Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка системи пошуку спільних ідей та інтересів»

Студента 2м курсу, 6м групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Гернець Максим Андрійович

Науковий керівник доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Пашорін Валерій Іванович

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Криворучко Олена Володимирівна

КИЇВ - 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студентові

Гернець Максим Андрійович
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Розробка системи пошуку спільних ідей та інтересів»

Затверджена наказом ректора від "13" грудня 2019 р. № 4304

2. Строк здачі студентом закінченої проекту 01 грудня 2020

3. Цільова установка та вихідні дані до проекту

Мета проекту розробити систему пошуку спільних ідей та інтересів

Об'єкт дослідження процес розробки пошуковою системи

Предмет дослідження розробка пошуковою системи для пошуку спільних ідей та інтересів

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз існуючих пошукових систем

1.2. Аналіз роботи пошукової системи

1.3. Висновок до розділу 1

РОЗДІЛ 2. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЕКТУВАННЯ

2.1. Мова програмування C#

2.2. Платформа .Net

2.3. Обґрунтування вибору Blazor

2.4. Обґрунтування вибору ОС Windows

2.5. Середовище розробки Visual Studio 2019

2.6. Висновок до розділу 2

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПОШУКОВОЇ СИСТЕМИ

3.1. Перелік програмних модулів

3.2. Розробка серверної частини

3.3 Інтерфейс та функціонал клієнтської частини

3.4. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проект*	*20.09.2019*	*20.09.2019*
2.	*Розробка та затвердження завдання на проект магістра*	*08.11.2019*	*08.11.2019*
3.	*Вступ та перелік літературних джерел*	*24.01.2020*	*24.01.2020*
4.	*Наукова стаття*	*01.09.2020*	*01.09.2020*
5.	*Технічне завдання*	*28.02.2020*	*28.02.2020*
6.	*Розділ 1. Аналіз предметної області застосування*	*25.06.2020*	*25.06.2020*
7.	*Розділ 2. Вибір програмних засобів та проектування*	*07.09.2020*	*07.09.2020*
8.	*Розділ 3. Реалізація пошукової системи*	*19.10.2020*	*19.10.2020*
9.	*Програма та методика тестування*	*21.10.2020*	*21.10.2020*
10.	*Керівництво користувача*	*23.10.2020*	*23.10.2020*
11.	*Висновки та пропозиції*	*30.10.2020*	*30.10.2020*
12.	*Здача випускного кваліфікаційного проекту на кафедру (перша перевірка)*	*03.11.2020*	*05.11.2020*
13.	*Підготовка автореферату та презентації доповіді*	*03.11.2020*	*05.11.2020*
14.	*Попередній захист випускного кваліфікаційного проекту*	*25.11.2020-27.11.2020*	*25.11.2020-27.11.2020*
15.	*Зовнішнє рецензування випускного кваліфікаційного проекту*	*01.12.2020*	*01.12.2020*
16.	*Підготовка до публічного захисту випускного кваліфікаційного проекту*	*8.12.2020-9.12.2020*	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту Пашорін В.І.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О.В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Гернець М.А.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Науковий керівник випускного кваліфікаційного проекту

_____

(підпис, дата)

Відмітка про попередній захист _____ Пашорін В.І.

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента_____ Гернець М.А.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____ Криворучко О. В.

(прізвище, ініціали, підпис)

Завідувач кафедри _____ Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена розробці системи пошуку спільних ідей та інтересів, для автомазації цього процесу та структоризації інформації.

В результаті аналізу сучасних пошукових систем визначено визначено вагомі недоліки подібних пошукових систем, що були прийняті до уваги та усунені у нашій розробці.

Розробка серверної частини виконана у середовищі розробки IDE Visual Studio 2019, на обраній мові програмування C#, а клієнтської – в браузері. Збирання проектів виконувалося за допомогою вбудованного інструмента автоматизації у Visual Studio 2019.

Готовий програмний комплекс SEARCHPSYSTEM було успішно протестовано відповідно до функціональних вимог.

Ключові слова: Пошукова система, інформація.

ABSTRACT

In accordance with the purpose of the study, the work is devoted to the development of a system for finding common ideas and interests, to automate this process and structuring information.

As a result of the analysis of modern search engines, significant shortcomings of such search engines have been identified, which have been taken into account and eliminated in our development.

The development of the server part is performed in the development environment IDE Visual Studio 2019, in the selected programming language C #, and the client - in the browser. Project collection was performed using the built-in automation tool in Visual Studio 2019.

The finished SEARCHPLUSPLUS software package has been successfully tested in accordance with the functional requirements.

Keywords: Search engine, information.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

ОС – операційна система

ПШ – пошукова система

IDE (англ. Integrated Development Environment) – інтегроване середовище розробки

SDK (англ. Software Development Kit) – набір із засобів розробки, утиліт і документації до певної технології або платформи

CLR (англ. Common Language Runtime) - виконуюча середовище для байт-коду CIL, в який компілюються програми, написані на .NET-сумісних мовах програмування

WEB або WWW (англ. World Wide Web) - розподілена система, що надає доступ до пов'язаних між собою документів, розташованим на різних комп'ютерах, підключених до мережі Інтернет.

					КНТЕУ 121 06М-06.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*19.10.20*	Розробка системи пошуку спільних ідей та інтересів	*Стадія*	*Арку*	*Аркушів*
Керівник		Пашорін В.І.		*19.10.20*		*ПС*	*2*	*37*
Гарант		Криворучко О.В.		*19.10.20*		*Факультет інформаційних технологій 2м курс, 6М група*		
Розробив		Гернець М.А.		*19.10.20*	*Перелік умовних скорочень*			

ЗМІСТ

ВСТУП

Актуальність. Перші пошукові системи з'явилися в мережі Інтернет більше десяти років тому. Тоді вони виконували лише одну функцію - пошуку посилань до недавно створеним сторінок.

На початковому етапі розвитку інтернету, число користувачів мережі було невелике і кількість інформації відносно невеликим. У переважній більшості випадків користувачами Інтернет були співробітники різних університетів або наукових організацій. У той час пошук потрібної інформації в мережі був не настільки актуальне, як тепер. Сьогодні ж пошукові системи перетворилися на багатофункціональний сервіс. Вони дозволяють користувачам знаходити в мережі Інтернет найрізноманітнішу інформацію, завдяки чому користуються величезним успіхом.

Мета дослідження: з’ясувати як працює сучасна пошукова система та створити свою пошукову систему.

Об’єкт дослідження: процес розробки системи пошуку.

Предмет дослідження: розробка системи для пошуку спільних ідей та інтересів.

У відповідності з метою дослідження поставлені наступні *завдання*:

1. Проаналізувати сучасну пошукову систему.
2. Спроектувати свою пошукову систему.
3. Реалізувати пошукову систему.

Наукова новизна дослідження полягає в удосконалення.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз існуючих пошукових систем

Всі ми вже давно звикли блукати по безмежному інтернету, а допомагають в цьому пошукові системи. На території пост радянського простору популярними пошуковими системами на сьогоднішній день є Google і Яндекс, крім них деякі використовують Bing, Yahoo, при цьому навіть ніхто і не замислюється про те, що існують і інші пошукові системи. У кожній країні вони різні, а користувачі вибирають зручний для себе варіант. До того ж є і «підпільні» пошуковики, через них можна знайти інформацію, яку не видасть Яндекс і Google.

В кожному регіоні і навіть країні, є відомі всім сервіси для інтернет пошуку потрібної інформації, а також свої особисті сайти, які використовують тільки в межах цих країн. Загальний графік часткою пошукових систем світу згідно з даними gs.statcounter.com виглядає так:

