НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки
Кафедра Автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедрою
_____ Ролік О. І.
(підпис) (ініціали, прізвище)
«__»_____ 2019 р.

Магістерська дисертація

зі спеціальності 126 Інформаційні системи та технології

на тему: «Система пошуку елементів на веб-сторінках із динамічною структурою»

Виконав: студент 6-го курсу, групи ІА-82мп
(шифр групи)

Савін Михайло Сергійович
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник к.т.н, доцент каф. АУТС, Кравець П. І.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
(повна назва)

Кафедра автоматики та управління в технічних системах
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність 126 Інформаційні технології та системи
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ролік О. І.
(підпис) (ініціали, прізвище)
«__» _____ 2019 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Савіну Михайлу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Система пошуку елементів на веб-сторінках із динамічною структурою»

науковий керівник дисертації Кравець Петро Іванович, к.т.н.,
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)
доцент кафедри АУТС

затверджені наказом по університету від «___» _____ 2019 р. №___

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження система пошуку елементів на веб-сторінках із динамічною структурою

4. Предмет дослідження елементи веб-сторінок зі структурою, що може змінюватися

5. Перелік завдання, які потрібно розробити огляд існуючих рішень, огляд моделі документа, розробка моделі, розробка системи, графічний матеріал

6. Орієнтовний перелік ілюстративного (графічного) матеріалу: схема структурна, схема функціональна, діаграма сценаріїв, блок-схеми алгоритмів, приклади роботи алгоритмів та системи

7. Орієнтовний перелік публікацій: «Міжнародна наукова інтернет-конференція на тему "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення"»

8. Консультанти розділів дисертації

9. Дата видачі завдання – 02.09.2019

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Строк виконання етапів проекту	Примітка
1.	Огляд існуючих рішень	2.09.2019 р.	
2.	Огляд об’єктної моделі документа	14.09.2019 р.	
3.	Розробка алгоритму	28.09.2019 р.	
4.	Аналіз результатів та вибір архітектури	1.10.2019 р.	
5.	Розроблення системи	15.10.2019 р.	
6.	Аналіз отриманих результатів	31.10.2019 р.	
7.	Розроблення схеми структурної	5.11.2019 р.	
8.	Розроблення схеми функціональної	10.11.2019 р.	
9.	Розробка стартап – проекту	15.11.2019 р.	
10	Оформлення текстової документації	28.11.2019 р.	

Студент _____ Савін М. С.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Кравець П. І.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Магістерська дисертація на здобуття ступеня магістру на тему «Система пошуку елементів на веб-сторінках зі змінюваною структурою»: 110с., 23 рис., 35 табл., 9 додатків, 33 джерела.

Об'єкт дослідження – система пошуку елементів на веб-сторінці із динамічною структурою.

Мета роботи – оптимізація розробка системи, що дозволяє ідентифікувати елементи веб-сторінки у разі змін її структури.

У магістерській дисертації розглядається проблема пошуку елементів на веб-сторінках, що зазнають змін. Пропонується алгоритм пошуку елементів на веб-сторінках. Також пропонується система, що на основі даного алгоритму здатна в автоматичному режимі ідентифікувати елементи на сторінках після зміни їхньої структури.

ВЕБ-СТОРІНКА, КОМБІНАТОРИКА, ВЕБ-ЗАСТОСУНОК, NODE.JS, CDN, POSTGRESQL, DOM

Aster's thesis for master's degree on the topic «A system for finding elements on web pages with dynamic structure»: 110p, 23 figures, 35 tables, 9 annexes, 33 sources.

Object of study – a system system for finding elements on web pages with dynamic structure.

The purpose of the work – optimization of the system which allows to identify elements of web-page in case of structural changes.

The master's thesis deals with the problem of search of elements on web-pages which are being changed. The algorithm for search of elements on web pages is proposed. In addition, the system that is able to identify elements on pages after structural changes basing on this algorithm is proposed.

WEB PAGE, COMBINATORICS, WEB APPLICATION, NODE.JS, CDN, POSTGRESQL, DOM

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ

1) DOM (Document Object Model) – API програмування для документів HTML та XML, що визначає логічну структуру документів та спосіб доступу до документа та маніпулювання ним;

2) DOM-дерево – ієрархічна деревоподібна структура елементів веб-сторінки;

3) елемент веб-сторінки – структурна складова веб-сторінки, що може вміщувати інші елементи або інформацію, з якою може взаємодіяти користувач веб-сторінки;

4) тег – властивість елемента веб-сторінки, що визначає його тип, а також використовується для позначення границь елемента веб-сторінки;

5) атрибут – властивість елемента веб-сторінки, що складається з імені та значення й розташовується всередені тегу елемента;

6) клас елемента веб-сторінки – атрибут елемента веб-сторінки, що пов’язує елемент та його CSS-стилі, що визначають його зовнішній вигляд;

7) UX дизайн – процес створення простого й зручного для користувача інтерфейсу програми;

8) API – це набір способів, за допомогою яких одна комп’ютерна програма може взаємодіяти з іншою програмою;

9) HTML – це стандарт розмітки документів, що призначені для відображення у веб-браузері;

10) CSS – це мова, розроблена для опису зовнішнього вигляду документа веб-сторінки;

11) реплікація – метод зменшення навантаження на базу даних шляхом безперервного копіювання даних з одного серверу бази даних до іншого. При цьому клієнти взаємодіють з обома серверами, в один з яких записуються нові дані, а з іншого зчитуються;

12) шардування – це спосіб зменшення навантаження на базу даних, що передбачає розділення даних між декількома серверами. На різних серверах можуть розташовуватися як різні таблиці, так і різні частини великих за об’ємом таблиць;

13) CDN – це сукупність географічно розподілених серверів, які працюють разом, щоб забезпечити швидке завантаження інтернет-сторінок у будь-якій точці земної кулі;

14) кешування – процес розташування даних у тимчасовому сховищі, що, як правило, є менш надійним, але більш швидким, ніж основне.

ВСТУП

Проблема пошуку елементу на веб-сторінці з’являється тоді, коли веб-сторінка підлягає тим чи іншим змінам. Наприклад, власник сайту може випустити оновлення із певними змінами у структурі, або ж контент на сторінці може бути змінений з настанням певної дати, або ж елементи сторінки можуть залежати від налаштувань користувача, його прав доступу до тих чи інших ресурсів.

Актуальність теми. Інтернет використовує більшість людей на планеті, й на даний момент існує велика кількість сайтів, з якими повинні взаємодіяти співробітники різних компаній. Чимало компаній хочуть прискорити засвоєння співробітниками нових інструментів, що лежать на тих чи інших веб-ресурсах. Для цього необхідно мати змогу прив’язати певні підказки до елементів інтерфейсу. Однак оскільки веб-сторінки дуже часто змінюються, необхідно мати змогу ідентифікувати елемент веб-сторінки, до якої була прив’язана підказка, якщо сторінка змінилася.

Мета і задачі досліджень. Метою досліджень є розробка системи, що дозволяє ідентифікувати елементи веб-сторінки у разі змін її структури. Для цього система повинна мати наступний перелік властивостей та мати здатність вирішувати наступні задачі:

- система повинна мати змогу працювати в автоматичному режимі, тобто приймати рішення без експертної оцінки або підготовчих дій людини;
- система повинна однозначно ідентифікувати елемент, що має бути знайдений на сторінці, у разі будь-яких змін її структури або ж у разі змін властивостей елемента або його батьківських елементів;
- система повинна ідентифікувати елемент базуючись лише на інформації, що була отримана нею при виборі елементу на сторінці з початковою структурою.

Об’єктом досліджень є система пошуку елементів на веб-сторінці із динамічною структурою.

Предметом досліджень є елементи веб-сторінок зі структурою, що може змінюватися.

Методи досліджень базуються на використанні:

- алгоритму побудови об’єкта DOM-дерева;
- інструментів з комбінаторики та теорії множин: декартів добуток множин, комбінація множини;
- методів пошуку елементу базуючись на його координатах відносно початку сторінки та його властивостях (клас, тег, атрибути).

Основними здобутками дисертації є розроблений алгоритм пошуку елементів на веб-сторінці та система, що на основі даного алгоритму здатна в автоматичному режимі ідентифікувати елементи на сторінці після зміни її структури. Дана система може бути використана розробниками систем підказок на веб-сторінках, що дозволяють пришвидшити оволодіння людиною веб-інтерфейсом.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Даний розділ вміщує аналіз систем та алгоритмів пошуку елементів на веб-сторінках із динамічною структурою.

1.2 Огляд існуючих систем пошуку елементів на веб-сторінках

Даний підрозділ вміщує аналіз існуючіх систем пошуку елементів на веб-сторінках, їхні переваги та недоліки, що мають бути виправлені при розробці системи пошуку елементів на веб-сторінці із динамічною структурою.

1.1.1 WalkMe

WalkMe – це система керівництва і взаємодії корпоративного класу, що спонукає користувачів до дій при використанні програмного забезпечення або веб-сайтів. WalkMe використовується підприємствами з різних галузей для збільшення продажів і коефіцієнтів конверсії, використання принципів UX дизайну, зниження витрат на підтримку і підвищення продуктивності праці співробітників. Всебічне покрокове керівництво надається через послідовність спливаючих підказок, при цьому користувачеві системи не потрібно залишати веб-сторінку, дивитися відеоуроки або втрачати час на читання керівництв або сторінок поширених запитань [1].

На даний момент WalkMe використовує алгоритм пошуку елементів на сторінках, заснований на даних елементу (його клас, атрибути, css-стилі, положення на сторінці відносно лівого верхнього кута, текст всередині елементу) безвідносно його батьківських елементів й його положення у структурі сторінки, що не забезпечує точність знаходження елементів й може знаходити елементи невірно або потребує втручання людини в процес пошуку елементу у складних випадках, оскільки елемент може змінити як своє положення на сторінці, так і отримати повністю нові стилі та

атрибути, що зробить неможливим знайти його за даними критеріями. На рисунку 1.1 зображена ілюстрація взаємодії різних компонентів платформи WalkMe.

