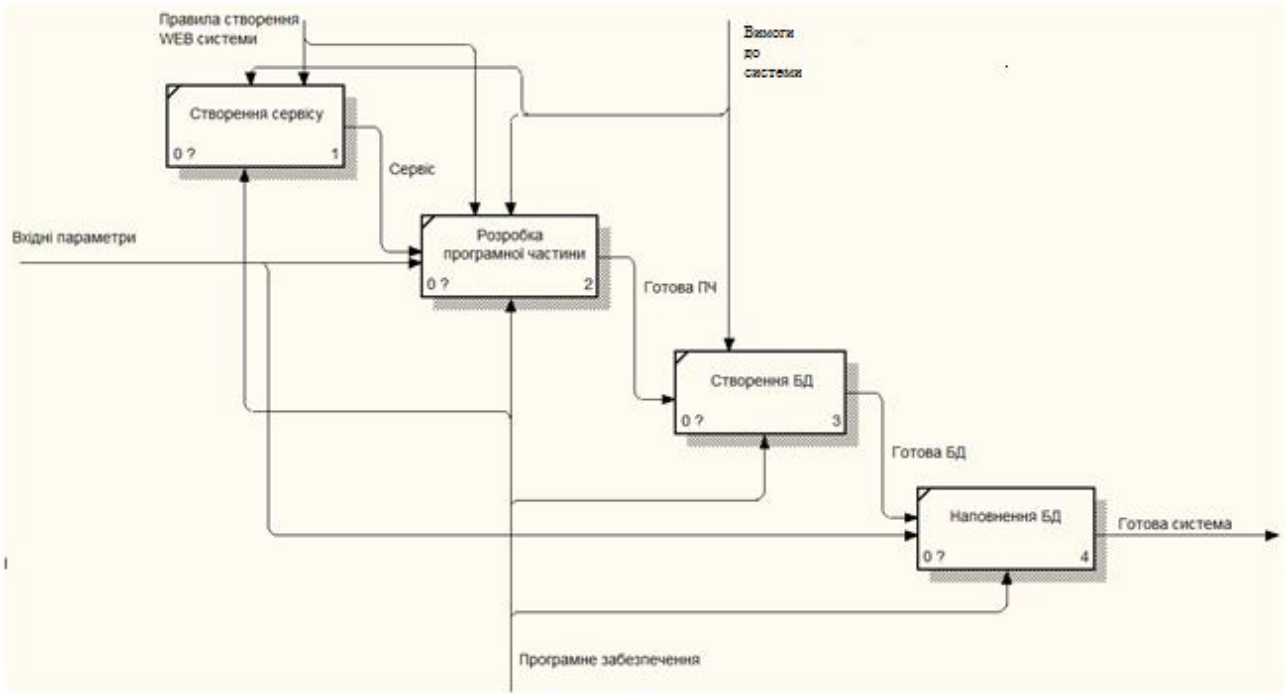

Рисунок 3.3 – Діаграми декомпозиції першого A1 блоку

При декомпозиції другого A2 блоку виділені 4 роботи:

«Визначити доменне ім'я» – робота керується бажаннями замовника та Інтернет провайдером; виконується за допомогою програмного забезпечення; виходом є ID системи, тобто саме доменне ім'я.

«Визначити зону реєстрації» – входом є ID системи, управляється бажаннями замовника та Інтернет провайдером; механізм це – програмне забезпечення, за допомогою якого виконуються всі операції; виходом роботи «Визначити зону реєстрації» буде готове доменне ім'я, яке буде відображатися в рядку браузера для клієнта медичного закладу.

«Реєстрація на хостингу» – для проведення реєстрації на хостингу потрібний готовий домен; управлятися буде особливостями хостингу та Інтернет провайдером; механізмом є – програмне забезпечення; виходом буде логін і пароль від облікового запису на хостинг сервері.

«Перенесення сайту на хостинг» – для перенесення сайту на хостинг потрібен логін і пароль від облікового запису на хостинг сервері і готова система; при цьому керуємося хостингом та Інтернет провайдером; механізм: програмне забезпечення; результатом роботи є вміщена на хостинг Web-система. При декомпозиції третього А3 блоку виділені 3 роботи:

«Перетворення у потрібний формат»–робота з вхідним документом, яка полегшує його індексацію та пошук плагіату.

«Організація індексації документів» – для організації індексації та перетворення документів у потрібний формат; робота буде керуватися загальними рекомендаціями та Інтернет провайдером; механізм – це програмне забезпечення; результатом роботи є готовий документ потрібного формату.

«Пошук за хешем» – дана розробка керується загальними рекомендаціями, та Інтернет провайдером; механізм даної роботи – це програмне забезпечення, а результатом є знайдені документи. Діаграма декомпозиції третього А3 блоку представлена на рисунку 3.4.