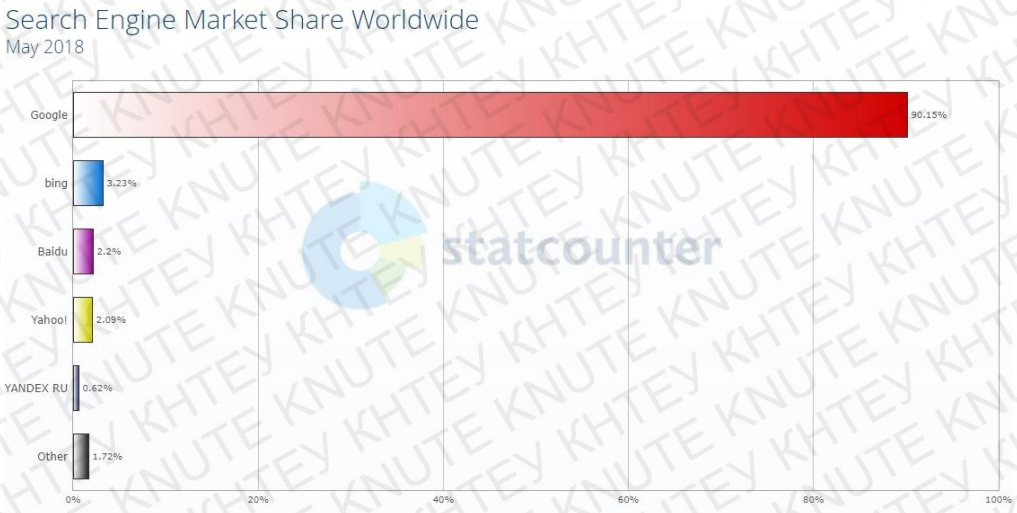

Рис. 1.1. Статистика користування пошукових систем

					КНТЕУ 121 06М-06.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		25.06.20	Розробка системи пошуку спільних ідей та інтересів	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Пашорін В.І.		25.06.20		*Р1*	*6*	*37*
Гарант		Криворучко О.В.		25.06.20		Факультет інформаційних технологій 2м курс, 6М група		
Розробив		Гернець М.А.		25.06.20	*Аналіз предметної області*			

Як бачимо, є беззаперечним лідером Google, однак існують і інші світові лідери. Які пошукові системи і їх світова частка відносяться до "Other» (інші) - дивимося в таблиці 1.1:

Таблиця 1.1 Світові пошукові системи

Пошукова система	Доля, %
Google (google.com)	90,15
Bing (bing.com)	3,23
Baidu (baidu.com)	2,2
Yahoo! (yahoo.com)	2,09
Yandex (yandex.ru)	0,80
Other	0,06

Google, Bing і ще деякі пошуковики використовуються по всьому світу. Існують країни в яких на першому місці своя пошукова система, наприклад Росія і Китай, як видно на карті. Детальніше статистику цих країн і їх пошукових систем розглянемо нижче.

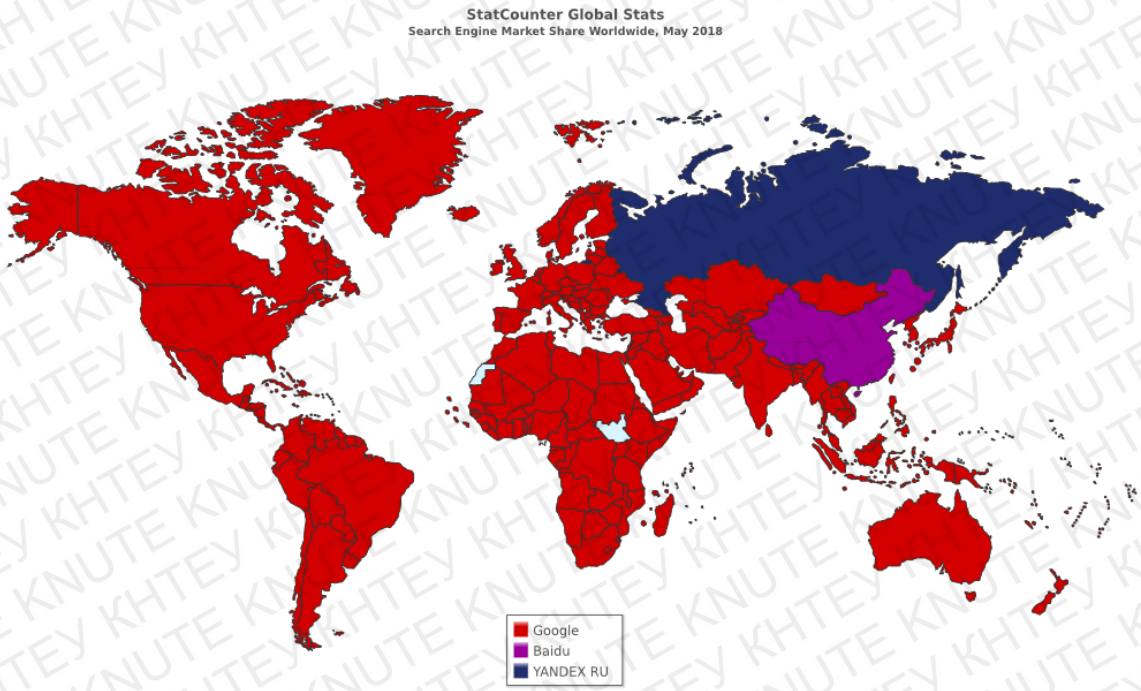

Рис. 1.2. Глобальна статистика по пошуковим системах

Більшість користувачів інтернету України віддають перевагу Google. Раніше використовувався Яндекс, але в зв'язку з політикою, ця пошукова система в Україні заборонена. Незважаючи на це деякі користуються нею, заходячи через VPN сервіси.

Більшість російських інтернет-користувачів віддають перевагу Яндексу 50,76%, Google 45,18%, Mail.ru 2,61%, Bing 0,61%.

Інтернет-користувачі Республики Беларусь отдают предпочтение поисковым системам, Google 67,88%, Яндекс 26,36%, Mail.ru 4,66%, bing 0,44%, Yahoo! 0,34%:

Пошукова система Bing є одним з дітищ корпорації Microsoft, займає 3-е місце в США з 5,59%, в минулому було до 10%. Варто відзначити, що на Світовому ринку дана система займає почесне третє місце в Світовий статистикою. Хоча відносно березня минулого року популярність Bing впала з 3,23%, до 2,39% в травні цього року.

Дана пошукова система має невелику популярність в європейських і азіатських країнах (наприклад Японії 3,05%, Китай 2,45%, де він не дуже відстає від Google). В Україні використовують 0,58% Bing, в Білорусі 0,29%, в Казахстані 0,29%, в Росії всього 0,47% користувачів Всесвітньої павутини. У Саудівській Аравії даної пошуковою системою користуються 1,28%.

Китайські користувачі інтернет мережі найчастіше роблять пошук потрібної інформації через Baidu.com (74,51%). На другому місці в їхньому списку Shenma (11,25%), на третьому, зі своїми 7,3% коштує пошуковик Haosou, на четвертому Sogou 3.76% і тільки п'яту позицію в Китаї займає пошукова система Google (1,68%). Далі йдуть Bing, Yahoo, Yandex.

Пошукова система Yahoo! своєю появою також зобов'язана двом студентам Стенфорда. Джеррі Янг і Девід Файло спочатку створили цю пошукову мережу, щоб трохи розважитися, а в підсумку вони отримали прибуткову компанію. За весь час існування, починаючи з 1994 року,

компанія переживала, як злети, так і падіння. Але і сьогодні їй вдається триматися в списку лідерів і навіть рости.

Системою Yahoo! активно користуються жителі Японії - цілих 20,91%. В Україні 0,53%, Китай - 0,03% населення.У Білорусі 0,18%, а в Росії користуються всього 0,37%.

1.2. Аналіз роботи пошукової системи

Проаналізуємо роботу пошукової системи над самою популярною пошукової системи в світі від Google.

Обробка сайту в Google включає три основних етапи.

Сканування. Google переглядає контент в Інтернеті за допомогою автоматизованих програм - роботів, які шукають нові або оновлені веб-сторінки. Адреси таких сторінок (URL) додаються в перелік для подальшого аналізу. Ми знаходимо сторінки безліччю різних способів, але основним є перехід по посиланнях зі сторінок, які вже зареєстровані в Google.

Індексування. Роботи Google відвідують сторінки, знайдені під час сканування, і намагаються визначити їх тематику. Для цього аналізується їх вміст, включаючи зображення і відеофайли. Отримана інформація зберігається в індексі Google - об'ємної базі даних, розміщеної на безлічі комп'ютерів.

Показ результатів пошуку. Коли користувач вводить в Google пошуковий запит, наша система підбирає найбільш підходящі результати, враховуючи місце розташування, мова, тип пристрою (комп'ютер або смартфон) і колишні запити користувача. Наприклад, результати за запитом "ремонт велосипедів" будуть відрізнятися в залежності від того, перебуваєте ви в Парижі або в Гонконзі.

Щоб надавати користувачам швидкі відповіді, архітектуру пошуку було розділено на 2 частини.

Базовий пошук - програма, яка здійснює пошук по своїй частині індексу і надає всі відповідні запиту документи.

Метапошук - програма, яка обробляє запит, визначає регіональність користувача, і якщо запит популярний, то видає вже готовий варіант видачі, а якщо запит новий, то вибирає базовий пошук і віддає команду на підбір документів, далі методом машинного навчання ранжує знайдені документи і надає користувачеві.

Існує чотири типи пошукових систем: з пошуковими роботами, керовані людиною, гібридні і мета-системи.