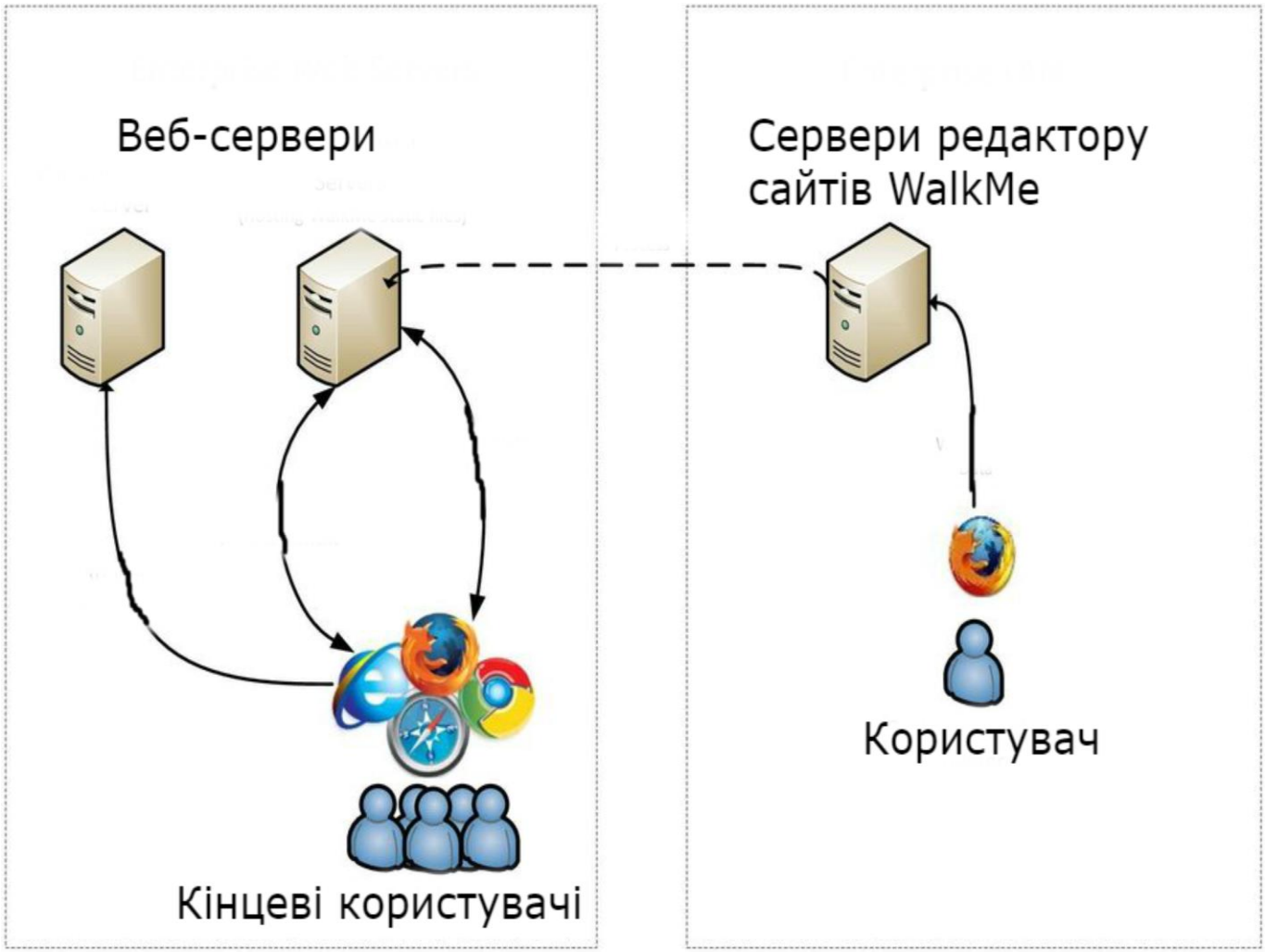

Рисунок 1.1 – Ілюстрація взаємодії різних компонентів платформи WalkMe [2]

У компанії розроблено чимало застосунків, що спрощують прив’язання підказок до елементу, таких як розширення для веб-браузерів, редактори підказок для різних операційних систем, що взаємодіють з браузером й дозволяють обирати елемент зручним чином, а також javascript-бібліотеки для вбудови на веб-сторінку, на якій буде здійснено вибір елементів.

До елемента можливо прикріпити багато різних типів підказок, від тексту або посилання на потрібний розділ до відео або текстового поля. Стиль та наповнення даних підказок повністю визначається користувачем (рисунок 1.2).

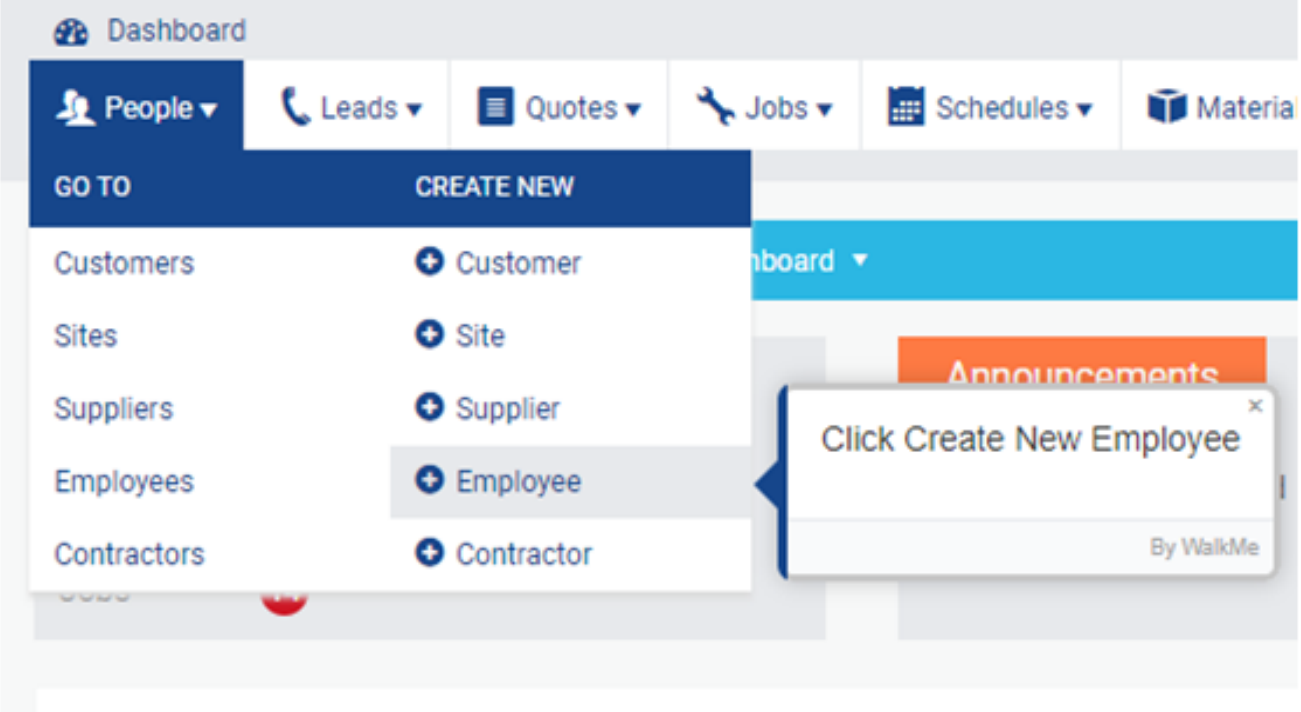

Рисунок 1.2 – Приклад підказки WalkMe [3]

1.1.2 Appcues

Appcues – це система, що дозволяє, так само, як, WalkMe, створювати підказки на веб-сторінках, що будуть прив’язані до певних елементів та дій користувача на веб-сторінці. Це дозволяє спростити процес навчання нових співробітників та клієнтів для компаній, що використовують Appcues [4].

Appcues надає користувачеві можливість обрати один з декількох типів підказок, що будуть прив’язані до елементів, а також обрати один з декількох варіантів їхнього зовнішнього вигляду (рисунок 1.3).

Appcues дозволяє додавати підказки до елемента за допомогою javascript-бібліотеки, що вбудовується на сторінку користувачем з використанням тега script.

Алгоритм, що використовує компанія, також заснований на даних елементу безвідносно його положення у структурі веб-сторінки й не дозволяє знайти елементи в автоматичному режимі на зміненій веб-сторінці у разі, якщо сторінка зазнала нетипових змін. Це призводить до необхідності перестворювати елементи вручну, що

є проблемним у разі великої кількості елементів, що потребують прив’язання підказок.

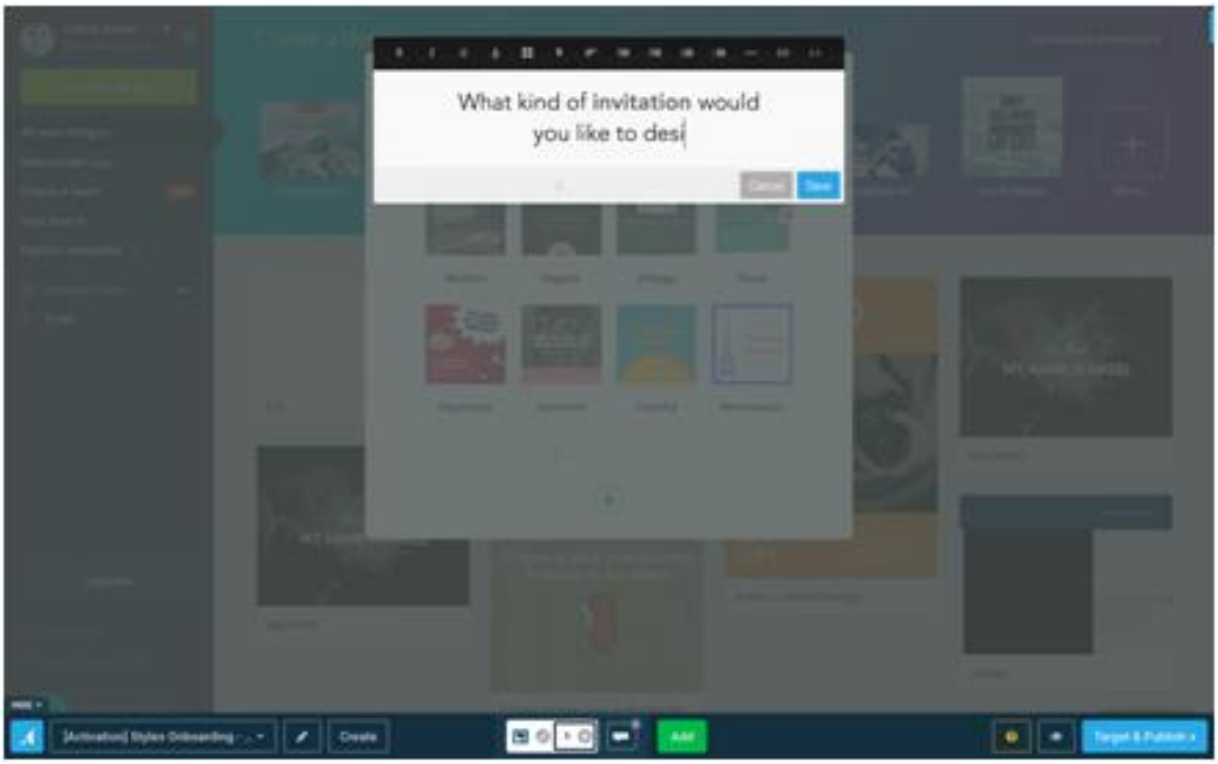

Рисунок 1.3 – Інтерфейс Appcues [5]

1.1.3 Whatfix

Whatfix є конкурентом WalkMe та Appcues й, так само, як перші дві системи, надає користувачеві можливість прикріплення підказок до елементів на веб-сторінці. Платформа Whatfix забезпечує впровадження нових веб-сервісів, адаптацію користувачів, навчання співробітників, самообслуговування і підтримку продуктивності для компаній, що використовують корпоративні веб-застосунки [6]. Платформа дозволяє користувачам створювати інтерактивні покрокові керівництва або прив’язувати до елементів інструкції й підказки, які можуть допомогти користувачам виконати певну задачу (рисунок 1.4).

Алгоритм пошуку елементів на сторінці, що використовується компанією Whatfix, заснований на використанні jQuery-селекторів, завдяки чому він може

знаходити елементи на сторінці за незначних змін структури веб-сторінки, однак у разі значних змін не може функціонувати автономно. Приклад підказки Whatfix зображено на рисунку 1.5.