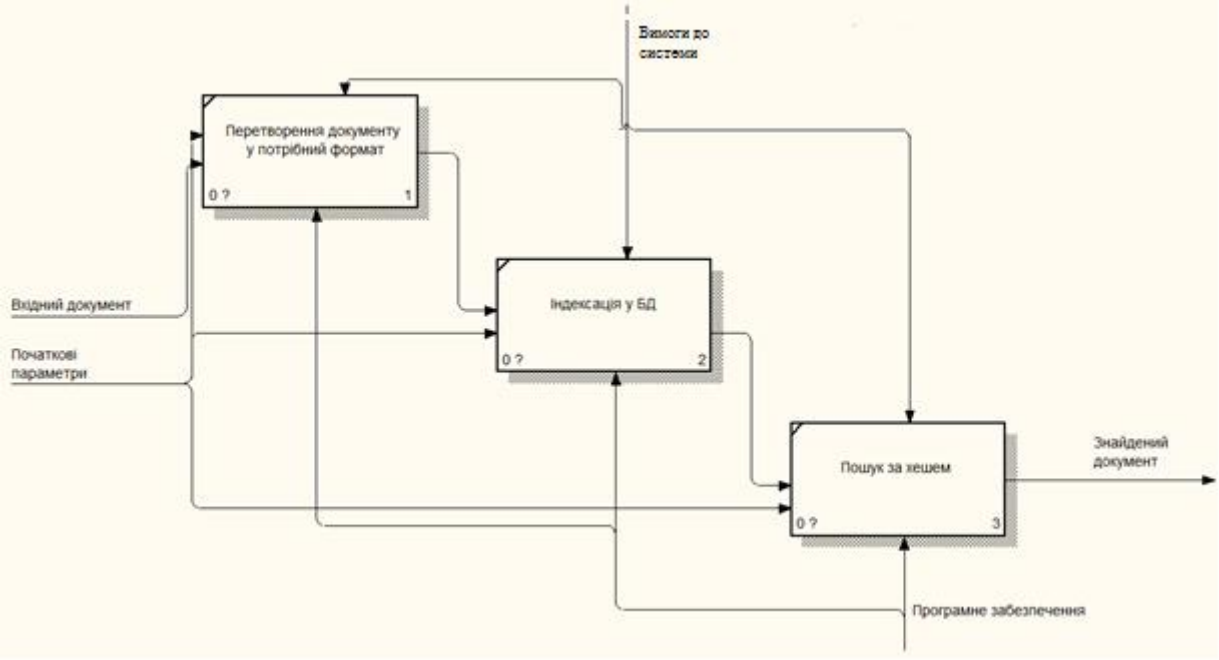

Рисунок 3.4 – Діаграми декомпозиції третього А3 блоку

3.2 Проектування інформаційної системи за допомогою послідовного виконання процесів Workflow Diagramming (Стандарт IDEF3)

Проектування системи виявлення плагіату в наукових та студентських роботах було здійснено за допомогою послідовного виконання процесів Workflow Diagramming (Стандарт IDEF3).

За допомогою IDEF3 описується логіка виконання дій. IDEF3 може використовуватися самостійно і спільно з методологією IDEF0: будь-який функціональний блок IDEF0 може бути представлений у вигляді послідовності процесів або операцій засобами IDEF3. Якщо IDEF0 описує, що робиться в системі, то IDEF3 описує, як це робиться.

Контекстна діаграма в IDEF3 відображає основну функцію системи. Вона складається з єдиної роботи – «Організація пошуку плагіату». Контекстна діаграма представлена на рисунку 3.5.

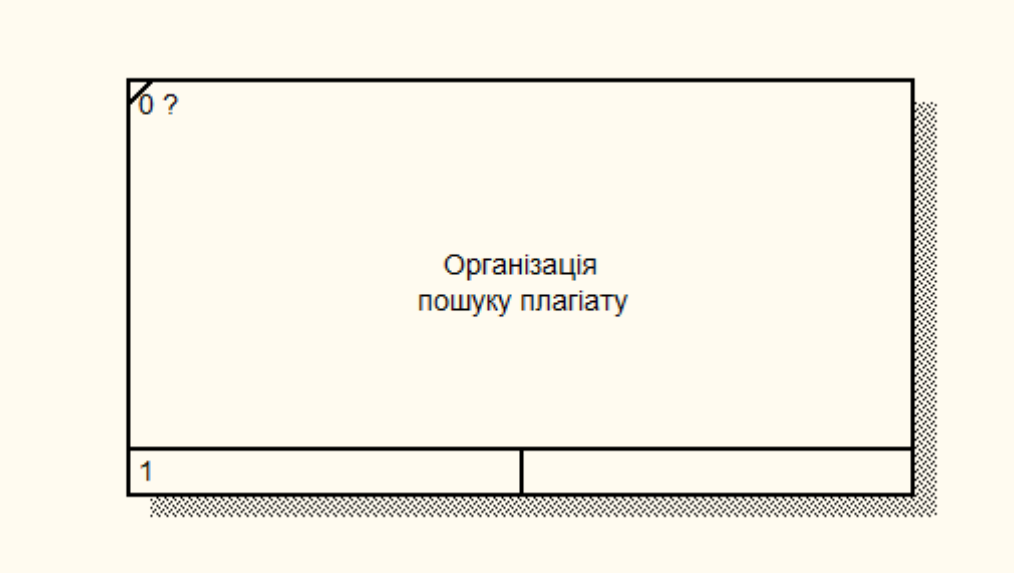

Рисунок 3.5 – Контекстна діаграма системи

Провівши декомпозицію контекстної діаграми, спостерігається послідовність виконання робіт.

Першою роботою системи є «Розробка WEB-системи». Після неї йде робота під назвою «Розміщення WEB-системи». Зв'язок між цими роботами означає, що робота-приймач може завершитись ще до закінчення роботи-джерела. Далі робота «Реалізація пошуку» пов'язана з блоком «Розміщення

WEB-системи» старшим зв'язком, означає те, що всі попередні роботи повинні, завершиться для того, щоб можна було без перешкод здійснити пошук.

Діаграма декомпозиції, представлена на рис. 3.6.