Системи, що використовують пошукові роботи. Складаються з трьох частин: краулер («бот», «робот» або «павук»), індекс і програмне забезпечення пошукової системи. Краулер потрібен для обходу мережі і створення списків веб-сторінок. Індекс - великий архів копій веб-сторінок. Мета програмного забезпечення - оцінювати результати пошуку. Завдяки тому, що пошуковий робот в цьому механізмі постійно досліджує мережу, інформація більшою мірою актуальна. Більшість сучасних пошукових систем є системами даного типу.

Системи, керовані людиною (каталоги ресурсів). Ці пошукові системи одержують списки веб-сторінок. Каталог містить адресу, заголовок і короткий опис сайту. Каталог ресурсів шукає результати тільки з описів сторінки, представлених йому веб-майстрами. Гідність каталогів в тому, що всі ресурси перевіряються вручну, отже, і якість контенту буде краще в порівнянні з результатами, отриманими системою першого типу автоматично. Але є і недолік - оновлення даних каталогів виконується вручну і може істотно відставати від реального стану справ. Ранжування сторінок не може миттєво змінюватися. Як приклади таких систем можна привести каталог Yahoo, dmoz і Galaxy.

Гібридні системи. Такі пошукові системи, як Yahoo, Google, MSN, поєднують в собі функції систем, що використовують пошукові роботи, і систем, керованих людиною. мета-системи.

Метапоісковая системи об'єднують і ранжирують результати відразу декількох пошукових систем. Ці пошукові системи були корисні, коли у кожної пошукової системи був унікальний індекс, і пошукові системи були менш «розумними». Оскільки зараз пошук набагато покращився, потреба в них зменшилася.

Пошуковий алгоритм - це сукупність правил, відповідно до яких пошукові системи оцінюють релевантність сторінок запитам і будують пошукову видачу.

Пошукові алгоритми за принципом дії можна розділити на дві групи.

Алгоритми прямої дії. Щоб відповісти на користувальницький запит, алгоритми прямої дії перебирають всі документи, що зберігаються в індексі пошукової системи. Такі алгоритми формують максимально релевантну видачу, але вимагають великих обчислювальних ресурсів і дуже повільно працюють. На даний момент великі пошукові системи їх не використовують, або використовують вкрай обмежено.

Алгоритми інвертованого (зворотного) індексу. Зворотний (інвертований) індекс - це структура даних, де для кожного терміна перераховані посилання на документи, які його містять. пошуку по зворотному індексу звертається до такої бази і відразу отримує повний список релевантних документів. Залишається лише впорядкувати їх. Релевантність пошукової видачі, створеної за допомогою алгоритмів зворотного індексу, трохи нижче, але її формування вимагає набагато менше ресурсів. В сучасних пошукових системах використовується саме цей спосіб.

Ранжування сайтів - це процес вибудовування пошуковою системою певного порядку сайтів на сторінці пошукової видачі у відповідь на запит користувача.

Алгоритми ранжирування всіх комерційних пошукових систем засекречені. Веб-майстрам невідомий ні повний список факторів, які враховують пошукові алгоритми, ні роль кожного з них у формулі. Крім того, і формула, і список факторів можуть змінюватися з часом і бути різними для різних колекцій документів. Разом з тим, завдяки деяким повідомленнями представників ПС, аналізу видачі, проведення експериментів і багаторічній практиці просування різних сайтів, оптимізатори змогли скласти список факторів, які точно враховуються алгоритмами. Їх можна об'єднати в кілька груп:

- доменні - вік домену, його рівень, входження ключового запиту в ім'я, історія;
- комерційні - наявність контактних даних, інформації про доставку та оплату, прайс-листа;
- текстові - вміст метатегов, обсяг тексту на сторінці, входження ключових фраз, унікальність, якість тексту і інші чинники;
- поведінкові - середня тривалість візиту, кількість переглядів за сесію, відсоток повернень, відсоток відмов, CTR сниппета у видачі та інші метрики, які описують поведінку користувача на сайті;
- посилальні - кількість, вік, вага і анкор зовнішніх посилань.

1.3. Висновки до розділу 1

Було проаналізовано існуючі пошукові системи та обрана за основу сама популярна пошукова система – це google. Також було розглянуто роботу пошукової системи.

РОЗДІЛ 2
ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЕКТУВАННЯ СЕРВІСУ

2.1. Мова програмування C#

Мова програмування C# був створений в кінці 1990-х років і став частиною загальної .NET-стратегії Microsoft. Вперше він побачив світло в якості б-версії в середині 2000 року. Головним архітектором C# був Андерс Хейлсберг (Anders Hejlsberg) - один з провідних фахівців в області мов програмування, що отримав визнання у всьому світі. Досить сказати, що в 1980-х він був автором вельми успішного продукту Turbo Pascal, витончена реалізація якого встановила стандарт для всіх майбутніх компіляторів. C# безпосередньо пов'язаний з C, C++ і Java.

Дана мова використовує об'єктно-орієнтований підхід до програмування у всьому. Це означає, що тобі потрібно буде описувати абстрактні конструкції на основі предметної області, а потім реалізовувати між ними взаємодія. Даний підхід користується великою популярністю, тому що дозволяє не тримати в голові всю інформацію, а працювати за принципом чорного ящика: подав вхідні дані, отримав потрібні дані.

Ще варто згадати, що все це працює на базі платформи .NET Framework. Що це означає? Для багатьох непосвячених, це просто якась приблуда, яку потрібно встановити на комп, щоб програма запустилася, але справа йде значно глибше. Написаний тобою код на мові C# транслюється в проміжний мова (IL), який в свою чергу вже перетворюється в машинний код на твоєму комп'ютері прямо під час виконання програми (JIT).

Особливості C# це вбудована підтримка автоматичної генерації XML-документації. Автоматичне звільнення динамічно розподіленої пам'яті. Можливість позначки класів і методів атрибутами, обумовленими користувачем, (це може бути корисно при документуванні та здатне впливати на процес компіляції -наприклад, можна помітити методи, які повинні компілюватися тільки в отладочном режимі) .Полная доступ до бібліотеки базових класів .NET, а також легкий доступ до WindowsAPI (якщо це дійсно необхідно) .Указателі і прямий доступ до пам'яті, якщо вони необхідні (проте мова розроблений таким чином, що в переважній більшості випадків можна обійтися і без цього). Підтримка властивостей і подій в стилі VisualBasic.Простое зміна ключів компіляції. Дозволяє отримувати виконувані файли або бібліотеки компонентів .NET, які можуть бути викликані іншим кодом так само, як елементи управління ActiveX (компоненти COM).

Можливість використання C # для написання динамічних web-сторінок ASP.NET. Однією з областей, для яких не призначений цю мову, є критичні за часом і високопродуктивні програми, коли має значення, займати на виконання циклу 1000 або 1050 машинних циклів, і звільняти ресурси потрібно негайно. C ++ залишається в цій області найкращим з мов високого рівня. У C # відсутні деякі ключові моменти, необхідні для створення високопродуктивних додатків, зокрема підставляються функції і деструктори, виконання яких гарантується в певних точках коду.

C#, будучи останнім з широко поширених мов програмування, повинен увібрати в себе весь наявний досвід і увібрати найкращі сторони існуючих мов програмування, при цьому будучи спеціально створеним для роботи в .NET. Сама архітектура .NET продиктувала йому (каки багатьох інших мов, на яких можна писати під .NET) об'єктно-орієнтовану спрямованість. Звичайно, це не є правилом, можливе створення

компіляторів навіть функціональних мов по .NET, на цю тему існують спеціальні работи.Свой синтаксис C# в чому успадкував від C і Java. Розробники, які мають досвід створення програмного забезпечення на цих мовах, знайдуть в C# багато знайомих рис. Але разом з тим він є багато в чому новаторським -аттрібути, делегати та події, прекрасно вписані в загальну ідеологію мови, міцно зайняли місце в серцях .NET розробника. Їх введення дозволило застосовувати принципово нові прийоми програмування.

Деякі найбільш часто зустрічаються типи додатків:

Додатки Windows. Це програми на кшталт MicrosoftOffice, мають знайомий "Windows-подібний" вид і уявлення. Створювати такі додатки досить просто за допомогою модуля .NETFramework, який називається WindowsFormsi є бібліотекою керуючих елементів (кнопок, панелей інструментів, меню і т. П.); ця бібліотека може використовуватися для створення призначеного для користувача інтерфейсу (userinterface, Ul) Windows.Web-додатки. Ці додатки є web-сторінки, які можуть переглядатися будь-яким web-браузером. До складу .NET Framework входит потужна система динамічного створення вмісту web-сторінок, що дозволяє ідентифікувати користувача, забезпечувати безпеку та ін. Ця система називається ActiveServerPages.NET (ASP.NET-активні серверні сторінки .NET); для створення додатків ASP.NET можно застосовувати WebForms мови C#.