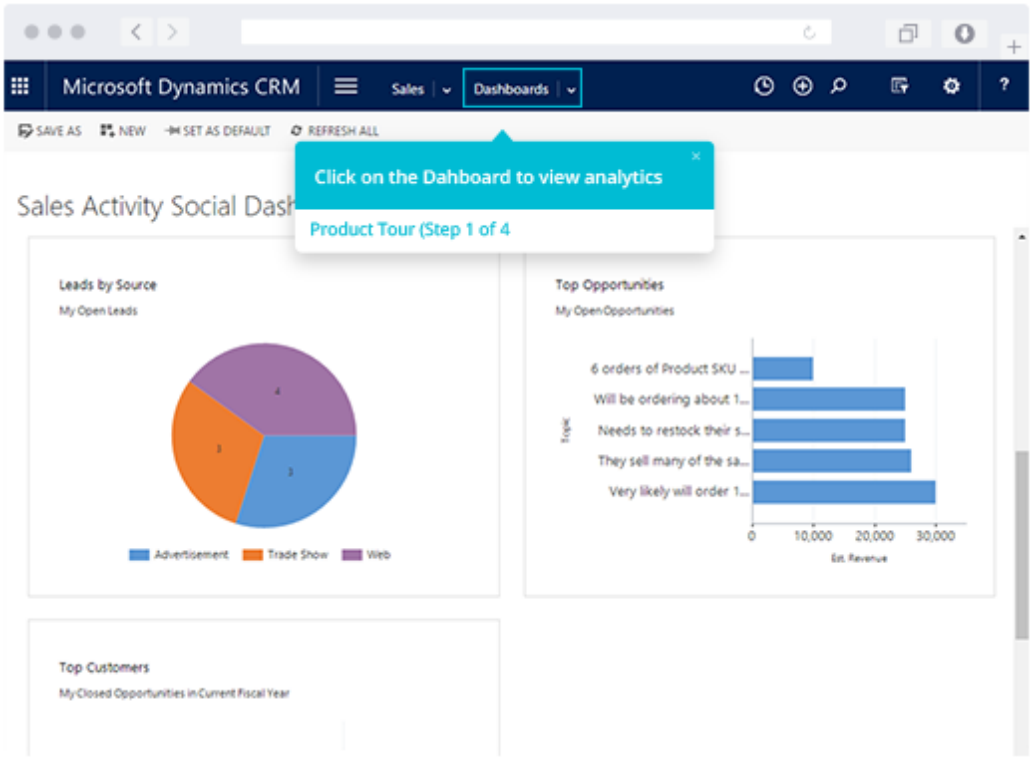

Рисунок 1.4 – Інтерфейс Whatfix [7]

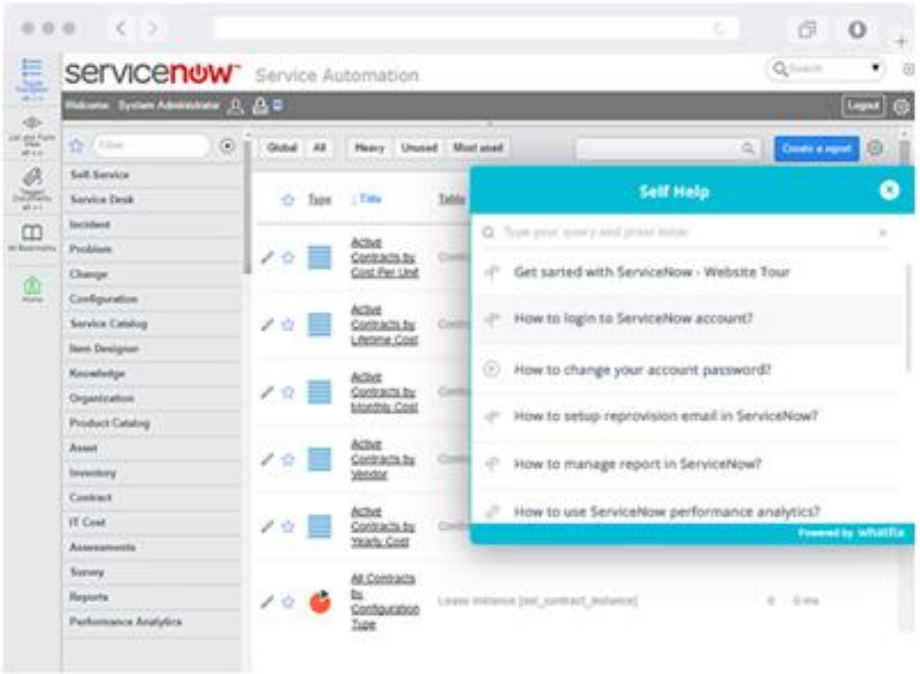

Рисунок 1.5 – Приклад підказки Whatfix [8]

1.2 Огляд алгоритмів пошуку елементів на веб-сторінках

У даному підрозділі аналізуються алгоритми пошуку елементів на веб-сторінці, що можуть бути порівняні з алгоритмом, що використовується у розробленій системі.

1.2.1 Пошук елементів на сторінці використовуючи його атрибути

Атрибути тегів HTML є модифікаторами тегів на веб-сторінці, що також слугують для надання додаткової інформації про цей елемент. У більшості випадків вони визначаються парами ім’я=«значення», і вони завжди оголошуються у відкриваючій половині html-тега. HTML-тег на веб-сторінці, до якого додаються атрибути, виглядає наступним чином:

<tagname attributename=”value” attribute2=”othervalue” attribute3>

Some content

</tagname>

Прикладом атрибуту для тега <img>, що слугує для відображення зображення на веб-сторінці, є атрибут src, що надає змогу користувачеві вказати шлях до файлу зображення. Елемент з тегом <img> та атрибутом <src> виглядає наступним чином: .

Можна виділити чотири атрибути елементів, що використовуються найбільш часто й мають найбільш важливе значення для відображення елементів на сторінці:

- id – атрибут, що має бути унікальним для кожного елемента, і тому за умови наявності даного атрибута в елемента може бути використаний для ідентифікації даного елемента на сторінці;
- title – атрибут, до додає запропонований заголовок до елемента. Поведінка цього атрибута залежить від елемента, до якого цей атрибут додається. Найбільш часто даний атрибут відображається як підказка, коли користувач наводить курсор на елемент або під час завантаження елемента;

- class – атрибут, що використовується для асоціації елемента з таблицею стилів і визначає клас елемента. Найбільш часто атрибут клас використовується для визначення таких параметрів елемента, як його розміри, колір фону та тексту, відступі від інших елементів й позиціонування елементу й тексту в ньому за допомогою каскадних таблиць стилей;
- style – атрибут, що дозволяє програмісту визначити css-правила для відображення елемента безпосередньо всередині елемента (не використовуючи окремі файли таблиць стилей).

Об’єктна модель документа надає можливість шукати елементи на сторінці за його атрибутами. Можна виокремити наступні методи, що дозволяють шукати елементи на сторінці за його тегом або атрибутами:

- document.getElementById – метод, що приймає атрибут id елементу, що треба знайти, й повертає об’єкт даного елемента на сторінці;
- document.getElementsByTagName – метод, що приймає тег елементу, що треба знайти, й повертає об’єкт даного елемента на сторінці;
- document.getElementsByClassName – метод, що приймає атрибут class елементу, що треба знайти, й повертає об’єкт даного елемента на сторінці;
- document.getElementsByName – метод, що приймає атрибут name елементу, що треба знайти, й повертає об’єкт даного елемента на сторінці
- document.querySelector – універсальний метод, що дозволяє шукати елемент одразу за багатьма атрибутами, включаючи атрибути батьківських елементів по відношенню до даного елемента.

Використовуючи дані методи, можна досить просто побудувати систему, що буде ідентифікувати елемент на сторінці за допомогою пошуку даного елементу за його атрибутами. Переваги даного підходу:

- простота реалізації;
- велика швидкодія;
- відсутність необхідності збереження великої кількості даних про веб-сторінку та її елементи.

Однак даний підхід має ряд недоліків, що змушують розробників шукати більш досконалі рішення, а саме:

- далеко не кожен елемент має ті чи інші атрибути. Навіть атрибут id, що дозволяє ідентифікувати елемент, не є обов’язковим для елементу. Отже, якщо елемент не має жодних атрибутів, його не можна знайти за атрибутами;
- атрибути елемента можуть бути змінені з часом. Це призведе до того, що атрибут, за яким система буде намагатися знайти елемент, не дозволить знайти бажаний елемент;
- більшість атрибутів можуть не бути унікальними. Це призводить до того, що за одним й тим самим атрибутом може бути знайдена велика кількість елементів, що сильно ускладнює ідентифікацію потрібного елементу.

Отже, даний алгоритм є недостатньо ефективним для елементів, що не мають атрибутів.

1.2.2 Пошук елементів на сторінці за його положенням відносно лівого верхнього кута сторінки

Javascript API об’єктної моделі документа надає користувачу можливість вирахувати відстань від елементу до кожного краю екрану. Для цього можна використати наступні методи:

- element.getBoundingClientRect() – метод, що повертає об’єкт DOMRect, що надає доступ до таких властивостей елемента, як top, left, right, bottom;
- element.offsetTop та element.offsetLeft це альтернатива методу element.getBoundingClientRect() для старих веб-браузерів, які не мають підтримки даного методу.

Отже, є можливість ідентифікувати елемент базуючись на відстані від цього елементу до верхнього лівого кута. Даний підхід має ряд суттєвих переваг:

- кожен елемент має координати, на відміну від певного переліку атрибутів;

- кожен елемент, що може бути виділений на сторінці користувачем, має унікальний набір координат та розмірів, а отже елемент можна однозначно ідентифікувати за даними критеріями;
- у більшості випадків зміна структури веб-сторінки не є значною, отже з великою ймовірністю елемент можна знайти за його старими координатами або поруч з ними;
- даний підхід може бути використаний у комбінації з іншими підходами, наприклад, у комбінації з пошуком елементу за допомогою атрибутів, що збільшить точність пошуку елемента.

Однак, даний спосіб пошуку елементів має також суттєвий недолік. веб-сторінка може суттєво змінити свою структуру, і тоді координати елементів на сторінці будуть “перемішані” відносно координат, що належали елементам раніше. У даному випадку пошук елемента на сторінці за його координатами не дозволить знайти цей елемент, у випадку ж, коли координати елемента використовуються у комбінації з його атрибутами, використання координат буде лише погіршувати точність пошуку, адже відстань від початкових координат елемента до кінцевих може бути дуже великою.

1.2.3 Пошук елементів на сторінці за його CSS-стилями

CSS (Cascading Style Sheets, або каскадні таблиці стилів) – це стандарт, що описує, як елементи на HTML-сторінці повинні виглядати при перегляді користувачем на веб-сторінці. Каскадні таблиці стилів здатні задавати правила відображення елементів одразу на багатьох веб-сторінках. Зазвичай таблиці стилів розміщуються у файлах з розширенням css. На рисунку 1.6 зображена веб-сторінка, що не має жодних css правил. Так виглядали сторінки на початку розвитку стандарту HTML.

Якщо ж застосувати css-правила на даній сторінці, то можна значно змінити її вигляд (рисунок 1.7).

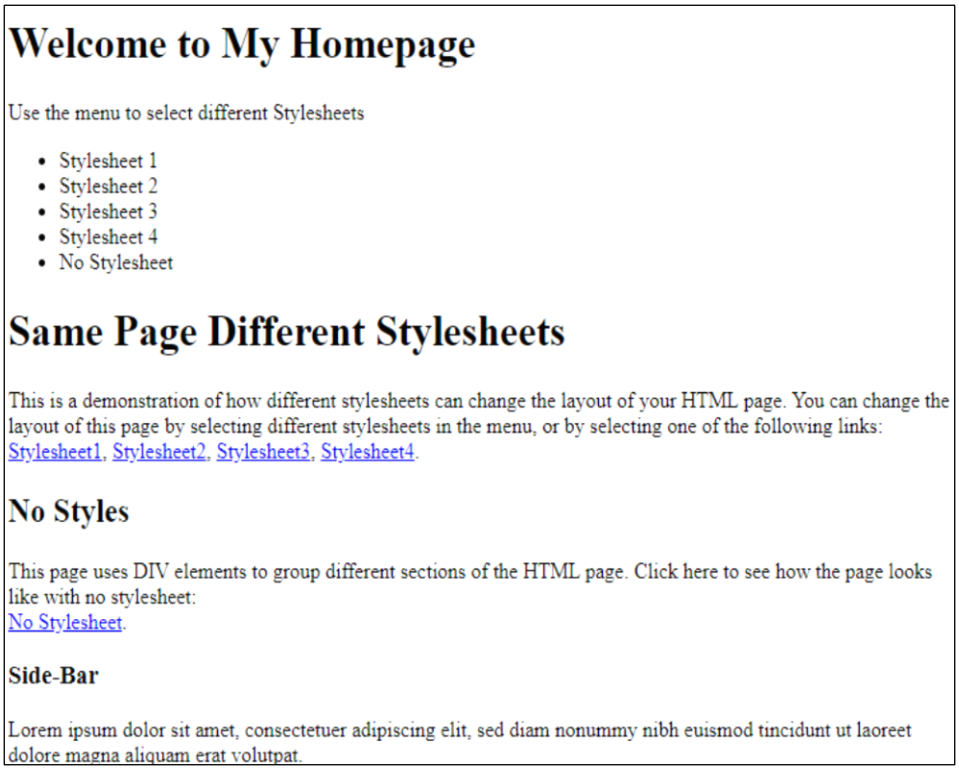

Рисунок 1.6 – Вигляд веб-сторінок на початку розвитку стандарту HTML [9]