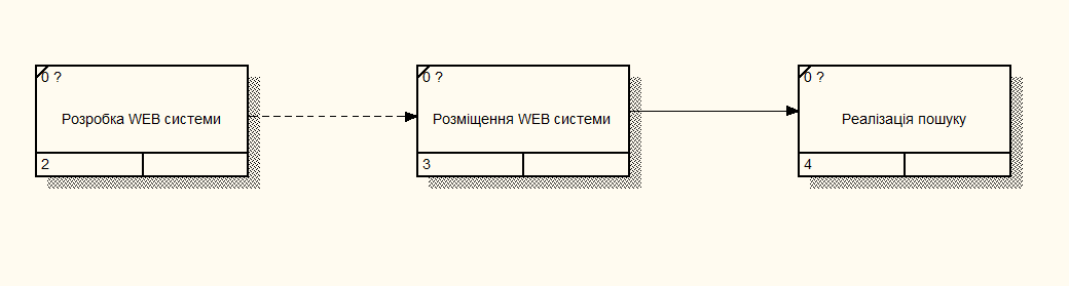

Рисунок 3.6 – Діаграма декомпозиції інформаційної системи

При декомпозиції наступного рівня роботи «Розробка WEB-системи» отримали чотири блоки – це роботи з двома перехрестями. Перше перехрестя «Асинхронне І», означає, що подальші роботи можуть початися не одночасно, але обов'язково повинні бути запущені. Це такі роботи як: «Розробка програмного коду» та «Створення сервісу». При злитті стрілок-виходів з цих робіт, використовується таке ж перехрестя «Асинхронне І». Означає те, що роботи мають, завершиться, але це може бути не одночасно. Далі перехрестя і робота «Створення БД» пов'язані з допомогою відношення І «Створення БД» і «Наповнення БД» пов'язані старшим зв'язком, тому що перш ніж наповнювати БД, її потрібно спочатку створити.

Діаграма декомпозиції, представлена на рис. 3.7.

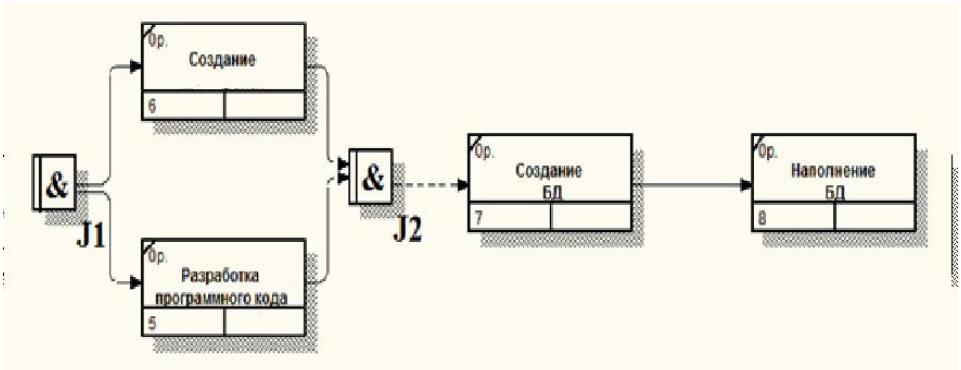

Рисунок 3.7 – Діаграма декомпозиції роботи «Розробка WEB-системи»

При декомпозиції роботи «Розміщення WEB-системи» всі роботи пов'язані старшим зв'язком і розміщені послідовно. А саме: «Визначити доменне ім'я», «Визначити зону реєстрації», «Реєстрація на хостингу», «Перенесення сайту на хостинг». Діаграма декомпозиції, представлена на рис. 3.8.

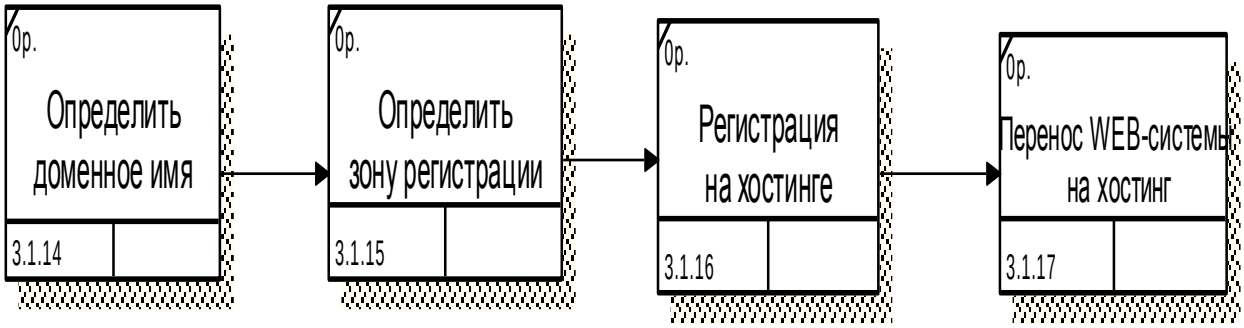

Рисунок 3.8 – Діаграма декомпозиції роботи «Розміщення WEB-системи»

При декомпозиції блоку «Реалізація пошуку» (рис. 3.9) отримали 3 блоки робіт, які пов'язані між собою простим зв'язком.

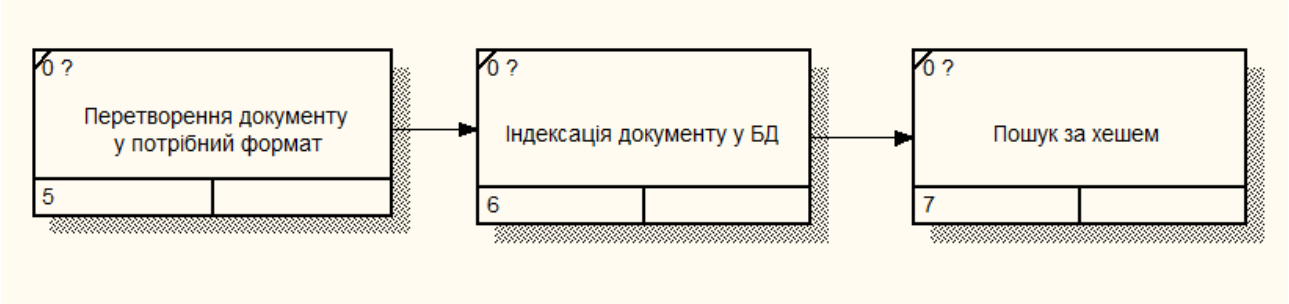

Рисунок 3.9 – Діаграма декомпозиції роботи «Реалізація пошуку»

Робота «Пошук за хешем» може початись тільки тоді, коли завершиться робота «Перетворення документу у потрібний формат»

3.3 Проектування інформаційної системи за допомогою діаграми потоків даних DFD

У відповідність з розглянутими методологіями системи виявлення плагіату в наукових та студентських роботах визначається як ієрархія діаграм потоків даних DFD, що описують процес перетворення інформації від введення в систему до видачі інформації користувачеві.