Web-служби. Це новий чудовий спосіб створення гнучких розподілених додатків. За допомогою web-служб можна обмінюватися по Інтернету практично будь-якими даними з використанням єдиного простого синтаксису незалежно від того, яку мову програмування застосовувався при створенні web-служби і на який системи вона розміщена. Додатків всіх перерахованих типів може знадобитися доступ до баз даних, що

здійснюється за допомогою розділу .NETFramework, званого ActiveDataObjects.NET (ADO.NET-активні об'єкти з даними .NET). Також можна використовувати і багато інших ресурсів, наприклад, інструменти для створення 15сетевих компонентів, графічного виведення, виконання складних математичних обчислень і т.д.

2.2. Платформа .Net

.NET - програмна платформа, випущена компанією Microsoft в 2002 році. Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), яка підходить для різних мов програмування. Функціональні можливості CLR доступні в будь-яких мовах програмування, що використовують цю середу. Вважається, що платформа .NET Framework стала відповіддю компанії Microsoft на набрала на той час велику популярність платформу Java компанії Sun Microsystems (нині належить Oracle).

Компанія Microsoft випустила .NET 5 в 2020 році, яка об'єднує .NET Core, .NET Framework, Xamarin і Mono, що дозволяє використовувати єдину кодову базу рішень для всіх платформ, включаючи Android і iOS.

Продукт .NET 5 продовжив розвиток відкритого проекту .NET Core 3.0 і прийшов на зміну класичному .NET Framework, який окремо більше розвиватися не буде і зупиниться на випуску .NET Framework 4.8. Вся пов'язана з платформою .NET розробка тепер зосереджена навколо проекту .NET Core, включаючи Runtime, JIT, AOT, GC, BCL (Base Class Library), C#, VB.NET, F#, ASP.NET, Entity Framework, ML.NET, WinForms, WPF і Xamarin. У наступному випуску .NET 6 до складу будуть включені напрацювання проектів Xamarin і Mono, які дозволять забезпечити підтримку платформ iOS і Android.

Як і в .NET Core в .NET 5 поставляється runtime CoreCLR з JIT-компілятором RyuJIT, стандартні бібліотеки, бібліотеки CoreFX, WPF, Windows Forms, WinUI, Entity Framework, інтерфейс командного рядка dotnet, фреймворки для розробки клієнтських додатків WPF і Windows Forms, а також інструменти для розробки мікросервісов, бібліотек, серверних, графічних і консольних додатків.

Крім JIT-компіляції в новому випуску надано заснований на напрацюваннях LLVM режим попередньої компіляції в машинний код і байткод WebAssembly (для статичної використані Mono AOT і Blazor). Істотно збільшена продуктивність різних компонентів платформи і бібліотек (особливо відзначається прискорення операцій сериализации JSON, регулярних виразів і HttpClient). Підвищено чуйність за рахунок модернізації збирача сміття. Вбудований клієнт ClickOnce для швидкої публікації додатків. Для Linux і macOS адаптований API System.DirectoryServices.Protocols для роботи з LDAP і Active Directory. Для Linux також додана підтримка однофайлових додатків, в яких всі компоненти і залежно упаковані в одному файлі.

2.3. Обґрунтування вибору Blazor

Blazor – це сучасний фреймворк, для створення веб-сервісів та веб-сторінок, випущена компанією Microsoft в 2018 році. Blazor написаний на .NET і підготовлений до запуску за допомогою WebAssembly. Він дає вам всі переваги багатих сучасних (SPA), дозволяючи при цьому використовувати .NET від початку і до кінця, аж до загального коду на сервері і клієнті.

Компоненти Blazor можуть бути розміщені різними способами для створення вашого веб-додатки. Перший підтримуваний спосіб називається Blazor Server. У додатку Blazor Server компоненти запускаються на сервері

з використанням .NET Core. Всі взаємодії та оновлення призначеного для користувача інтерфейсу обробляються за допомогою з'єднання WebSocket в реальному часі з браузером. Додатки Blazor Server швидко завантажуються і прості в реалізації. Підтримка Blazor Server доступна в .NET Core 3.1 LTS.

Blazor WebAssembly поставляється з великою кількістю функцій, які допоможуть вам продуктивно працювати в наступному проекті веб-додатки:

- Використовуйте продуктивність C# і строгу типізацію під час виконання.
- Побудуйте стабільну і зрілу екосистему .NET. Легко повторно використовуйте код і існуючі бібліотеки .NET Standard на клієнті і сервері.
- Модель загальних компонентів з додатками Blazor Server. Розгорніть свій додаток як самостійний статичний сайт або розміщений на ASP.NET Core.
- Створення прогресивних веб-додатків (PWA) з автономними можливостями і вбудованою інтеграцією з ОС.
- Вбудована підтримка аутентифікації.
- Інтегрована підтримка глобалізації та локалізації.
- Конфігурація на основі середовища.
- Обрізка IL і попередня компресія в процесі побудови.
- Налагодження фул-скл.
- Відмінна взаємодія з Visual Studio, Visual Studio для Mac і Visual Studio Code.

На даний момент WebAssembly підтримується всіма основними браузерами, в тому числі і мобільними. Це компактний байткод-формат, оптимізований для зменшення обсягу завантажуваних даних і прискорення виконання. Незважаючи на те, що багато розробників могли б так подумати,

WebAssembly не привносить ніяких нових проблем безпеки, так як це не звичайні бінарні файли (на кшталт x86 / x64) - це новий формат, у якому байткод, який може робити тільки те ж саме, що і JavaScript.

2.4. Обґрунтування вибору ОС Windows

Windows 10 - інноваційна ОС від усім відомої компанії Microsoft. Представлена програма була відносно недавно. Сьогодні вона стала особливо популярна серед користувачів. За даними, які надала сама компанія Microsoft, дану ОС вже встановило понад 75 мільйонів чоловік по всьому світу.

На сьогоднішній день, остання операційна система сімейства Windows, являється Window 10 версія 1909, реліз якої відбувся 29 липня 2015 року.

Розробники щиро постаралися врахувати всі побажання користувачів і створити єдину платформу для всіх пристроїв. Для настільних комп'ютерів, планшетів, смартфонів і т.д. Тепер Windows 10 доступна для комп'ютерів, смартфонів, планшетів і т.д. Варто відзначити, що дана версія є останньою у всій серії Windows. На цьому нумерація її закінчується, і тепер будь-яке оновлення буде встановлюватися виключно для певних програм.

Також разом з Windows 10 вийшло DirectX 12 - компонента, що забезпечує взаємодію додатків з відеокартою.

Таким чином, операційна система Windows 10 є цілком вдалим продуктом. У ній враховані побажання користувачів, згладжені недоліки 8-й Windows і з'явилися цікаві нововведення. Велика частина користувачів, які розбираються в комп'ютері і настройках операційної системи, оцінили Windows 10, вирішивши повністю на неї перейти.

2.5. Середовище розробки Visual Studio 2019

Microsoft Visual Studio - лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів. Дані продукти дозволяють розробляти як консольні додатки, так і ігри та програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework і Silverlight.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторінга коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і відладчик машинного рівня. Решта вбудовуються інструменти включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних.

Visual Studio дозволяє писати код на своїй мові або будь-які інші бажані мови, використовуючи весь час один і той же інтерфейс (IDE). Крім того, Visual Studio також дозволяє створювати веб-сторінки на різних мовах, а також розміщувати їх все в одному і тому ж веб-додатку. Єдиним обмеженням є те, що на кожній веб-сторінці можна використовувати лише яким-небудь одним мовою (очевидно, що у протилежному випадку проблема при компіляції була просто не уникнути). використовуючи весь час один і той же інтерфейс (IDE). Крім того, Visual Studio також дозволяє створювати веб-сторінки на різних мовах, а також розміщувати їх все в одному і тому ж веб-додатку. Єдиним обмеженням є те, що на кожній веб-сторінці можна використовувати лише яким-небудь одним мовою

(очевидно, що в протилежному випадку проблема при компіляції була просто не уникнути).

Для створення більшості додатків потрібно пристойну кількість стандартного стереотипного коду, і Web-сторінки ASP. NET тому не виключення. Наприклад, додавання Web-елемента управління, приєднання обробників подій і коригування форматування вимагає установки в розмітці сторінки ряду деталей. У Visual Studio такі деталі встановлюються автоматично.