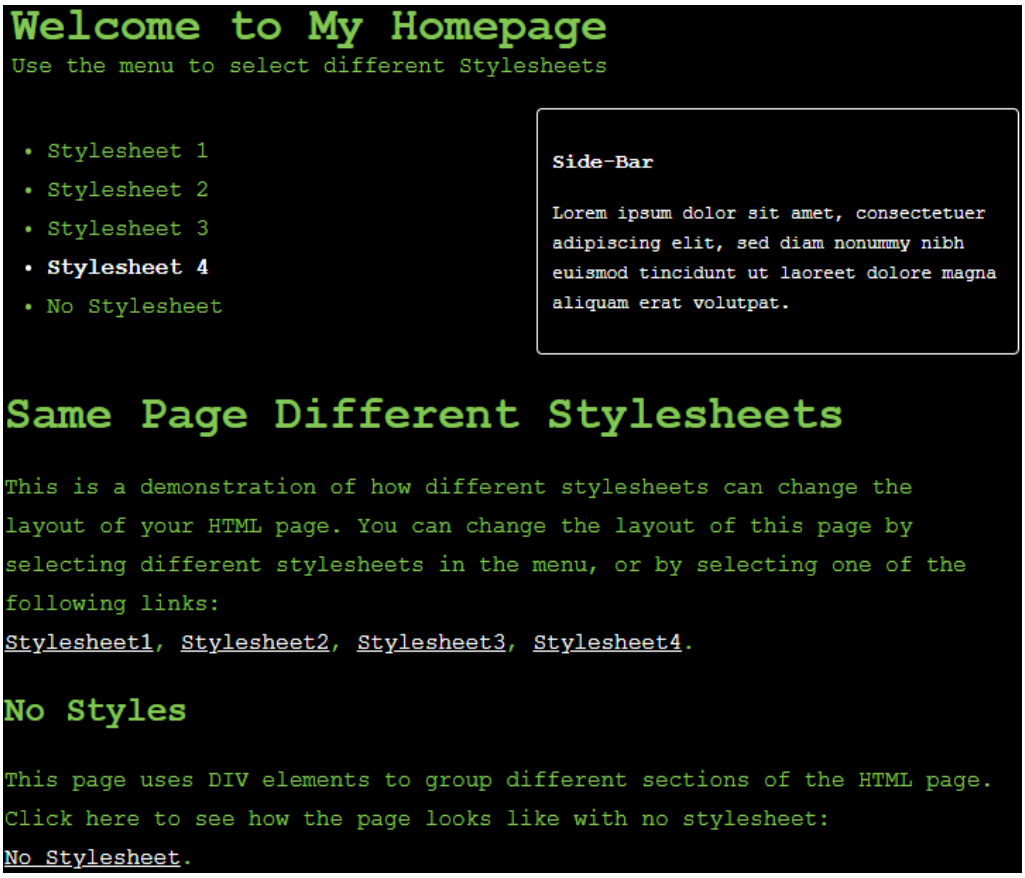

Рисунок 1.7 – Веб-сторінка після застосування css-правил [9]

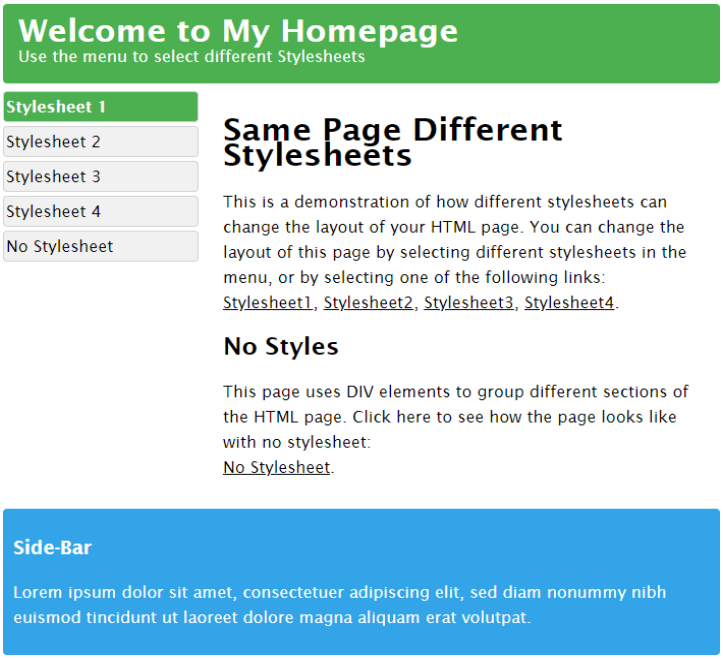

Рисунок 1.8 – Веб-сторінка після застосування інших css-правил [9]

Також, застосовуючи інші правила стилів для елементів на сторінці, можна надати елементам на веб-сторінці іншого вигляду (рисунок 1.8).

Використовуючи припущення про те, що з часом елементи, навіть змінюючи свій атрибут class, будуть переважно зберегати свій зовнішній вигляд та інші параметри, що задаються css-правилами, можна розробити алгоритм, що зберігає css-правила для елементу, й після зміни структури сторінки шукає елемент із css-стилями, що максимально схожі на збережені стилі.

Даний алгоритм має ряд переваг:

– простота реалізації порівняння різних значень стилів між собою;
– малий об’єм даних, що доведеться зберігати;
– відсутність необхідності взаємодії з об’єктною моделлю документа при пошуку елемента (можна обмежитися порівнянням збережених стилів елемента зі стилями, що розташовані у css-файлах оновленої сторінки).

Однак, також даний алгоритм має ряд недоліків:

– відсутність можливості знаходити елементи на старих сторінках, що не мають жодних css-правил;

- відсутність можливості знаходити елементи у разі значної зміни зовнішнього вигляду веб-сторінки;
- складність знаходження усіх стилів для елементу (оскільки css-правила можуть прив’язуватися до елементу не тільки за його класом, а й також за його тегом, або ж за комбінацією тегів та класів елементу та його батьківських елементів).

Враховуючи дані недоліки, можна стверджувати, що даний алгоритм може бути використаний розробником лише як допоміжний для більш досконалого або комбінованого алгоритму.

У таблиці 1 наведено порівняння існуючих алгоритмів пошуку елементів на сторінці.

Таблиця 1 – Порівняння алгоритмів пошуку елементів на веб-сторінці зі змінюваною структурою

Здатність алгоритму	Пошук елементів на сторінці використовуючи його атрибути	Пошук елементів на сторінці за його положенням відносно лівого верхнього кута сторінки	Пошук елементів на сторінці за його CSS-стилями
Простота реалізації	+	+	-
Велика швидкодія	+	+	-
Відсутність необхідності збереження великої кількості даних	+	+	+

Продовження таблиці 1

Здатність алгоритму	Пошук елементів на сторінці використовуючи його атрибути	Пошук елементів на сторінці за його положенням відносно лівого верхнього кута сторінки	Пошук елементів на сторінці за його CSS-стилями
Можливість застосування до будь-якого елементу	-	+	-
Відсутність необхідності взаємодії з об’єктною моделлю документа	-	-	+
Можливість знаходити елемент за будь-яких змін сторінки	-	-	-

У даному розділі були проаналізовані існуючі системи та алгоритми ідентифікації елементів на веб-сторінці. Було проведено порівняння алгоритмів пошуку елементів на веб-сторінках за його атрибутами, за його положенням на сторінці, за його CSS-стилями. В результаті проведеного аналізу було встановлено, що існуючі алгоритми не дозволяють знаходити елемент на сторінці за будь-яких змін сторінки, а отже існуючі системи пошуку елементів не можуть працювати в автоматичному режимі.

2 ПОНЯТТЯ ОБ’ЄКТНОЇ МОДЕЛІ ДОКУМЕНТА

У даному розділі наводиться поняття об'єктної моделі документа, а також проводиться аналіз структури веб-сторінки та методів об'єктної моделі документа для визначення структури документа.

2.1 Об’єктна модель документа та структура веб-сторінки

Кожна веб-сторінка знаходиться всередині вікна браузера, який можна розглядати як об’єкт. Об’єкт документу представляє документ HTML, який відображається у цьому вікні. Об'єкт документу має різні властивості, які посилаються на інші об'єкти, які дозволяють отримувати доступ та змінювати вміст документа. Спосіб доступу та змінення вмісту документа називається об’єктною моделлю документу, або DOM. Об'єкти на веб-сторінці організовані в ієрархічну структуру. Ця ієрархічна структура застосовується до організації об'єктів у веб-документі.

Об’єктна модель документа (DOM, або Document Object Model) – API програмування для документів HTML та XML. Дана модель визначає логічну структуру документів та спосіб доступу до документа та маніпулювання ним. У специфікації DOM термін «документ» використовується в широкому розумінні – дедалі частіше XML використовується як спосіб представлення багатьох різних видів інформації, яка може зберігатися в різних системах, і багато з цього традиційно сприймається як дані, а не як документи. Тим не менш, XML представляє ці дані як документи, і DOM може бути використаний для управління цими даними [10].

За допомогою об’єктної моделі документа програмісти можуть створювати та редагувати документи, переміщуватися по їх структурі та додавати, змінювати чи видаляти елементи та вміст. До всього, що знаходиться в документі HTML або XML, можна отримати доступ, змінити, видалити або додати за допомогою моделі об'єкта документа, за кількома винятками – зокрема, інтерфейси DOM для внутрішньої

підмножини та зовнішньої підмножини ще не визначені.

За специфікацією W3C, однією з важливих цілей для об’єктної моделі документа є створення стандартного інтерфейсу програмування, який може бути використаний у найрізноманітніших середовищах та програмах. Модель об'єкта документа може використовуватися з будь-якою мовою програмування.

Безпосередньо об'єктна модель документа дуже нагадує структуру документів, які вона моделює. Нижче наведено елемент типу table, взятий з документа HTML.

```
<table>
<rows>
<tr>
<td>Shady Grove</td>
<td>Aeolian</td>
</tr>
<tr>
<td>Over the River, Charlie</td>
<td>Dorian</td>
</tr>
</rows>
</table>
```

Об’єктна модель документу представляє цей елемент як деревоподібну структуру (рисунок 2.1).

У об’єктній моделі документа документи мають логічну структуру, яка має спільні риси з деревом або з набором дерев. Однак об’єктна модель документа не не задає структуру документа чи відносини між його елементами. Об'єктна модель задає логічну модель інтерфейсу програмування, і ця логічна модель може бути реалізована будь-яким способом, який користувач моделі вважає зручним. Однією з важливих властивостей моделей структур DOM є структурний ізоморфізм: якщо будь-які дві реалізації об'єктної моделі документа використовуються для створення представлення одного і того ж документа, вони створять одну й ту саму структуру моделі з повністю однаковими об'єктами та взаємозв'язками.