Діаграми потоків даних використовуються для опису руху документів і обробки інформації як додаток до методології функціонального моделювання IDEF0. На відміну від методології IDEF0, стрілки на діаграмах DFD показують лише те, як об'єкти (включаючи дані) рухаються від однієї роботи до іншої. Діаграма потоків даних DFD – це граф, на якому показано рух значень даних від їх джерел через перетворюючи їх процеси до їх споживачів в інших об'єктах.

Діаграми верхніх рівнів ієрархії (контекстні діаграми) відображають зв'язок основного процесу системи із зовнішніми сутностями, які визначаються відповідними входами і виходами. Контекстні діаграми деталізуються за допомогою діаграм нижнього рівня. Така декомпозиція триває, створюючи багаторівневу ієрархію діаграм, до тих пір, поки не буде досягнутий такий рівень декомпозиції, на якому процеси стають елементарними і деталізувати їх далі неможливо.

У контекстній діаграмі головним процесом системи є «Організація пошуку плагіату». Зовнішніми сутностями, які впливаю на систему, є: «Замовник», «Документ» і «Хост-сервер». Існує блок як сховище даних. Зв'язок між замовником і головною роботою полягає в «Бажанні» і «Бюджеті». Із системи йдуть дані в зовнішні сутності «Документ» – пошук плагіату, і в «Хост –сервер» – Web-система. Зі сховища даних в систему передаються дані про індексовані документи. Контекстна діаграма представлена на рис. 3.10.

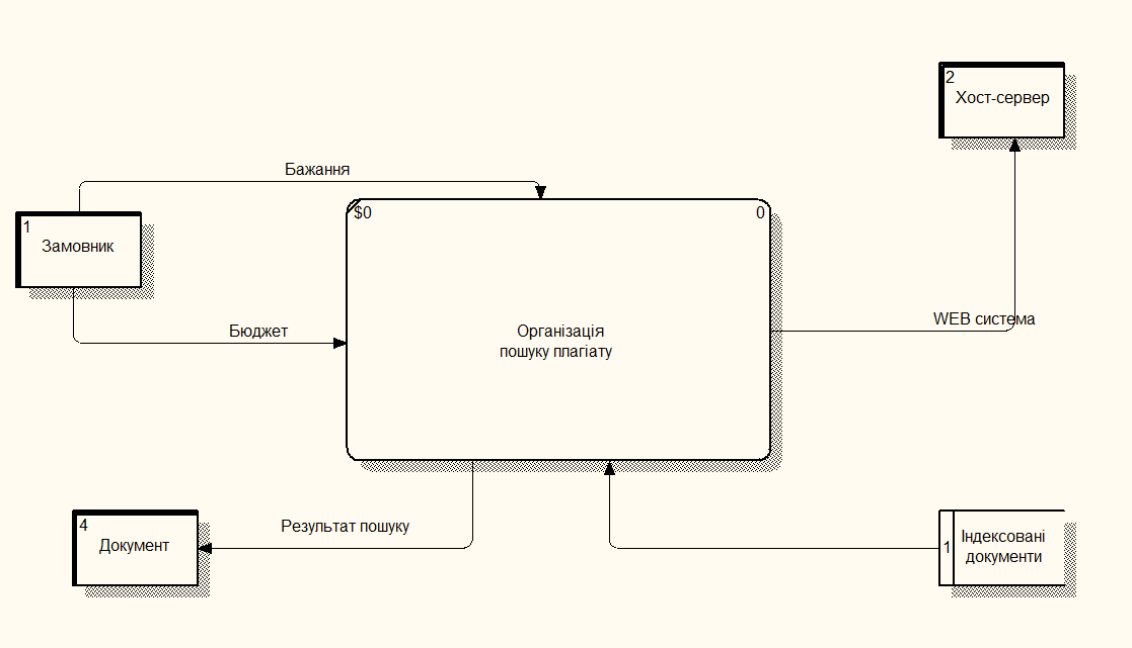

Рисунок 3.10 – Контекстна діаграма системи виявлення плагіату в науко-вих та студентських роботах.

Для головного процесу, присутнього на контекстній діаграмі, проводиться декомпозиція. На першому рівні ієрархії показані основні внутрішні процеси системи, відповідні їм зовнішні сутності.

Перший процес системи – «Розробка Web-системи», приймає потік даних з зовнішньої сутності «Замовник» – Бюджет замовника і Бажання замовника, і з сховища даних – «Індексовані документи». Вихідний потік даних з цього блоку це «Інтерфейс системи(сервіс)».

На другий блок – «Надання допомоги» подається вихідний потік даних з блоку «Розробка Web-системи», а вихідним є «Готова система», який є входом в зовнішнє сутність « Клієнт».

Наступний блок «Розміщення на хостингу». Вхідними даними для нього є дані із зовнішньї сутності «Замовник», а саме: «Бюджет» та «Бажання».І резуль-татом роботи є «Web-система», яка входить в зовнішню сутність «Хост-сервер». Діаграма декомпозиції представлена на рис. 3.11.