Інтуїтивний стиль кодування. За замовчуванням Visual Studio форматує код у міру його введення, автоматично вставляючи необхідні відступи і застосовуючи колірне кодування для виділення елементів типу коментарів. Такі незначні відмінності роблять код більш зручним для читання і менш схильним до помилок. Застосовувані Visual Studio автоматично параметри форматування можна навіть налаштовувати, що дуже зручно у випадках, коли розробник вважає за краще інший стиль розміщення дужок (наприклад, стиль K & R, при якому відкриває дужка розміщується на тому самому рядку, що і оголошення, якому вона передує).

Більш висока швидкість розробки. Багато з функціональних можливостей Visual Studio спрямовані на те, щоб допомагати розробнику робити свою роботу якомога швидше. Зручні функції, на зразок функції IntelliSense (яка вміє перехоплювати помилки і пропонувати правильні варіанти), функції пошуку і заміни (яка дозволяє відшукувати ключові слова як в одному файлі, так і в усьому проекті) і функції автоматичного додавання і видалення коментарів (яка може тимчасово приховувати блоки коду), дозволяють розробнику працювати швидко і ефективно.

2.6. Висновки до розділу 2

Отже, для разробки сервісу, була обрана платформа .NET від компанії Microsoft. Серверна частина та клієнська частина, буду реалізована за допомогую фрейворку Blazor. Написання коду буду проводиться в IDE Visual Studio 2019, яка позволяє створити швидко та легко веб-сервіс.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПОШУКОВОЇ СИСТЕМИ
3.1. Перелік програмних модулів

Пошукова система складається з частин, кожен з яких виконує свої завдання, на рисунку 3.1 схематично описують механізм роботи пошукової системи.

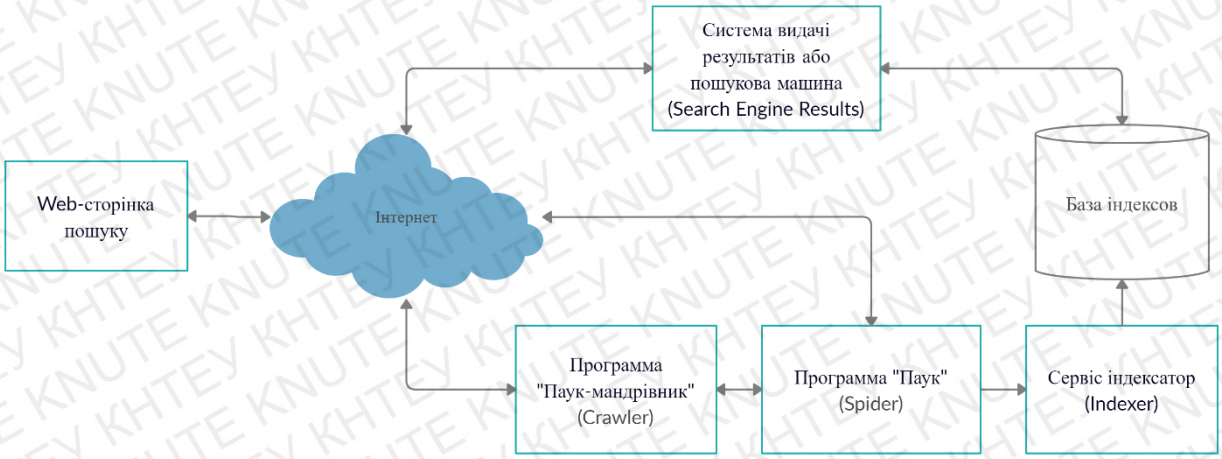

Рис.3.1. Спрощена схема роботи пошукової системи

Web server (веб-сервер). Відповідає за взаємодію пошукової системи з користувачем. Надає зручний і наочний інтерфейс для завдання запитів. Це html сторінка з формою пошуку і візуальної видачею результатів пошуку.

Spider (павук) - це браузероподобная програма, що викачує веб-сторінки. Заповнює базу даних пошукової системи. «Павук» викачує веб-сторінки так само як користувальницький браузер. Відмінність в тому, що браузер відображає міститься на сторінці текстову, графічну або іншу інформацію, а павук працює з html-текстом сторінки безпосередньо, у нього

немає візуальних компонент. Саме, тому потрібно звертати увагу на помилки в html кодах сторінок сайту.

Crawler (краулер, «мандрівний» павук) - це програма, що проходить автоматично по всіх посиланнях, які знайдені на сторінці. Програма виділяє все знаходяться на сторінці посилання. Завдання програми обчислити, куди повинен далі попрямувати павук, виходячи із заданого заздалегідь, адресного списку або йти по посиланнях на сторінці. Краулер «бачить» і слід по всіх посиланнях, знайденим на сторінці і шукає нові документи, які пошукова система, поки ще не знає. Саме, тому, потрібно видаляти або виправляти биті посилання на сторінок сайту і стежити за якістю посилань сайту.

Indexer (індексатор) - це програма, що аналізує веб-сторінки, завантажені павуками. Аналіз сторінок сайту для їх індексації. Програма ділить сторінку на складові частини, далі аналізує кожну частину окремо. Виділенню і аналізу піддаються заголовки, абзаци, текст, спеціальні службові html-теги, стильові та структурні особливості текстів, і інші елементи сторінки. Саме, тому, потрібно виділяти заголовки сторінок і розділів мета тегами (h1-h4, h5, h6), а абзаци укладати в теги <p>.

Basedate (база індексів). База даних пошукових систем зберігає всі завантажені і аналізовані пошуковою системою дані. У базі даних пошукових систем зберігаються всі завантажені сторінки та сторінки, перенесені в пошуковій індекс. У будь-якому інструменті веб майстрів кожного пошукача, ви можете бачити і знайдені сторінки та сторінки в пошуку.

Search engine results engine (система видачі результатів або пошукова машина) - це програма, яка займається вилученням з бази даних проіндексованих сторінок, згідно пошуковим запитом. Саме ця програма вибирає сторінки, що задовольняють запиту користувача, і визначає

порядок їх сортування. Це, як правило, дуже потужний комп'ютер, який обробляє базу індексів відповідно до отриманого запитом.

Пошуковий робот. Комп'ютер, оснащений спеціальною програмою, яка безперервно переглядає весь Інтернет, індексуючи все зустрічаються Web-сторінки, і оновлюючи базу індексів.

Основне завдання нашої пошукової системи - простота установки, мінімальні системні вимоги і висока швидкість роботи. Більшість існуючих пошукових систем вимагають один або кілька виділених серверів, я ж намагався написати пошуковий механізм не вимагає не тільки виділеного сервера, але не вимагає навіть додаткового ПЗ, такого як SQL сервера. Весь індекс пошукової системи зберігається в файлах, які мають складну деревоподібну структуру. Програма складається з двох основних частин - пошуковий робіт (індексатор) і механізм пошуку за індексом. Розглянемо більш докладно функції кожної частини:

Індексатор повинен виконувати наступні дії:

- Пошук файлів заданого типу на диску. (Зазвичай HTML, HTM, STHML, SHTM)
- Виділення з кожного файлу заголовка (в HTML документах заголовок знаходиться між тегами <TITLE> і </ TITLE>)
- Виділення інформації з документів, фільтрація скриптів (теги <SCRIPT>: </ SCRIPT>) , таблиць стилів (<STYLE> .. </ STYLE>), коментарів (<! - ->) і інших службових лексем.
- Збереження інформації про документи в спеціальному файлі, присвоєння документу унікального номера.
- Збереження інформації про кожне слово, і номер документа в індексі.

Механізм пошуку за індексом.

- Механізм повинен шукати в індексному файлі кожне слово з пошукового запиту.

- Повинен сортувати результати пошуку за релевантністю і відповідності.
- В першу чергу виводяться результати строго відповідають запиту (якщо кожне слово запиту існує в знайденому документі).
- Результати сортуються за релевантністю - кількості слів запиту, зустрінутих в знайденому документі.
- Форматування результатів пошуку і розбивка їх на сторінки.

Індексний файл (index): Містить інформацію про слова зустрічаються в документах, і приналежність слова документу. Має деревоподібну структуру, що складається з двох типів записів. Завжди починається з записи ідентифікації слова.

Таблиця 3.1 - запис ідентифікації слова.

Зсув	Розмір	Опис
0	4	Значення хеш функції слова
4	32	Слово (великими літерами)
36	4	Покажчик на наступну структуру ідентифікації слова, значення хеш функції якої менше поточного. Якщо нуль, то кінець гілки дерева.
40	4	Покажчик на наступну структуру ідентифікації слова, значення хеш функції якої більше поточного. Якщо нуль, то кінець гілки дерева.
44	4	Покажчик на структуру ідентифікації документа.