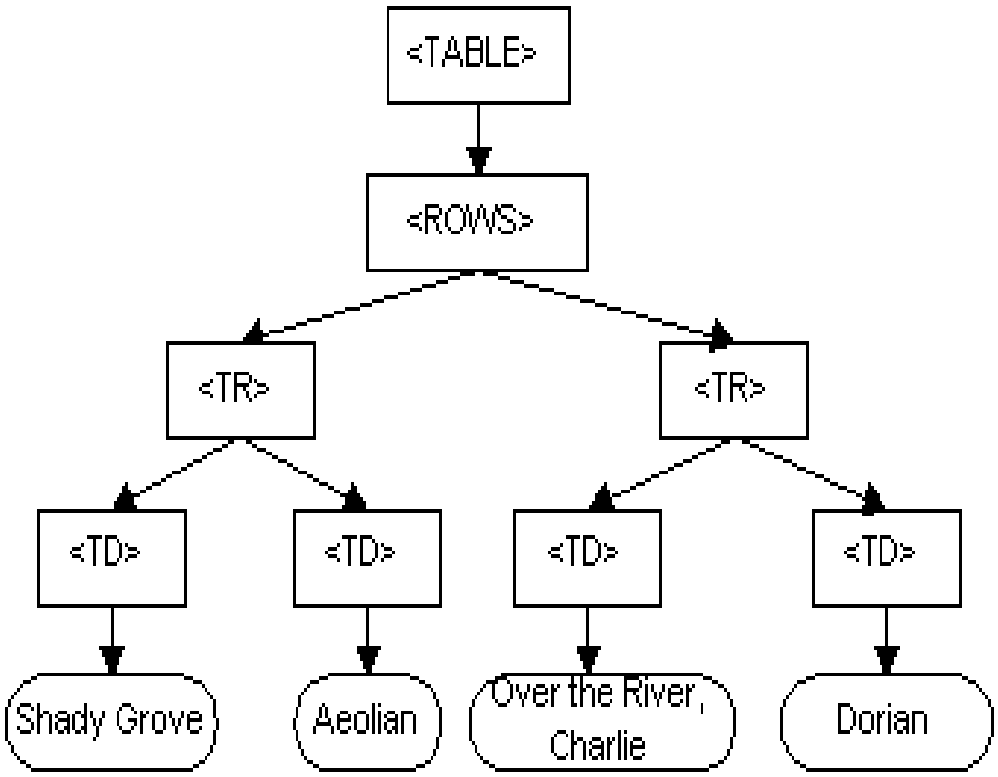

Рисунок 2.1 – Структура таблиці HTML [11]

Поняття “Об’єктна модель” використовується в традиційному об'єктно-орієнтованому розумінні: документи моделюються за допомогою об'єктів, а модель охоплює не тільки структуру документа, але і поведінку документа і елементів, з яких він складається. Іншими словами, вузли на рис. [TODO] не представляють структуру даних, вони представляють об'єкти, які мають функції та певні характеристики.

Як об’єктна модель, об’єктна модель документа визначає:

- інтерфейси та об'єкти, які використовуються для представлення та маніпулювання документом;
- семантику цих інтерфейсів та об'єктів, включаючи як поведінку, так і атрибути;
- взаємозв'язки та характер взаємодії між цими інтерфейсами та об'єктами.

Об’єктна модель документа в даний час складається з двох частин: ядро DOM та DOM HTML. Ядро DOM представляє функціонал, що використовується для XML-документів, а також служить основою для DOM HTML. Усі реалізації DOM повинні

підтримувати інтерфейси, перелічені як «основні» в специфікації ядра; крім того, реалізація XML повинна підтримувати інтерфейси, перелічені як «розширені» в специфікації ядра. Специфікація HTML DOM рівня 1 визначає додаткову функціональність, необхідну для документів HTML.

Модель об'єкта документа виникла як специфікація, що дозволяє переносити скрипти JavaScript та програми Java серед веб-браузерів. Динамічний HTML був безпосереднім предком об'єктної моделі документа, і спочатку він розвивався лише для роботи у веб-браузері. Однак, коли була сформована Робоча група з об'єктної моделі документа, до неї також приєдналися постачальники в інших областях, включаючи редактори HTML або XML та сховища документів. Деякі з цих постачальників працювали з SGML до розробки XML; як результат, на модель об'єкта документа вплинули такі стандарти, як SGML Groves та HyTime. Деякі з цих постачальників також розробили власні об'єктні моделі для документів, щоб надати API програмування для редакторів SGML чи XML або сховищ документів, і ці об'єктні моделі також вплинули на об'єктну модель документа.

У фундаментальних інтерфейсах DOM немає об'єктів, що представляють сутності. Числові посилання символів та посилання на попередньо визначені об'єкти в HTML та XML замінюються єдиним символом, що становить заміну об'єкта. Наприклад, у послідовності « This is a dog & a cat » «&» буде замінено на символ «&», а текст всередені елемента сформує єдине послідовне речення, що складається з букв. Представлення загальних сутностей, як внутрішніх, так і зовнішніх, визначається в розширених інтерфейсах (XML) специфікації рівня 1. Коли представлення DOM документа серіалізується як XML або HTML текст, програмам потрібно буде перевірити кожен символ у текстових даних, щоб побачити, чи потрібно його уникнути, використовуючи числовий або попередньо визначений об'єкт. Якщо цього не зробити, це може призвести до недійсного HTML або XML [12].

DOM задає інтерфейси, які можуть використовуватися для управління XML або HTML документами. Важливо усвідомити, що ці інтерфейси є абстракцією – подібно до «абстрактних базових класів» в C ++, вони є засобом визначення способу доступу та маніпулювання внутрішнім представленням документа додатком. Зокрема,

інтерфейси не передбачають конкретної реалізації. Кожна програма DOM вільна зберігати документи в будь-якому зручному представленні, якщо підтримуються інтерфейси, показані в цій специфікації. Деякі реалізації DOM будуть існуючими програмами, які використовують інтерфейси DOM для доступу до програмного забезпечення, написаного задовго до існування специфікації DOM. Тому DOM призначений для уникнення залежностей від впровадження.

Зокрема:

- атрибути, визначені в DOM, не передбачають конкретних об'єктів, які повинні мати конкретні дані – у мовних зв'язках вони переводяться на пару функцій get() та set(), а не в конкретні дані (функції лише для читання мають лише функцію get() у мовних зв’язках);
- розширення DOM можуть надавати додаткові інтерфейси та об'єкти, які не знайдені в оригінальній специфікації DOM, і все ще вважаються сумісними з DOM.

Нижче наведені приклади об’єктів, що можуть представляти елементи, що знаходяться на веб-сторінці:

- об’єкт window являє собою вершину ієрархії веб-сторінки. Це найвищий елемент в ієрархії об’єктів;
- об’єкт document це об’єкт, що створюється щоразу, коли html-документ завантажується у вікно браузера. Об’єкт document включає в себе весь вміст сторінки;
- об’єкт форми – об’єкт, що включає в себе все, що міститься всередині тегів <form>...</form>;
- елементи керування формою – усі об’єкти, що містить об’єкт форми й що слугують для маніпуляцій з нею (текстові поля, поля з варіантами вибору, кнопки та прапорці).

На рисунку 2.2 наведена ієрархія деяких важливих об’єктів, що можуть знаходитися на веб-сторінці.

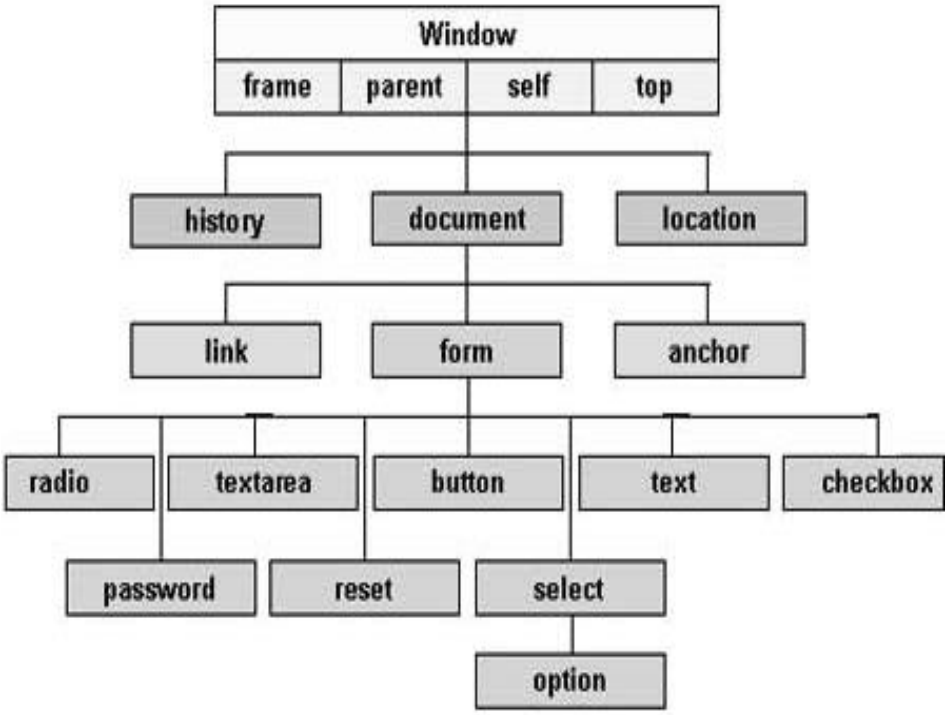

Рисунок 2.2 – Ієрархія деяких важливих об’єктів, що можуть знаходитися на веб-сторінці [12]

2.2 Методи об’єктної моделі документа для визначення структури документа

Будь-який HTML документ має деревоподібну структуру, й вузли цього дерева мають ієрархічні відношення один до одного. Для опису відношень між елементами на сторінці можуть використовуватися терміни «батьківський елемент», «дочірній елемент», а також «елемент зі спільним батьком». Також для структури веб-сторінки вірні наступні твердження:

- найвищий елемент дерева називають кореневим елементом;
- кожен елемент має тільки один батьківський елемент, окрім кореневого (що не має батьківського елемента);
- кожен елемент може мати будь-яку кількість дочірніх елементів.

Об’єктна модель документа надає можливість розробнику або людині, що переглядає веб-сторінку, отримати доступ до будь якого-елемента на сторінці, й, маючи доступ до цього елемента, отримати доступ до елемента, що слідує за ним, до розташований перед ним, до його батьківського елемента а також до елементів, що

лежать всередині даного елемента [13]. Для цього використовуються наступні властивості:

- element.parentNode – властивість елементу, що повертає батьківський елемент по відношенню до елемента, на якому вона викликається;
- element.childNodes – властивість елементу, що повертає дочірні елементи по відношенню до елемента, на якому вона викликається;
- element.firstChild – властивість елементу, що повертає перший дочірній елемент по відношенню до елемента, на якому вона викликається;
- element.lastChild – властивість елементу, що повертає останній дочірній елемент по відношенню до елемента, на якому вона викликається;
- element.nextSibling – властивість елементу, що повертає наступний елемент зі спільним батьком по відношенню до елемента, на якому вона викликається;
- element.previousSibling – властивість елементу, що повертає попередній елемент зі спільним батьком по відношенню до елемента, на якому вона викликається.

На рисунку 2.3 проілюстровані співвідношення між елементами веб-сторінки.