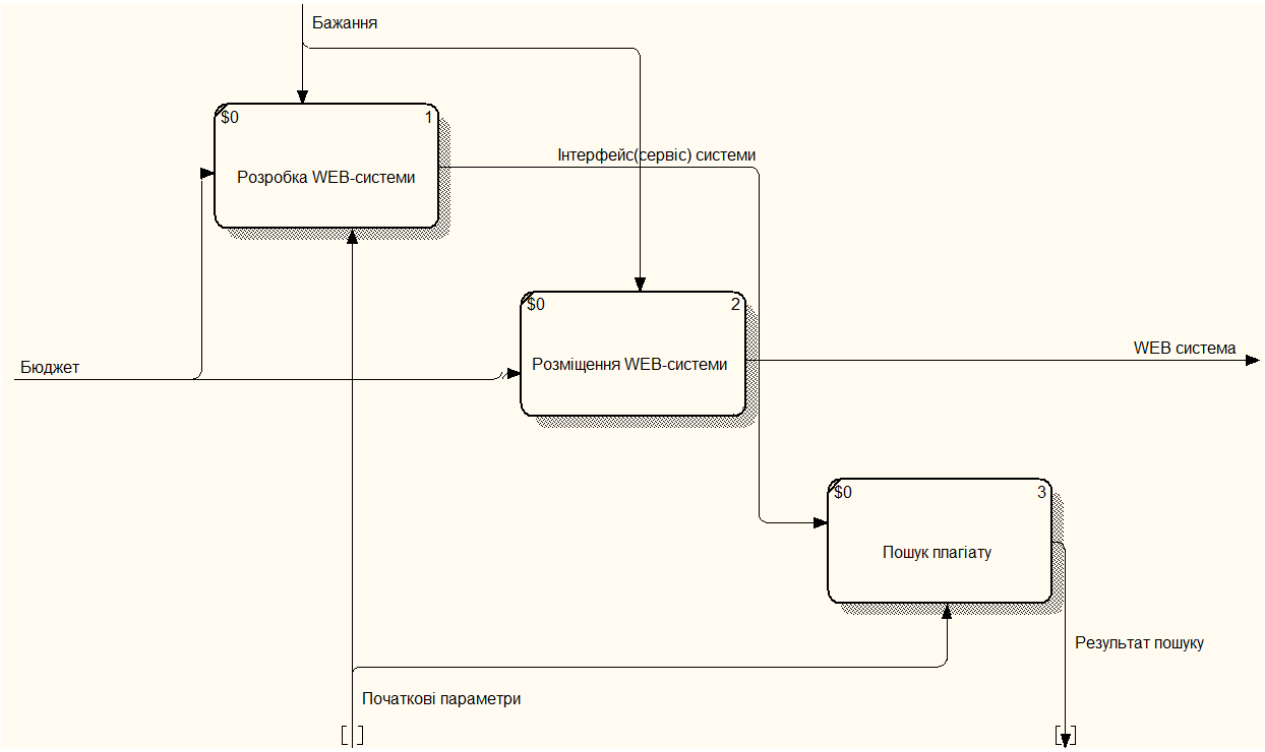

Рисунок 3.11 – Діаграма спеціалізованої системи виявлення плагіату в наукових та студентських роботах.

3.4 Проектування бази даних системи

Для представлення структури системи виявлення плагіату в наукових та студентських роботах обрана не реляційна модель представлення даних.
Документи представлені одним індексом у сховищі Solr.

Був створений один індекс – dup_documents (табл. 3.1) у якому зберігаються усі проіндексовані документи.

Індекс складається з полів, які мають свій тип та свої на лаштунки.

Існує два види оголошення полів

– статичний;
– динамічний.

Ці поля обробляються по-різному Solr. Статичний тип поля має тільки одну назву. Динамічні поля–це поля доступні під багатьма іменами. Насправді імена динамічних полів є простим регулярним виразом (ім'я, яке починається і закінчується на '*' знак). Зверніть увагу, що Solr спочатку вибирає статичні по-

ля, а потім динамічне поля. Крім того, якщо ім'я поля відповідає більш ніж одному визначенню, Solr вибере поле з більш довгою назвою pattern.

– name–ім'я поля (обов'язковий атрибут);
– type–тип поля, яке є одним з визначених типів (обов'язковий атрибут);
– indexed–чи має поле індексуватися (встановити true, якщо ви хочете знайти або впорядкувати по цьому полю);
– stored–зберегти вихідні значення (встановити true, якщо ми хочемо отримати початкове значення поля).

Головними для нас є: indexed, stored.

Таблиця 3.1 – Сутність «Dup-documents»

Атрибут	Тип	Indexed	Stored
content	text_general_rev	false	true
content_lg	string	true	true
content_sm	string	true	true
doc_title	string	true	true
first_word	string	true	true
id	string	true	true
md5	string	true	true
md5_hash	Strings[array]		
path	string	false	true

Однієї цієї сутності вистачає для нормального функціонування системи.

У результаті проведено проектування бази даних системи, визначені основні сутності та атрибути бази даних системи, які дозволяють зберігати та здійснювати доступ до всієї необхідної інформації. База може бути розширена в залежності від потрібного функціоналу системи.

3.5 Проектування алгоритму пошуку плагіата

Були проаналізовані існуючі алгоритми пошуку запозичень:

– метод шинглів і його модифікації (метод "Супершінглів", "Surrounding Context NGrams");
– I-Match • Метод Опорних слів;

– метод коефіцієнта збігу документів;
– методи, які використовують зовнішні пошукові системи;

І серед них був обраний найоптимальніший для цієї системи варіант – трохи змінений метод шинглів.

Сам пошук бере на себе наша пошукова платформа Solr, але до того як документ буду проіндексований нам потрібно пройти через деякі кроки перетворення та його модифікації:

– нормалізація тексту. Видалення стоп-слів, розділових знаків, перетворення тексту у нижній регістр, видалення займенників;
– отримання md5 хешу від результату нормалізації;
– отримання NGram шинглів від результату нормалізації;
– отримання заздалегідь заданої кількості мінімумів усіх хеш-функцій для множини S.