3.2. Розробка серверної частини

При використанні моделі розміщення Blazor Server додаток виконується на сервері з програми ASP.NET Core. Оновлення елементів призначеного для користувача інтерфейсу, обробка подій і виклики JavaScript обробляються через підключення SignalR.

Додаток Blazor Server попередньо рисує у відповідь на перший клієнтський запит, який налаштовує стан призначеного для користувача

інтерфейсу на сервері. Коли клієнт намагається створити підключення SignalR, він повинен повторно підключитися до того ж сервера. Додатки Blazor Server, що використовують кілька внутрішніх серверів, повинні реалізовувати закріплені сеанси для з'єднань SignalR.

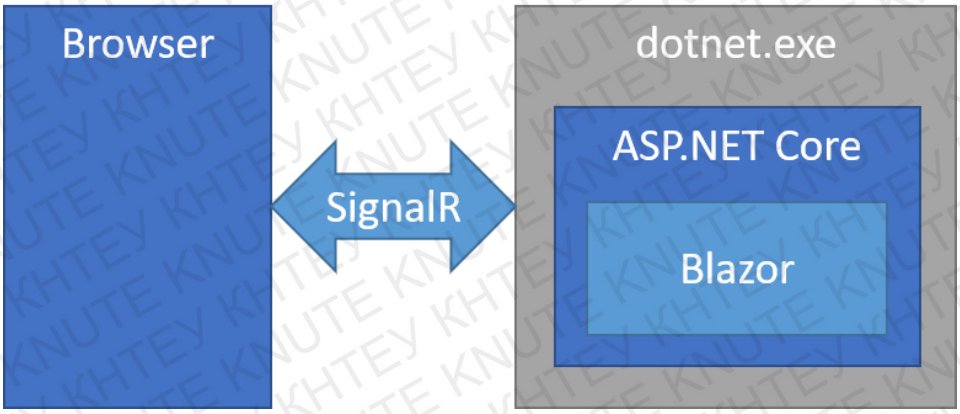

Рис.3.2. Blazor Server

У Blazor Server основна логіка додатка розташовується на стороні сервера. Якщо на боці клієнта присходят якісь події, то за допомогою SignalR клієнт посилає серверу інформацію про проведені дії. Сервер отримує цю інформацію, обробляє її і посилає клієнтові відповідь. Оновлення елементів призначеного для користувача інтерфейсу, обробка подій, виклики JavaScript на стороні клієнта здійснюються за допомогою взаємодії сервера і клієнта через SignalR.

Основні елементи проекту:

Папка wwwroot для зберігання статичних файлів, за замовчуванням зберігає використовувані файли css, зокрема, файли фреймворку bootstrap.

Папка Services зберігає класи C#, які описують Indexer.

Папка Pages містить сторінки Razor Pages, що визначають візуальну частину програми та його логіку, а також компоненти Razor (розташовуються в файлах з розширенням *.razor), які представляють основний зміст сторінки.

_Host.cshtml - головна сторінка (Razor Page) додатки, в рамках якої будуть розгортатися додаток. Counter.razor зберігає код компонента Counter,

суть якого в визначення лічильника, значення якого збільшується при натисканні на кнопку.

Error.razor зберігає код компонента Error, який застосовується для виведення повідомлення про помилку.

FetchData.razor зберігає код компонента FetchData, який за допомогою сервісу WeatherForecastService отримує деякі дані і виводить їх на веб-сторінку Index.razor зберігає код компонента Index.

Папка Shared зберігає додаткові компоненти Razor.

MainLayout.razor зберігає код компонента MainLayout, який визначає структуру або компоновку сторінки.

NavMenu.razor зберігає код компонента NavMenu, який визначає елементи навігації.

_Imports.razor містить підключення просторів імен за допомогою директиви using, які будуть підключатися в компоненти Razor (файли з розширенням .razor).

App.razor містить визначення кореневого компонента додатка, який дозволяє встановити маршрутизацію між вкладеними компонентами за допомогою іншого вбудованого компонента Router.

Файл appsettings.json зберігає конфігурацію програми.

Файл Startup.cs представляє клас Startup, стандартний для додатка ASP.NET Core, де налаштовується конвеєр обробки запиту, впроваджуються сервіси і здійснюється конфігурація додатки. Файл

Program.cs містить клас Program, який представляє точку входу в додаток. В даному випадку це стандартний для додатка ASP.NET Core клас Program, який запускає і конфигурирует хост, в рамках якого розгортається додаток з Blazor.

3.3. Інтерфейс та функціонал клієнтської частини

Користувач, бажаючи знайти необхідну інформацію за допомогою браузера з'єднується з веб-сервером пошукової системи. На основній сторінці, міститься поле, в якому набирається запит, наприклад, одне або кілька ключових слів, а також спеціальних слів і символів. Після того як запит підготовлений, користувачеві залишається віддати команду на пошук Web-сторінок, які відповідали б запиту. Для цього є кнопка Search, Find, Submit і т.д. Сформований запит передається на пошукову машину.

Сторінка результатів пошуку або пошукова - веб-сторінка, що генерується пошуковою системою у відповідь на пошуковий запит користувача. В сторінці результатів пошуку можна виділити кілька областей:

- органічні пошукові результати - основна частина видачі;
- шорткати (one-box'и, чаклунчик і т. п.) - область перед основними результатами пошуку, де може бути поміщений готову відповідь на запит, корисна інформація або посилання, або запропоновано виправлення помилок в запиті;
- пов'язані запити - переформулювання і уточнення введеного запиту, схожі на нього запити;
- поле введення пошукового запиту, можливість автоматичних підказок (автодоповнення);
- посилання для переходу на наступну, попередню і кілька сусідніх сторінок видачі.

При запиті в пошукову систему користувачеві демонструється сторінка зі структурно розташованими посиланнями на сторінки сайтів з необхідною інформацією. Під посиланнями розташовується пара рядків з описом сайту, так званий сниппет, на підставі якого користувач вирішує, перейти йому на

даний сайт чи ні. Посилання будуються відповідно до визначеної ієрархії, складеної пошуковою системою.

Крім посилань на серп можна знайти блоки контекстної і графічної реклами, і наступні сторінки видачі з посиланнями. На серпі можна також знайти шорткати, або швидкі інформативні відповіді на запит користувача, які видаються пошукачем. Різні системи на один і той же запит показують різні шорткати.

Елементи, які відображаються в результі пошуку:

Фавікон - це значок веб-сайту, який відображається не тільки в пошуковій видачі, але і в браузері, при відкритті того чи іншого сайту. Вказати пошуковій системі, який фавікон виводити дуже просто, потрібно всього лише завантажити його на сайт і пошуковики його знайдуть і почнуть використовувати (завантажується зображення в форматі .ico з назвою favicon в корінь сайту; розмір файлу favicon.ico - 16 на 16 пікселів; зробити фавікон можна тут і тут).

Title – тема блоку у видачі пошукові системи, як правило, беруть з заголовка потрапляє в результати пошуку сторінки. Відповідно, щоб поміняти заголовок у видачі, потрібно змінити його сторінці. Іноді заголовок береться не з мета-тега <title>, а безпосередньо з заголовка сторінки, який знаходиться між тегами <h1> </ h1>.;

URL – це домен сайту, на якому знаходиться сторінка, знайдена за ключовою фразою;

Description – це короткий опис вмісту сторінки.

3.4. Висновки до розділу 3

Отже, систематизоване поєднання наведених модулів у клієнтську та серверну частини, і налагодження обміну даними між ними, дозволили створити єдиний програмний комплекс для пошуку потрібної інформації.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

З проведених досліджень можна зробити наступні висновки:

1. Проведено аналіз структури та роботи пошукової системи Серверна частина, на відміну від альтернативних рішень, встановлюється локально і не вимагає доступу до Інтернет.

2. В якості операційних систем обрано ОС Windows, як найзручніші та найрозповсюдженіші ОС на сучасному ринку.

3. Проведений аналіз засобів розробки показав, що найзручнішим для реалізації поставлених завдань є Visual Studio 2019.

4. В процесі виконання магістерської роботи розроблено систему для пошуку спільних ідей та інтересів.