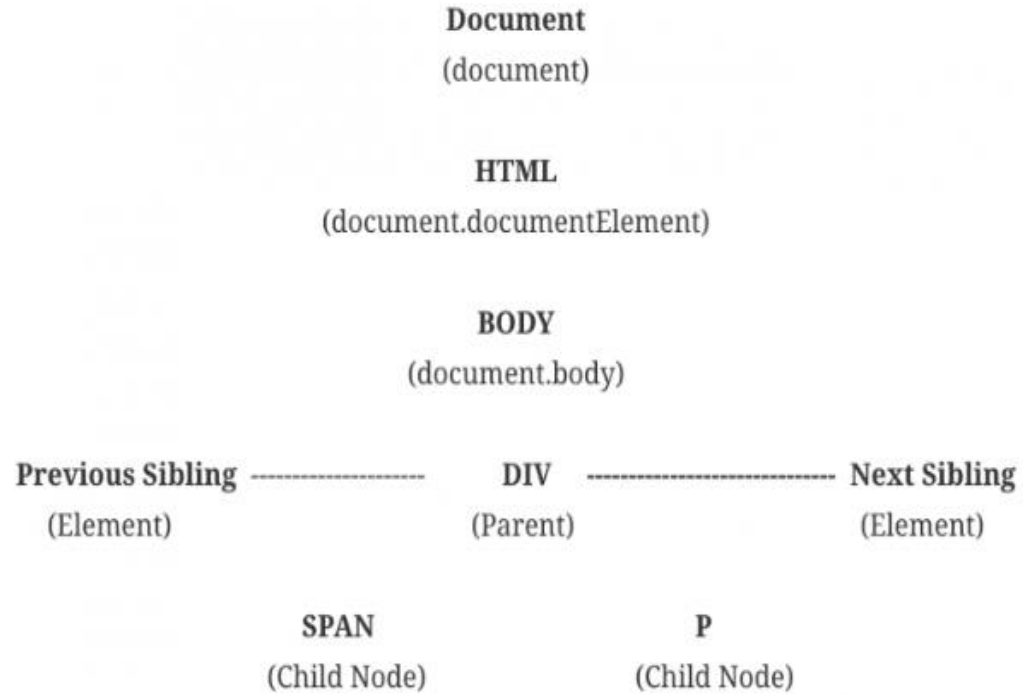

Рисунок 2.3 – співвідношення між елементами веб-сторінки [13]

У даному розділі було проаналізовано об'єктну модель документа, структуру веб-сторінки та методи об'єктної моделі документа для визначення структури документа. Були виділені такі методи об’єктної моделі документа, як:

– document.querySelector(), document.querySelectorAll() для пошуку елементу на веб-сторінці за його положенням відносто батьківських елементів;
– element.childNodes, element.parentNode, element.firstChild, element.nextSibling, element.previousSibling для отримання сусідніх елементів.

Використання даних методів необхідне для реалізації алгоритму пошуку елементів на веб-сторінках із динамічною структурою.

3 РОЗРОБКА МОДЕЛІ

У даному розділі розглядається розробка моделі, а саме розробка функціональних та нефункціональних вимог до системи, а також розробка та оптимізація алгоритму пошуку елементів на сторінці за сукупністю можливих шляхів до кореневого елементу.

3.1 Загальний підхід до поставленої задачі

Для вирішення поставленої задачі необхідно розробити алгоритм, що враховує всі можливі шляхи знайдення певного елементу на сторінці від кореневого елементу, оскільки положення кореневого елементу на сторінці завжди відомо, а врахування всіх можливих шляхів від кореневого елементу до обраного елементу дозволить врахувати будь-які переміщення обраного елементу на сторінці відносно кореневого, адже його все ще можна буде знайти на сторінці за деякими шляхами, за якими його можна було знайти раніше. Наприклад, якщо на сторінці знаходиться кореневий елемент <body>, всередині якого знаходиться елемент , що має атрибут class=”myDivClass”, всередині якого знаходиться інший елемент з класом “myDivClass2”, всередині якого знаходиться шуканий елемент <a>, то сторінка наступну структуру:

```
<div class="myDivClass" id="myDivId">
<div class="myDivClass2" id="myDivId2">
<a href="#" class="myAClass" id="myId" customAttr="customValue">
Click on me
</a>
</div>
</div>
```

У цьому випадку дійти до елементу <a> від кореневого елементу, використовуючи такий метод, як document.querySelector, можна наступними шляхами:

- body > div.myDivClass > div.myDivClass2 > a;
- body > div.myDivClass > a;
- body div.myDivClass > a;
- body div.myDivClass a;
- body div.myDivClass2 > a;
- body > div > div > a;
- body > div.myDivClass > div.myDivClass2 > *;
- body > div.myDivClass > div > *;
- body div.myDivClass2 > *;
- body > a;
- body > div.myDivClass > div.myDivClass2 > a;
- body > div.myDivClass > div.myDivClass2 > a.

У деяких з цих можливих шляхів від кореневого елементу до шуканого відсутній перший елемент , у деяких з них відсутній другий елемент , у деяких з них відсутні обидва елементи. Це означає, що частина з можливих шляхів дозволить знайти шуканий елемент за будь-яких змін структури сторінки [14]. Дане твердження проілюстроване на рисунку 3.1.

Також, частина шляхів до елементу може відкидати певні атрибути чи тег обраного елемента або його батьків. Нижче наведено приклади шляхів від кореневого елементу body до обраного елементу a:

- body > div > div > a;
- body > div > div > *.

У першому випадку елемент буде знайдено на сторінці лише за умови якщо він не змінить свій тег з <a> на будь-який інший. Другий шлях знайде елемент навіть якщо тег елементу буде змінено на, наприклад, .

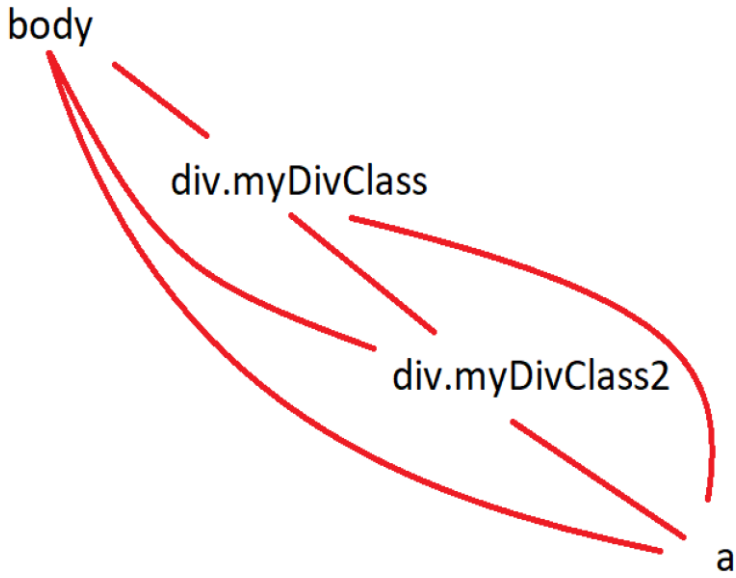

Рисунок 3.1 – Ілюстрація пошуку елементу через його батьківські елементи

3.2 Розробка функціональних вимог до системи

Система розробляється як сервіс для пошуку елементів на сторінці за наявності можливості зміни структурі сторінки. Функціональні вимоги включають можливість вибору елементу, який має бути ідентифікований у майбутньому, пошук раніше обраного елементу, збір та перегляд інформації про елемент та статистики. Повний список функціональних вимог наведено нижче:

- користувач повинен мати змогу обирати елемент на сторінці, до якого у майбутньому будуть додаватися підказки;
- користувач повинен мати змогу знайти даний елемент на сторінці у разі зміни структури сторінки;
- користувач повинен мати змогу переглядати статистику про кінцевих користувачів системи, що заходять на сторінку з певними елементами та бачать підказки на певних елементах;
- користувач має змогу сплатити підписку, щоб користуватися системою;
- користувач має змогу користуватися системою протягом пробного періоду безкоштовно;
- кінцевий користувач сайту користувача має доступ до елементів, що були обрані

користувачем на власному сайті, й має змогу бачити прикріплені користувачем до обраних елементів елементи.

3.3 Розробка нефункціональних вимог до системи

Список нефункціональних вимог системи наведено нижче:

- система має правильно знаходити елементи на сторінках, структура яких зазнала значних змін з часу вибору елементу на сторінці;
- точність пошуку елементів на сторінці не повинна бути нижче 80%;
- час пошуку елементу на сторінці не повинен перевищувати 0.5 секунди;
- система повинна працювати з HTML-документами будь-якого розміру;
- система повинна мати змогу ідентифікувати елемент незалежно від його тегу, атрибутів та положення на сторінці.

3.4 Алгоритм пошуку елементів на сторінці за сукупністю можливих шляхів до кореневого елементу

Використовуючи ідею з пункту 3.3 про прокладення усіх можливих шляхів від кореневого елементу до обраного елементу, можна розробити алгоритм, що прокладає дані шляхи, перевіряє, чи усі вони ведуть до шуканого елементу, й після зміни сторінки використовує усі ці шляхи для знайдення елементу, обираючи той елемент, на який вказує більшість з згенерованих шляхів. Нижче наведені етапи роботи алгоритму побудови усіх можливих шляхів до елементу:

1) знайти усі елементи між кореневим елементом та шуканим елементом;
2) використовуючи усі можливі комбінації множини проміжних елементів, скласти множину можливих шляхів, частина з яких включає лише частину проміжних елементів;
3) знайти усі атрибути для шуканого елементу та для кожного з проміжних елементів;

4) використовуючи комбінації множини для атрибутів можного з проміжних елементів та для атрибутів шуканого елементу, а також декартовий добуток множин, розширити множину можливих шляхів до шуканого елементу таким чином, що частина зі шляхів буде використовувати лише частину з атрибутів елементу;
5) продублювати шляхи для кожного з проміжних елементів та для шуканого елементу, відкидаючи вимоги відносно тегу елементу.

Нижче кожен з етапів алгоритму розглянутий більш докладно.

3.4.1 Знаходження усіх елементів між кореневим та шуканим елементом

Для знаходження усіх елементів між кореневим та шуканим елементом перш за все необхідно представити DOM-дерево у зручній для роботи з ним структурі. Таку структуру являє собою javascript-об’єкт, оскільки javascript – це єдина мова програмування, що доступна для виконання у веб-браузері. Цей об’єкт представляє собою небінарне дерево, коренем якого виступає кореневий елемент сторінки й дочірніми елементами кожного елементу дерева будуть виступати дочірні елементи даного елементу на сторінці. Також кожен елемент даного дерева буде мати наступні властивості:

1) type – тег елементу або ‘text’ для текстового елементу;
2) value – текстове наповнення для текстового елементу;
3) attributes – масив атрибутів елементу;
4) children – масив посилань на дочірні елементи;
5) parent – посилання на батьківський елемент.

Використовуючи дане дерево, можливо, маючи доступ до обраного елементу, з використанням властивості parent циклічно пройти по батьківським елементам від обраного елементу до кореневого, й таким чином знайти усі елементи між обраним елементом та кореневим елементом.

Також, кожному елементу на сторінці має бути привласнений унікальний

ідентифікатор, за яким його можна буде знайти в дереві протягом роботи алгоритму. Для цього кожному елементу буде привласнений тимчасовий атрибут find-element-id, що буде дорівнювати числу, від одного до довжини з масиву усіх елементів на сторінці мінус один.

3.4.2 Складення множини можливих шляхів до елементу, використовуючи комбінацію множини проміжних елементів

Для складення множини можливих шляхів до елементу, частина з яких не враховує деякі з проміжних елементів, необхідно застосувати поняття комбінації та булеану з комбінаторики та теорії множин [15].

Комбінація деякої множини – це спосіб вибору підмножин з даної множини.

Булеан певної множини – це сукупність усіх підмножин даної множини. Булеан для множини з трьох елементів S = {A, B, C} дорівнює P(S) = { {}, {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C} }.

Використовуючи поняття булеану, можна визначити множину можливих шляхів до елементу як булеан множини елементів між кореневим та обраним елементом.

3.4.3 Знаходження усіх атрибутів для шуканого елементу та для кожного з проміжних елементів

Для знаходження атрибутів кожного елементу можна використовувати API об’єктної моделі документа, що дозволяє отримати список атрибутів елементу, викликавши метод element.getAttributes().

Список атрибутів, що були отримані з використанням методу element.getAttributes(), буде записаний до властивості attributes елементу дерева елементів, що описане в пункті 3.4.1.

3.4.4 Розширення множини можливих шляхів до шуканого елементу, використовуючи тег та атрибути елементів

Представивши атрибути обраного елементу та проміжних елементів у вигляді множини (або, іншими словами, масиву), можливо розширити множину можливих шляхів до обраного елементу, перетворивши кожний елемент даної множини на декартів добуток булеанів атрибутів, що належать елементам даної множини.

Декартів добуток для сімейства множин – це впорядкована множина усіх можливих комбінацій елементів з даних множин [16]. Ілюстрацію декартового добутку для двох множин A = (x, y, z) та B = (1, 2, 3) зображено на рисунку 3.2.