Коли усі ці кроки виконані відбувається пошук за цими хеш-функціями у нашій пошуковій платформі. Можна проіндексувати вхідний текст, але це не обов’язково. Завдяки алгоритму MinHash цей пошук займає секунди на 50 тисячах проіндексованих документах. Також була встановлена можливість обмеження результатів пошуку і це пришвидшило його ще більше.

3.6 Проектування REST API для доступа до функцій системи

Для доступа до основних функцій системи було створено REST API, через яке зовнішній клієнт має можливість індексувати документи у системи, або перевіряти ці документи через Solr.

У загальному випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL. Кожна URL в свою чергу має строго заданий формат.

Було створений головний сервіс с доступом до таких основних методів:

– індексування документу у БД;
– пошук плагіату у цьому документі за локальною базою;
– індексація архіву з документами;
– видалення документу за його id;
– завантаження оригіналу документа з бази.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ

4.1 Огляд сктруктури основного проекту ситеми пошуку плагіату

Спочатку проект складався с 2 частин (рис. 4.1): main project server, ui server. Потім серевер фронтенду був винесений окремо.

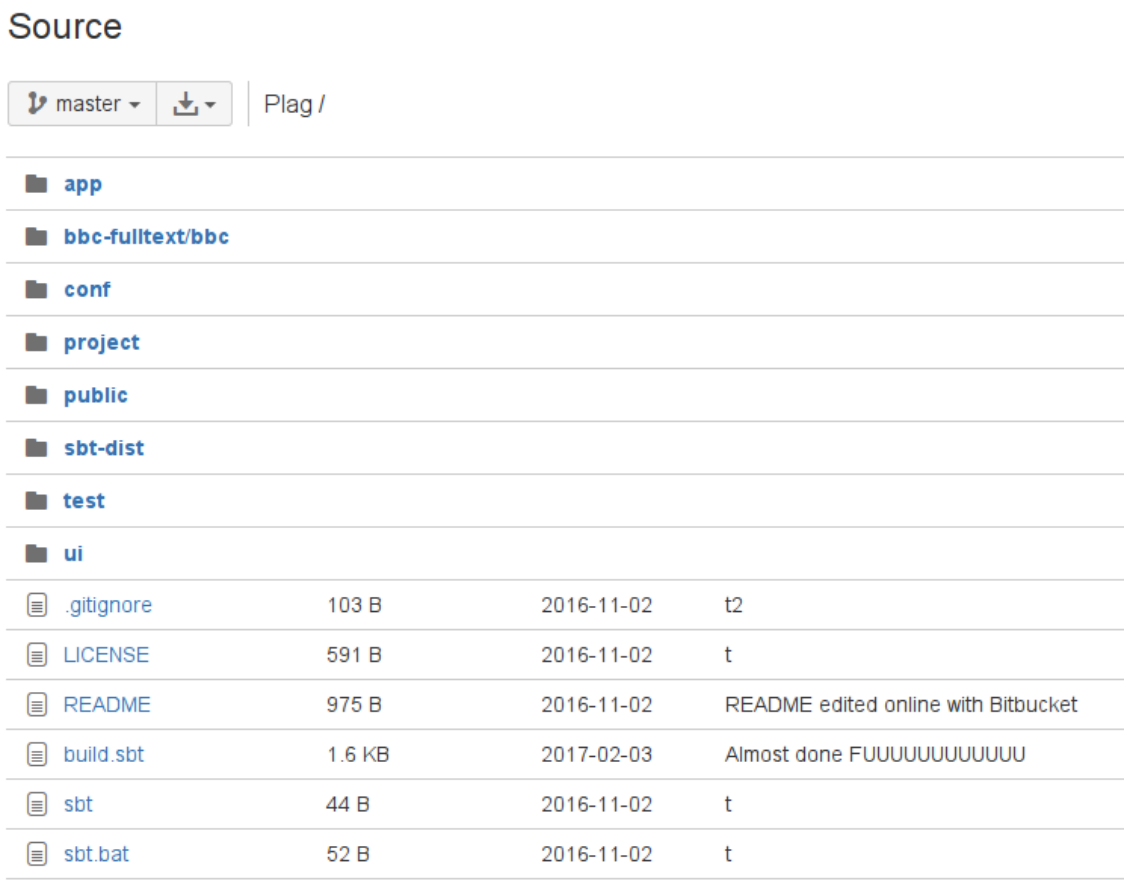

Рисунок 4.1 – Структура проекту

Як можна побачити, уся логіка була розділена на різні проекти з можливістю спільної конфігурації серверів через Simple Build Tool (SBT). Іншими словами була можливість керувати роботою серверів централізовано. Але згодом від такої конфігурації було вирішено відмовитись, та повністю розділити проекти

4.1.1 Головний модуль app

Головний модуль системи має свою структуру (рис 4.2). Він складється з таких компонентів:

- контроллери де відбувається уся взаємодія з клієнтами, визначаються методі для доступу до основних функцій системи;
- фільтри за дапомогою яких можна керувати та обробляти запити від клієнтів, змінювати їх;
- сервіси де знаходиться уся логіка роботи та реалізація алгоритмів пошуку;
- утилітний компонент містить в собі класи та методи які не увійшли до головних інтерфейсів;
- компонент відображення.