5. Розроблена система повністю задовольняє поставлені функціональні вимоги. Тестування продемонструвало повну відповідність функціональним вимогам та надійну роботу застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Список пошукових систем [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Список_пошукових_систем
2. Як працює пошук Google [Електронний ресурс]. - Режим доступу: https://www.google.com/intl/uk/search/howsearchworks/algorithms/
3. Язык программирования C# и платформа .NET [Електронний ресурс]. - Режим доступу: https://metanit.com/sharp/
4. Введение в Blazor [Електронний ресурс]. - Режим доступу: https://metanit.com/sharp/blazor/1.1.php
5. Как работают поисковые системы [Електронний ресурс]. - Режим доступу: https://habr.com/ru/company/yandex/blog/464375/
6. Как устроены поисковые системы [Електронний ресурс]. - Режим доступу: https://www.sembook.ru/book/poiskovye_sistemy/kak-ustroeny-poiskovye-sistemy/

					КНТЕУ 121 06М-06.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		24.01.20	Розробка системи пошуку спільних ідей та інтересів	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Пашорін В.І.		24.01.20		*СВД*	*32*	*37*
Гарант		Криворучко О.В.		24.01.20		Факультет інформаційних технологій 2м курс, 6М група		
Розробив		Гернець М.А.		24.01.20	*Список використаних джерел*			

ТЕХНІЧНЕ ЗАВДАННЯ

Однією з найголовніших запорук відповідності вимогам та високої якості розробленого програмного рішення є грамотно спроектоване до початку розробки технічне завдання. Отже, далі розкриємо невід'ємні положення технічного завдання до розробки сервісу для пошуку спільних ідей та інтересів.

1. Загальні відомості

Назва проекту: SearchSystem.

Планові строки розробки: серпень 2019 – вересень 2019.

Потенційні користувачі: користувачі стаціонарних ПК зі встановленим браузером.

Призначення програмного рішення: надати можливість користувачу ПК шукати спільні ідеї та інтереси.

Мета створення програмного рішення: опанування сучасних технологій розробки пошукових систем.

2. Структура проекту

Серверна частина виконуватиметься на ПК, оскільки він надає доступ до своєї системи.

Клієнтська частина буде реалізована у вигляді Web-додатку, що виконуватиметься у браузері.

3. Функціональні вимоги до додатку

Перелік функціональних можливостей визначений базовими діями, що виконує комп'ютерна мишка:

					КНТЕУ 121 06М-06.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		28.02.20	Розробка системи пошуку спільних ідей та інтересів	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Пашорін В.І.		28.02.20		*ТЗ*	*33*	*37*
Гарант		Криворучко О.В.		28.02.20		Факультет інформаційних технологій 2м курс, 6М група		
Розробив		Гернець М.А.		28.02.20	*Технічне завдання*			

ТЕСТУВАННЯ ДОДАТКА SEARCHSYSTEM

З метою забезпечення якості розробленого програмного забезпечення нами було проведено тестування створенного сервісу.

Для тестування розробленої програми було використано реальні пристрої з наступними конфігураціями (табл. 3.1).

Таблиця Т.1

Конфігурація пристроїв, використаних під час тестування

Характеристика	**Комп'ютер-Сервер**	**Комп'ютер-Клієнт**
Операційна система	Windows 10 (x64)	Windows 10 (x64)
Процессор	Ryzen 5 1600	Intel I7-6700HQ
Об'єм ОЗУ	16 Гб	16 Гб
Видеокарта	Gtx 1060 3gb	Gtx 960m 4gb
Спосіб підключення до локальної мережі	Wi-Fi	Wi-Fi

На основі вимог до програмного забезпечення (технічне завдання) було розроблено перелік обов'язкових позитивних тест-кейсів (тестових випадків), виконання яких ми перевірили. Зміст тест-кейсів зазначено у табл. 3.2.

Передумова: виконано запуск обох частин програмного забезпечення (клієнтської та серверної).

Тест-кейс 1.1.

```
[Test]
public void Rebuild_Index()
{
    using (var d = new RandomIdRAMDirectory())
    using (var indexer = new TestIndex(d, new StandardAnalyzer(Version.LUCENE_30)))
    {
        indexer.CreateIndex();
        indexer.IndexItems(indexer.AllData());

        var indexWriter = indexer.GetIndexWriter();
        var reader = indexWriter.GetReader();
        Assert.AreEqual(100, reader.NumDocs());
    }
}
```

Тест-кейс 1.2.

```
[Test]
public void Index_Exists()
{
    using (var luceneDir = new RandomIdRAMDirectory())
    using (var indexer = new TestIndex(luceneDir, new StandardAnalyzer(Version.LUCENE_30)))
    {
        indexer.EnsureIndex(true);
        Assert.IsTrue(indexer.IndexExists());
    }
}
```

Тест-кейс 1.3.

```
[Test]
public void Can_Add_One_Document()
{
    using (var luceneDir = new RandomIdRAMDirectory())
    using (var indexer = new TestIndex(luceneDir, new StandardAnalyzer(Version.LUCENE_30)))
    {


        indexer.IndexItem(new ValueSet(1.ToString(), "content",
            new Dictionary<string, IEnumerable<object>>
            {
                {"item1", new List<object>(new[] {"value1"})},
                {"item2", new List<object>(new[] {"value2"})}
            }));

        var indexWriter = indexer.GetIndexWriter();
        var reader = indexWriter.GetReader();
        Assert.AreEqual(1, reader.NumDocs());
    }
}
```

Тест-кейс 1.4.

```
[Test]
public void Can_Add_Same_Document_Twice_Without_Duplication()
{
    using (var luceneDir = new RandomIdRAMDirectory())
    using (var indexer = new TestIndex(luceneDir, new StandardAnalyzer(Version.LUCENE_30)))
    {


        var value = new ValueSet(1.ToString(), "content",
            new Dictionary<string, IEnumerable<object>>
```

```
            {
                {"item1", new List<object>(new[] {"value1"})},
                {"item2", new List<object>(new[] {"value2"})}
            });

        indexer.IndexItem(value);
        indexer.IndexItem(value);

        var indexWriter = indexer.GetIndexWriter();
        var reader = indexWriter.GetReader();
        Assert.AreEqual(1, reader.NumDocs());
    }
}
```

Тест-кейс 1.5.

```
[Test]
    public void Can_Add_Multiple_Docs()
    {
        using (var luceneDir = new RandomIdRAMDirectory())
        using (var indexer = new TestIndex(luceneDir, new StandardAnalyzer(Version.LUCENE_30)))
        {
            for (var i = 0; i < 10; i++)
            {
                indexer.IndexItem(new ValueSet(i.ToString(), "content",
                    new Dictionary<string, IEnumerable<object>>
                    {
                        {"item1", new List<object>(new[] {"value1"})},
                        {"item2", new List<object>(new[] {"value2"})}
                    }));
            }

            var indexWriter = indexer.GetIndexWriter();
            var reader = indexWriter.GetReader();
            Assert.AreEqual(10, reader.NumDocs());
        }
    }
```

Тест-кейс 1.6.

```
[Test]
    public void Can_Add_Doc_With_Easy_Fields()
    {
        using (var luceneDir = new RandomIdRAMDirectory())
        using (var indexer = new TestIndex(luceneDir, new StandardAnalyzer(Version.LUCENE_30)))
        {

            indexer.IndexItem(ValueSet.FromObject(1.ToString(), "content",
                new { item1 = "value1", item2 = "value2" }));

            using (var s = (LuceneSearcher)indexer.GetSearcher())
            {
                var luceneSearcher = s.GetLuceneSearcher();
                var fields = luceneSearcher.Doc(0).GetFields().ToArray();
                Assert.IsNotNull(fields.SingleOrDefault(x => x.Name == "item1"));
                Assert.IsNotNull(fields.SingleOrDefault(x => x.Name == "item2"));
                Assert.AreEqual("value1", fields.Single(x => x.Name == "item1").StringValue);
                Assert.AreEqual("value2", fields.Single(x => x.Name == "item2").StringValue);
            }
        }
    }
```

Тест-кейс 1.7.

```
    [Test]
public void Number_Field()
{
   using (var luceneDir = new RandomIdRAMDirectory())
   using (var indexer = new TestIndex(
      new FieldDefinitionCollection(new FieldDefinition("item2", "number")),
      luceneDir,
      new StandardAnalyzer(Version.LUCENE_30)))
   {


      indexer.IndexItem(new ValueSet(1.ToString(), "content",
         new Dictionary<string, IEnumerable<object>>
         {
            {"item1", new List<object>(new[] {"value1"})},
            {"item2", new List<object>(new object[] {123456})}
         }));

      using (var s = (LuceneSearcher)indexer.GetSearcher())
      {
         var luceneSearcher = s.GetLuceneSearcher();
         var fields = luceneSearcher.Doc(luceneSearcher.MaxDoc - 1).GetFields().ToArray();

         var valType = indexer.FieldValueTypeCollection.GetValueType("item2");
         Assert.AreEqual(typeof(Int32Type), valType.GetType());
         Assert.IsNotNull(fields.SingleOrDefault(x => x.Name == "item2"));
      }
   }

}
```