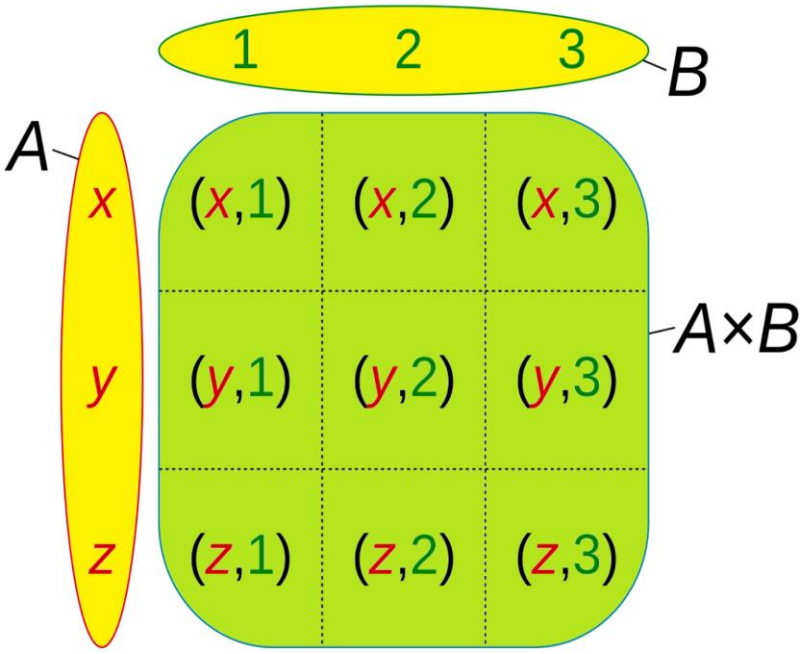

Рисунок 3.2 – Ілюстрація декартового добутку для двох множин A=(x, y, z) та B=(1, 2, 3) [16]

Отже, для двох множин A=(x, y, z) та B=(1, 2, 3) декартовий добуток AxB буде дорівнювати AxB=((x, 1), (x, 2), (x, 3), (y, 1), (y, 2), (y, 3), (z, 1), (z, 2), (z, 3)).

Використавши булеан та декартовий добуток множин атрибутів та проміжних елементів, ми отримали множину шляхів до елемента, що враховує усі комбінації з множини проміжних елементів та множин атрибутів кожного з проміжних елементів.

Це дозволяє знайти елемент за умови будь-якої зміни його положення у структурі сторінки або ж у разі будь-якої зміни атрибутів елементу, за якої даний елемент міг би бути розцінений людиною як ідентичний до своєї попередньої версії.

Маючи множину усіх можливих шляхів знаходження обраного елементу від кореневого елемента, можна описати алгоритм пошуку елементу на сторінці наступним чином:

1) використати на зміненій сторінці кожен із раніше знайдених шляхів до елементу;
2) підрахувати кількість знаходжень шляхами кожного з елементів на сторінці;
3) обрати той елемент на сторінці, якому відповідає найбільша кількість шляхів знаходження елемента.

3.5 Оптимізація алгоритму пошуку елементу на сторінці

Для оптимізації алгоритму пошуку елементу на сторінці необхідно мати змогу зменшувати кількість шляхів, за якими можна знайти елемент. Для цього необхідно впровадити алгоритм оцінки важливості врахування певного проміжного елементу чи атрибуту при побудові множини можливих шляхів до елементу. Для цього можливо використати наступний алгоритм:

1) видалити елемент з множини проміжних елементів (або атрибут певного елементу з множини його атрибутів);
2) повторити етапи 2-5 алгоритму пошуку елементу (пункт 3.4);
3) змінити структуру сторінки шляхом переміщення елементів або зміни їхніх атрибутів випадковим чином;
4) оцінити точність знаходження елементу.

Використовуючи даний алгоритм, можна значно зменшити кількість можливих шляхів знаходження елементу, що пропорційно збільшує швидкість пошуку елемента.

У даному розділі було розроблено алгоритм пошуку елементу на сторінці після зміни структури сторінки за допомогою прокладання різних шляхів до елементу від

кореневого елементу. При написанні цього алгоритму були використані такі елементи комбінаторики, як булеан та декартовий добуток множин, а також наступні методі об’єктної моделі документа:

- метод document.querySelector() для пошуку елементу за його положенням відносно інших елементів на сторінці;
- властивості елемента element.childNodes, element.firstChild, element.nextSibling, element.previousSibling для доступу до елементів, що розташовані поруч з поточним елементом.

Отриманий алгоритм складається з наступних етапів:

1) знаходження всіх елементів між кореневим елементом та шуканим елементом;
2) використання всіх можливих комбінацій множини проміжних елементів для складання множини можливих шляхів, частина з яких включає лише частину проміжних елементів;
3) знаходження всіх атрибутів для шуканого елементу та для кожного з проміжних елементів;
4) використання комбінацій множини для атрибутів можного з проміжних елементів та для атрибутів шуканого елементу, а також декартового добутку множин для розширення множини можливих шляхів до шуканого елементу таким чином, що частина зі шляхів буде використовувати лише частину з атрибутів елементу;
5) дублювання шляхів для кожного з проміжних елементів та для шуканого елементу шляхом відкидання вимог відносно тегу елементу.

4 РОЗРОБКА СИСТЕМИ

У даному розділі наводиться обгрунтування оформлення деяких додатків до розроблюваної магістерської дисертації (діаграми сценаріїв, структурної та функціональної схеми), а також проводиться аналіз існуючих елементів та технологій, що мають бути використані для розробки системи пошуку елементів на веб-сторінках із динамічною структурою.

4.1 Вибір та обґрунтування елементів та технологій

У даному підрозділі наводиться аналіз існуючих елементів та технологій, що використані під час реалізації системи пошуку елементів на веб-сторінках зі змінюваною структурою.

4.1.1 Вибір серверної мови програмування

Ключовим критерієм для вибору мови програмування як технології для розробки системи ідентифікації елементів на сторінках зі змінюваною структурою є можливість веб-серверу обробляти велику кількість запитів одночасно без значної затримки.

Багато мов програмування використовують багатопоточну архітектуру “запит-відповідь”, коли для кожного з запитів клієнта, що обробляються одночасно, виділяється потік для обробки запиту.

Структура сервера з багатопоточною архітектурою наведена на рисунку 4.1.

На рисунку 4.1 зображено структуру веб-серверу, до якого звертаються клієнти під номерами від 1 до n. Також даний сервер має пул потоків під номерами від 1 до m. Якщо n стає більшим, ніж m (що стається майже завжди), сервер збільшує кількість використовуваних потоків m до максимально можливої. Коли ж кількість одночасних запитів до сервера стане більшим за максимальну кількість потоків, що може бути

виділена на обробку запитів, частина запитів буде повинна чекати початку своєї обробки в черзі запитів, місце в якій буде з’являтися після завершення обробки попередніх запитів. Якщо ж потоки, що обробляються у даний час, використовують операції вводу-виводу, такі як запис чи зчитування файлів або запити до бази даних, то потокам, що знаходяться в черзі, необхідно чекати завершення усіх цих операцій.

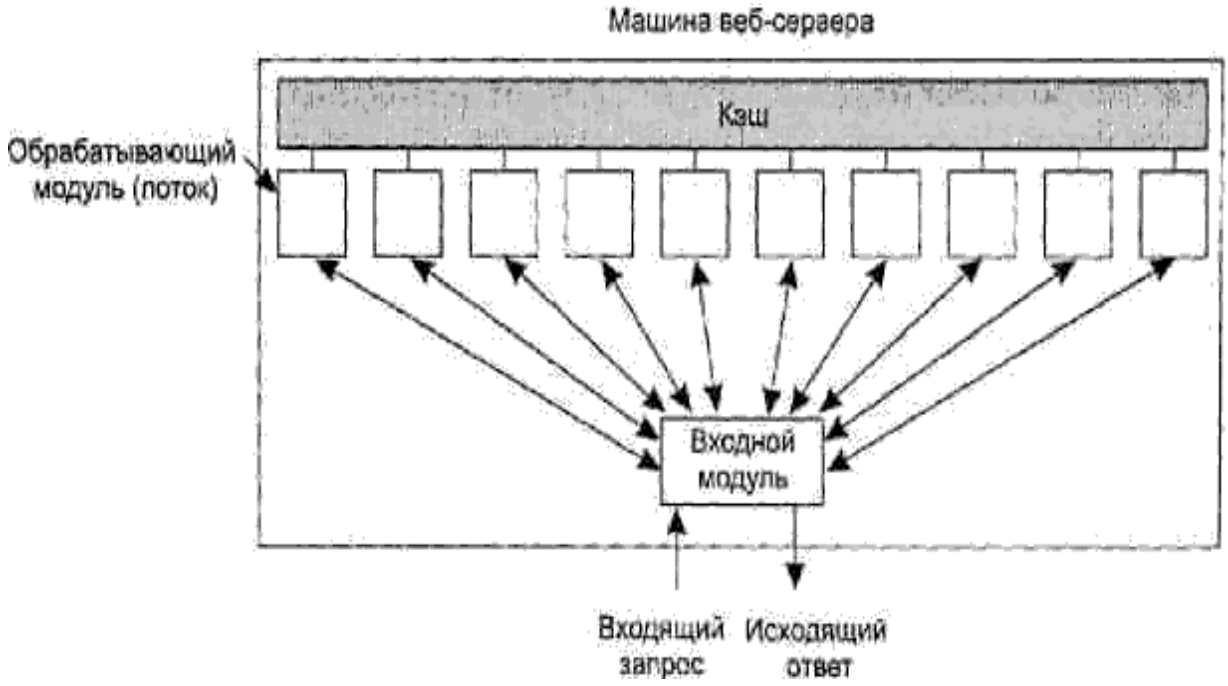

Рисунок 4.1 – структура багатопоточного веб-сервера [17]

Також суттєвим недоліком такої моделі є те, що кожен потік займає споживає певну кількість пам’яті, що стає суттєвою за великої кількості потоків, що запущені одночасно. Отже, недоліками багатопоточної архітектури є:

- складність обробки все більшої кількості одночасних запитів;
- витрати пам’яті на одночасну роботу багатьох потоків;
- необхідність для запитів у черзі чекати завершення операцій вводу-виводу в оброблюваних потоках;
- витрати часу на обробку задач вводу-виводу.