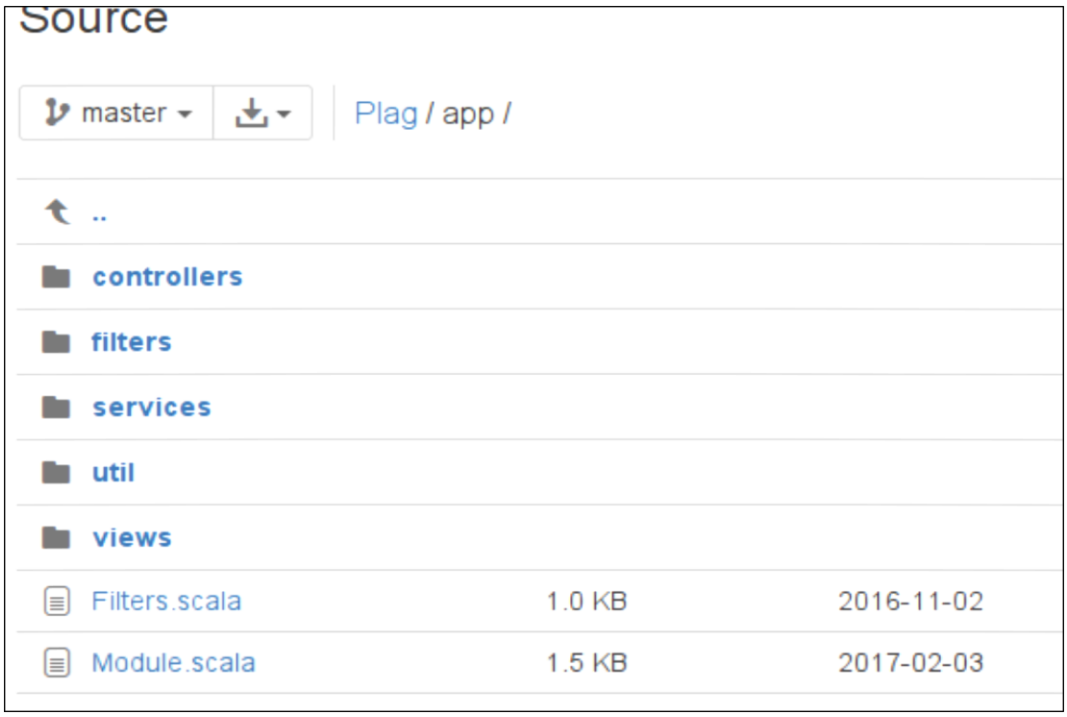

Рисунок 4.2- Структура модуля app

Розглянемо контроллери (рис. 4.3) та сервіси (рис. 4.4) більш детальніше. Головним контроллером у цій системи є DocumentController. Це наш інтерфейс для доступу до роботи з сервісом пошуку плагіату. У ньому були описані такі методи:

– /api/checkFileNoUpload POST. Відповідає за перевірку документу на плагіат без його індексації у базі;
– /api/checkFileUpload POST. Відповідає за перевірку документу на плагіат з його індексацією;
– /api/download?id= GET. Відповідає за вилучення оригіналу файла за бази;
– /api/upload POST. Індексує файл у базу.

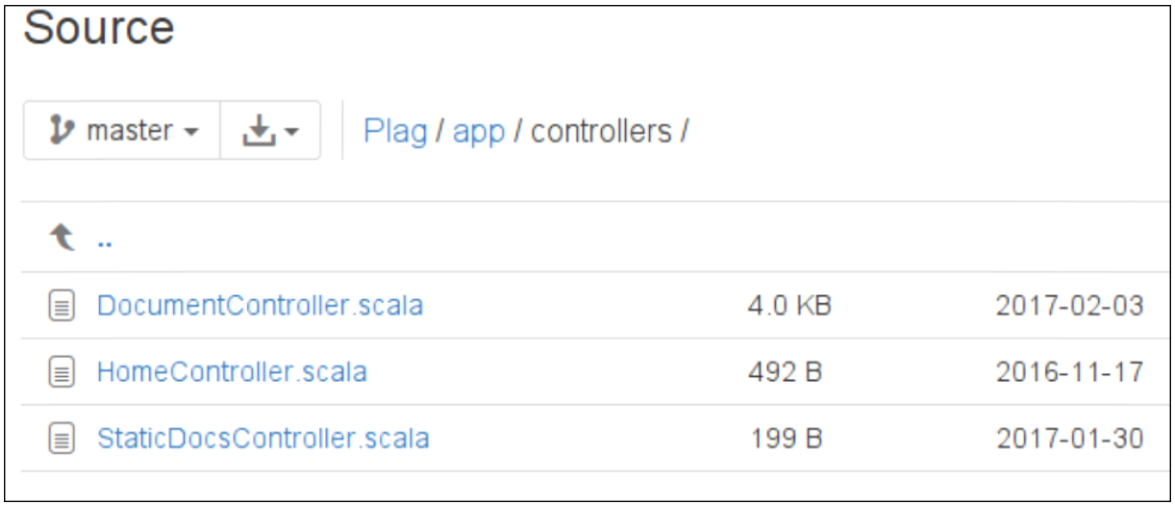

Рисунок 4.3- Наявні контроллери

Рисунок 4.4- Наявні сервіси

4.2 Інсталяція та конфігурація Solr

Для нормальної роботи Solr була спочатку встановлена актуальна версія Java за допомогою такої команди (рис. 4.5)

```
sudo apt-get -y install openjdk-7-jdk
```

Рисунок 4.5- Інсталяція Java

Для початку, потрібно завантажити і розпакувати всі необхідні для роботи файли (рис. 4.6)

```
cd /opt
wget http://archive.apache.org/dist/lucene/solr/4.7.2/solr-4.7.2.tgz
tar -xvf solr-4.7.2.tgz
cp -R solr-4.7.2/example /opt/solr
cd /opt/solr
java -jar start.jar
```

Рисунок 4.6- Інсталяція Solr

Далі зайти за адресою http: // YOUR_IP: 8983 / solr, щоб переконатися, що все працює нормально. Після перевірки повернутися в свою SSH консоль і закрити програму за допомогою комбінації клавіш Ctrl + C.