ДОДАТКИ

ДОДАТОК А

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.ComponentModel;
using System.IO;
using System.Linq;
using LuceneEngine.Search;
using Lucene.Net.Analysis;
using Lucene.Net.Analysis.Standard;
using Lucene.Net.Search;

namespace LuceneEngine.Providers
{
   ///<summary>
   /// A provider that allows for searching across multiple indexes
   ///</summary>
   public class MultiIndexSearcher : BaseLuceneSearcher, IDisposable
   {
      private readonly Lazy<IEnumerable<ISearcher>> _searchers;

      #region Constructors

      /// <summary>
      /// Constructor to allow for creating a searcher at runtime
      /// </summary>
      /// <param name="name"></param>
      /// <param name="indexes"></param>
      /// <param name="analyzer"></param>
      public MultiIndexSearcher(string name, IEnumerable<IIndex> indexes, Analyzer analyzer = null)
         : base(name, analyzer ?? new StandardAnalyzer(Lucene.Net.Util.Version.LUCENE_30))
      {
         _searchers = new Lazy<IEnumerable<ISearcher>>(() => indexes.Select(x => x.GetSearcher()));
         _disposer = new DisposableSearcher(this);
      }

      /// <summary>
      /// Constructor to allow for creating a searcher at runtime
      /// </summary>
      /// <param name="name"></param>
      /// <param name="searchers"></param>
      /// <param name="analyzer"></param>
      public MultiIndexSearcher(string name, Lazy<IEnumerable<ISearcher>> searchers, Analyzer analyzer = null)
         : base(name, analyzer ?? new StandardAnalyzer(Lucene.Net.Util.Version.LUCENE_30))
      {
         _searchers = searchers;
         _disposer = new DisposableSearcher(this);
      }

      #endregion


      ///<summary>
      /// The underlying LuceneSearchers that will be searched across
      ///</summary>
      public IEnumerable<LuceneSearcher> Searchers => _searchers.Value.OfType<LuceneSearcher>();

      /// <summary>
      /// Returns a list of fields to search on based on all distinct fields found in the sub searchers
      /// </summary>
      /// <returns></returns>
      public override string[] GetAllIndexedFields()
      {
         var searchableFields = new List<string>();
         foreach (var searcher in Searchers)
         {
```

```
            searchableFields.AddRange(searcher.GetAllIndexedFields());
        }
        return searchableFields.Distinct().ToArray();
    }

    /// <summary>
    /// Gets the searcher for this instance
    /// </summary>
    /// <returns></returns>

    public override Searcher GetLuceneSearcher()
    {
            var searchables = new List<Searchable>();
                    //NOTE: Do not convert this to Linq as it will fail the Code Analysis because Linq screws with it.
                    foreach(var s in Searchers)
                    {
                        var searcher = s.GetLuceneSearcher();
        if (searcher != null)
            searchables.Add(searcher);
                    }
                    return new MultiSearcher(searchables.ToArray());
    }

    public override ISearchContext GetSearchContext()
    {
        return new MultiSearchContext(GetLuceneSearcher(), Searchers.Select(s => s.GetSearchContext()).ToArray());
    }


    #region IDisposable Members

    private readonly DisposableSearcher _disposer;

    private class DisposableSearcher : DisposableObjectSlim
    {
        private readonly MultiIndexSearcher _searcher;

        public DisposableSearcher(MultiIndexSearcher searcher)
        {
            _searcher = searcher;
        }

        /// <summary>
        /// Handles the disposal of resources. Derived from abstract class <see cref="DisposableObject"/> which handles common
required locking logic.
        /// </summary>

        protected override void DisposeResources()
        {
            foreach (var searcher in _searcher.Searchers)
            {
                searcher.Dispose();
            }
        }
    }

    /// <summary>
    /// Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.
    /// </summary>
    public void Dispose()
    {
        _disposer.Dispose();
    }

    #endregion
}
                                                                                                                    }
```

ДОДАТОК Б

```
using System.IO;
using Lucene.Net.Analysis;

namespace LuceneEngine
{
   /// <summary>
   /// Used for email addresses
   /// </summary>
   public class EmailAddressAnalyzer : Analyzer
   {

      public override TokenStream TokenStream(string fieldName, TextReader reader)
      {
         return new LowerCaseFilter(               //case insensitive
            new EmailAddressTokenizer(reader));    //email tokenizer
      }

      /// <summary>
      /// Used for email addresses
      /// </summary>
      public class EmailAddressTokenizer : CharTokenizer
      {
         public EmailAddressTokenizer(TextReader input)
            : base(input)
         {
         }

         protected override bool IsTokenChar(char c)
         {
            // Make whitespace characters and the @ symbol be indicators of new words.
            return !(char.IsWhiteSpace(c) || c == '@');
         }
      }
   }
```

ДОДАТОК В

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security;
using Lucene.Net.Index;
using Lucene.Net.Store;

namespace LuceneEngine
{

   internal class OpenReaderTracker
   {
      private static readonly OpenReaderTracker Instance = new OpenReaderTracker();

      private readonly List<Tuple<IndexReader, DateTime>> _oldReaders = new List<Tuple<IndexReader, DateTime>>();

      private readonly object _locker = new object();

      public static OpenReaderTracker Current
      {
         get { return Instance; }
      }

      public void AddOpenReader(IndexReader reader)
      {
         lock (_locker)
         {
            //reader.IncRef();
            _oldReaders.Add(new Tuple<IndexReader, DateTime>(reader, DateTime.Now));
```

```
    }
}

public int CloseStaleReaders(Directory dir, TimeSpan ts)
{
    lock (_locker)
    {
        var now = DateTime.Now;

        var readersForDir = _oldReaders.Where(x => x.Item1.Directory().GetLockId() == dir.GetLockId()).ToList();
        var newest = readersForDir.OrderByDescending(x => x.Item2).FirstOrDefault();
        readersForDir.Remove(newest);
        var stale = readersForDir.Where(x => now - x.Item2 >= ts).ToArray();

        foreach (var reader in stale)
        {
            //close reader and remove from list
            try
            {
                reader.Item1.Close();
            }
            catch (AlreadyClosedException)
            {
                //if this happens, more than one instance has decreased referenced, this could occur if this
                //somehow gets called in conjuction with the shutdown code or manually, etc...
            }
            _oldReaders.Remove(reader);
        }
        return stale.Length;
    }

}

public int CloseAllReaders(Directory dir)
{
    lock (_locker)
    {
        var readersForDir = _oldReaders.Where(x => x.Item1.Directory().GetLockId() == dir.GetLockId()).ToArray();
        foreach (var reader in readersForDir)
        {
            //close reader and remove from list
            try
            {
                reader.Item1.Close();
            }
            catch (AlreadyClosedException)
            {
                //if this happens, more than one instance has decreased referenced, this could occur if this
                //somehow gets called in conjuction with the shutdown code or manually, etc...
            }

            _oldReaders.Remove(reader);
        }
        return readersForDir.Length;
    }
}

public int CloseAllReaders()
{
    lock (_locker)
    {
        var readers = _oldReaders.ToArray();
        foreach (var reader in readers)
        {
            //close reader and remove from list
            try
            {
                reader.Item1.Close();
            }
            catch (AlreadyClosedException)
```

```
                {
                    //if this happens, more than one instance has decreased referenced, this could occur if this
                    //somehow gets called in conjuction with the shutdown code or manually, etc...
                }

                _oldReaders.Remove(reader);
            }
            return readers.Length;
        }
    }

}
```

ДОДАТОК Г

```
using System;
using System.Collections.Concurrent;
using System.Security;
using Lucene.Net.Index;
using Lucene.Net.Store;

namespace LuceneEngine
{

    public sealed class WriterTracker
    {
        /// <summary>
        /// Used for tests
        /// </summary>
        internal void Reset()
        {
            _writers.Clear();
        }

        private readonly ConcurrentDictionary<string, IndexWriter> _writers = new ConcurrentDictionary<string, IndexWriter>();

        public static WriterTracker Current { get; } = new WriterTracker();

        public IndexWriter GetWriter(Directory dir)
        {
            return GetWriter(dir, false);
        }

        public IndexWriter GetWriter(Directory dir, bool throwIfEmpty)
        {
            IndexWriter d = null;
            if (!_writers.TryGetValue(dir.GetLockId(), out d))
            {
                if (throwIfEmpty)
                {
                    throw new NullReferenceException("No writer was added with directory key " + dir.GetLockId() + ", maybe an
indexer hasn't been initialized?");
                }
            }
            return d;
        }

        public IndexWriter GetWriter(Directory dir, Func<Directory, IndexWriter> factory)
        {
            var resolved = _writers.GetOrAdd(dir.GetLockId(), s => factory(dir));
            return resolved;
        }

    }
```