Потім, необхідно створити користувача Solr і забезпечити йому необхідними правами доступу (рис. 4.7)

```
sudo useradd -d /opt/solr -s /sbin/false solr
sudo chown solr:solr -R /opt/solr
```

Рисунок 4.7- Створення користувача для Solr

Після цього, потрібно скачати скрипт запуска та зробити щоб він стартував автоматично при завантаженні системи (рис. 4.8)

```
sudo wget -O /etc/init.d/jetty http://dev.eclipse.org/svnroot/rt/org.eclipse.jetty/jetty/t
sudo chmod a+x /etc/init.d/jetty
sudo update-rc.d jetty defaults
```

Рисунок 4.8- Устновка скрипту запуска Solr

Далі потрібно налаштувати нашу створену заздалегідь схему та заімпортувати її у пошукову систему.

Першим кроком потрібно створити нову колекцію у директорії Solr. Потім, видалити директорію data і змінити schema.xml на таку, яка нам потрібна та відповідає вимогам системи (рис. 4.9)

```
rm -R data
nano conf/schema.xml
```

Рисунок 4.9 – Зміна схеми колекції Solr

4.3 Керівництво користувача серверної частини

Перш за все потрібно налаштувати фаервол на машині з сервером для доступу.

UFW, або Uncomplicated Firewall («нехитрий фаєрвол») – це інтерфейс IPTables. Основна мета цього зручного у використанні інтерфейсу – істотно спростити управління фаєрволом. Він популярний серед користувачів Linux і навіть встановлений на багато дистрибутивів за замовчуванням. Його установку можна упустити, він вже встановлений та працює.

Нам потрібно дозволити деякі з'єднання до нашого серверу. Синтаксис дуже простий. Наприклад дозволити усі TCP з’єднання для одного із портів виконується такою командою: `sudo ufw allow 22/tcp`

Також можна і для окремих діапазонів портів або адрес.

Деплой та запуск серверу дуже простий і виконується стартом сервісу Linux. Трохи складніше буде з конфігурацією nginx. Він має можливості

віртуального хостингу та балансувальник навантаження і потрібен як простий фронтенд сервер.

Налаштування Nginx сильно відрізняються тут є тільки один основний файл конфігурації і файли для віртуальних хостів. Налаштування з кожної директорії, як в Apache не підтримується:

– /etc/nginx/nginx.conf – головний файл конфігурації;
– /etc/nginx/sites-available/ * – файли конфігурації для віртуальних хостів, простіше кажучи для кожного сайту;
– /etc/nginx/sites-enabled/ * – файли конфігурації активованих сайтів.

Для його налаштування вже є конфіг, який потрібно трохи підредагувати під свої вимоги (рис. 4.10).

```
worker_processes  1;

events {
    worker_connections  1024;
}

http {
  include       mime.types;
  default_type  application/octet-stream;

  sendfile        on;
  keepalive_timeout  65;

  proxy_buffering    off;
  proxy_set_header   X-Real-IP $remote_addr;
  proxy_set_header   X-Forwarded-Proto $scheme;
  proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header   Host $http_host;
  proxy_http_version 1.1;

  upstream my-backend {
     server 127.0.0.1:9000;
  }

  server {
    listen       80;
    server_name www.mysite.com;
    location / {
       proxy_pass http://my-backend;
    }
  }
```

Рисунок 4.10 – Проста кофігурація Nginx

ВИСНОВКИ

В результаті роботи над магістерським дипломним проектом, була розроблена система виявлення плагіату в наукових та студентських роботах, що представляє собою готову Web-систему та сервіс для можливості використання цієї системи іншими користувачами.

Була реалізована можливість швидкого пошуку за локальною базою даних, індексація документів, вигрузка документів. При цьому був протестований пошук на великій базі даних (більше 10 тисяч документів).

У разі проведеного, в ході дипломної роботи, проектування системи виявлення плагіату був підібраний оптимальний алгоритм, обрана платформа повнотекстового пошуку та написаний сервіс для зовнішніх кліентів.

У ході написання дипломногої роботи сервіс зазнавав частих змін алгоритму та взаемодії кліента і сервера.

ПЕРЕЛІК ПОСИЛАНЬ

1. Відеоуроки з створення Інтернет-систем [Електроний ресурс] – Режим доступу: http://www.ruseller.com/
2. Сайт компанії Quetext [Електроний ресурс] – Режим доступу : http://www.quetext.com.
3. Сайт компанії Text.ru [Електроний ресурс] – Режим доступу : http://www.text.ru.
4. Сайт компанії PR.CY [Електроний ресурс] – Режим доступу : http://pr.cy.com.
5. Сайт компанії Plagiarisma [Електроний ресурс] – Режим доступу : http://plagiarisma.net.
6. Федорчук А. Як створюються Web-системи. – СПб.: Питер, 2000. – 224с.
7. Тереза Нейл, Білл Скотт. Проектування веб-інтерфейсів. – СПб.: СимволбуПлюс, 2010. – 352 с
8. Каба М. MySQL і Perl. – СПб.: Питер, 2001. – 288 с.
9. А. Рубен, А. Горєв, С. Макшаріпу. Ефективна робота з СУБД. – СПб.: Питер, 2009. – 822 с.
10. Карпова Т.С. Бази даних: моделі, розробка, реалізація. – СПб.: Питер, 2001. – 304 с.
11. Кей С. Хорстманн. Функціональне програмування. SCALA для нетерплячих 2010. – 407 с.
12. М. Одерскі. Програмування на Scala – Artima, 2013 – 325 с.
13. M. Fowler. Patterns of Enterprise Application Architecture / M.Fowler,D.Rice, M.Foemmel – Addison Wesley, 2002 – 560 с.
14. Дейт К. Дж. Введение в системы баз данных — 8-е изд. / Дейт К.Дж. : «Вильямс», 2006. — 1328 ст