Альтернативою до багатопоточної архітектури є однопоточна архітектура з застосуванням циклу обробки подій (event loop), що застосовується у мові програмування Node.js [17]. Дана архітектура використовує механізм зворотного

виклику Javascript, що дозволяє обробляти велику кількість запитів в одному потоці, а операції вводу виводу зробити неблокуючими потік. Дана архітектура виглядає наступним чином (рисунок 4.2):

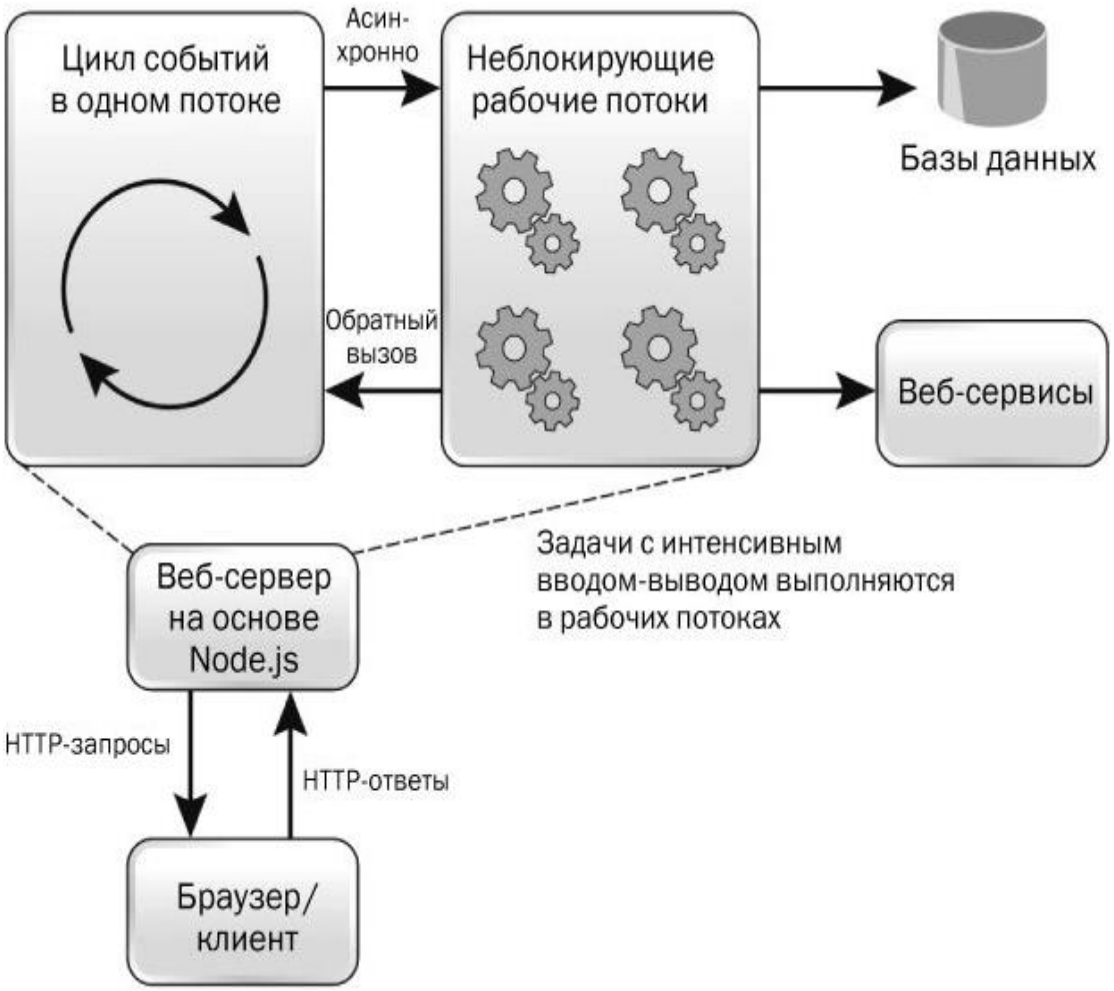

Рисунок 4.2 – Структура веб-сервера, що завдяки Node.js використовує однопоточну архітектуру [17]

Перевагами даної архітектури є:

- більш проста обробка все більшої кількості одночасних запитів;
- відсутність витрат пам’яті на одночасну роботу багатьох потоків;
- відсутність необхідності для запитів у черзі чекати завершення операцій вводу-виводу в оброблюваних потоках;
- відсутність витрат часу на обробку задач вводу-виводу, оскільки у той час як один запит чекає на виконання операції вводу-виводу, у потоці виконуються

операції іншого запиту [18].

Отже, для розроблюваної системи прийняте рішення використовувати Node.js, оскільки дана технологія реалізує однопоточну архітектуру обробки багатьох запитів.

4.1.2 Вибір типу структури СУБД для серверної частини системи

Коли необхідно вибрати СУБД, головне питання зазвичай полягає у виборі реляційної (SQL) або нереляційної (NoSQL) структури. В обох варіантів є свої переваги, а також кілька ключових особливостей, які варто мати на увазі при виборі.

Нижче перераховані основні відмінності між реляційними та нереляційними системами управління базами даних:

- мова написання запитів. Якщо для реляційних баз даних існує єдина мова, що використовується для написання запитів, то для нереляційних такої мови не існує. Реляційні бази даних використовують структурований мову запитів (Structured Query Language, SQL) для визначення і обробки даних. З одного боку, це відкриває великі можливості для розробки: SQL є одним з найбільш гнучких і поширених мов запитів, так що його вибір дозволяє мінімізувати ряд ризиків, і використання єдиної мови запитів буде особливо до речі, якщо необхідно здійснювати складні запити до бази даних. З іншого боку, в SQL є ряд обмежень. Побудова запитів на цій мові зобов'язує визначати структуру даних, і подальша зміна структури даних може бути згубним для всієї системи. Нереляційні бази даних, в свою чергу, пропонують динамічну структуру даних, яка може зберігатися кількома способами: орієнтовано по колонках, документо-орієнтовано, в вигляді графів або на основі пар «ключ-значення». Така гнучкість означає, що користувач може створювати документи, не задаючи їхню структуру наперед, що кожен документ може мати власну структуру, а також що кожна окрема база даних може мати власний синтаксис для написання запитів;
- масштабування. У більшості випадків SQL бази даних вертикально масштабовані, тобто розробник може збільшувати навантаження на окремо

взятий сервер, нарощуючи потужність центральних процесорів, обсяги ОЗУ або системи зберігання даних. А NoSQL бази даних горизонтально масштабовані. Це означає, що користувач може збільшувати трафік, розподіляючи його або додаючи більше серверів до власної СУБД. Якщо проводити аналогію зі зведенням будівель, то вертикальне масштабування дозволяє додавати більше поверхів до будинку, а горизонтальне масштабування дозволяє додавати більше будівель на вулицю. У другому випадку, система може стати більш потужною, тому структура NoSQL має перевагу над SQL для великих або постійно мінливих структур даних;

– структура. У реляційних СУБД дані представлені у вигляді таблиць, в той час як в нереляційних – у вигляді документів, пар «ключ-значення», графів або wide-column сховищ. Це робить SQL бази даних кращим вибором для застосунків, які передбачають транзакції з декількома записами – як, наприклад, система облікових записів (рисунок 4.3);
– цілісність даних. У реляційних СУБД наявні зв’язки між таблицями, реалізовані через зовнішні ключі, що дозволяє запобігти випадковому запису до таблиці даних, що не мають зв’язаних з ними даних у інших таблицях, у той час як вони повинні їх мати. У NoSQL розробник не проектує свою базу даних на основі зв’язків між сутностями даних, а розробляє свою базу даних, грунтуючись на запитах, які він буде виконувати проти неї. При цьому він керується тими ж самими критеріями, які він би використовував для денормалізації реляційної бази даних: нереляційні бази даних використовуються, якщо для даних важливіше мати згуртованість, ніж бути цілісними й ненадмірними. Але це неминуче призводить для оптимізації бази даних для одного типу запиту (наприклад, коментарів будь-якого користувача до даної статті) за рахунок втрати у швидкості для інших типів запитів (коментарів до будь-якої статті даного користувача). Якщо у розробленій програмі є необхідність обох типів запитів бути однаково оптимізованими, дані не слід денормалізувати. І також не слід використовувати рішення NoSQL, якщо є необхідністість у даних бути

зв’язаними, цілісними та ненадмірними. У разі денормалізації та надмірності існує ризик, що надлишкові набори даних вийдуть із синхронізації один з одним. Це називається аномалією. У разі використання нормалізованої реляційної базу даних, RDBMS може запобігти аномалії. У денормалізованій базі даних або в NoSQL лише логіка коду, написаного програмістом, може запобігти даній ситуації.

У таблиці 4.1 наведено порівняння структур SQL та NoSQL для систем управління базами даних.

Таблиця 4.1 – Порівняння структур SQL та NoSQL

Можливості, які отримує розробник при використанні СУБД	Реляційні	Нереляційні
Зв’язаність даних	+	-
Оптимізація під будь-які види запитів	+	-
Оптимізація під конкретні види запитів	-	+
Масштабування	вертикальне	горизонтальне
Універсальна мова написання запитів	+	-
Орієнтованість на написання складних запитів	+	-
Збереження багатьох видів сутностей в одному документі	-	+

Після проведення аналізу даної таблиці встановлено, що для розроблюваної системи реляційна СУБД буде кращим вибором, оскільки:

- первинна необхідність системи – бути цілісною з боку даних й не давати можливості розробнику записати неправильні дані до бази даних;

- для збору статистики необхідна можливість швидко писати складні запити до бази даних;
- оптимізованість для більшої кількості запитів у даному випадку більш важлива, ніж краща оптимізованість для певних запитів, оскільки планується використання сторонніх систем кешування, що дозволить зменшити кількість найбільш затратних з боку ресурсів запитів до серверу;
- відсутня наявність потреби у горизонтальному масштабуванні, оскільки з використанням сторонніх систем кешування дозволить зменшити навантаження на сервер.

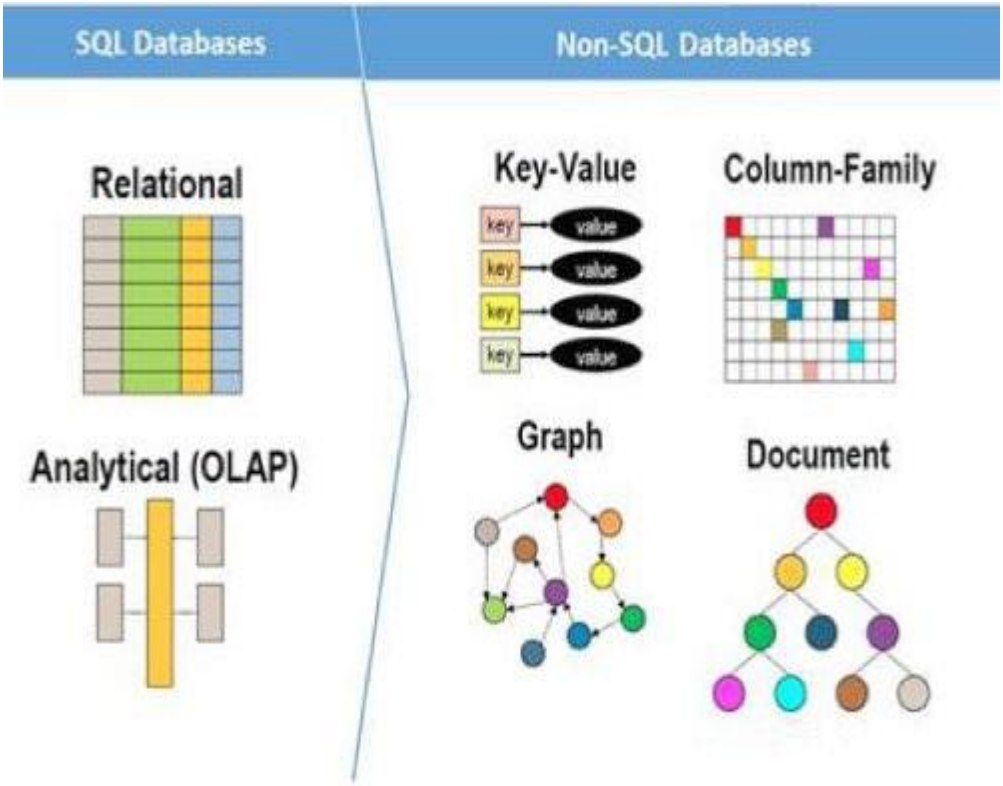

Рисунок 4.3 – Графічне представлення структури SQL та NoSQL баз даних [19]

4.1.3 Вибір реляційної СУБД для серверної частини системи

За даними сайту DB-Engineers, найбільш популярними реляційними СУБД є PostgreSQL, MySQL, MSSQL, Oracle [20].

Оскільки лише PostgreSQL та MySQL є повністю безкоштовними та наявними

у вільному доступі, доцільно порівняти спочатку ці дві системи управління базами даних.

Нижче наведена таблиця порівняння PostgreSQL та MySQL (Таблиця 4.2).

Таблиця 4.2 – Порівняння PostgreSQL та MySQL

Можливості, що отримує розробник при використанні СУБД	PostgreSQL	MySQL
Реплікація	фізична, master-slave	логічна, master-master та master-slave, має проблеми зі стабільністю та швидкістю роботи
Документація	детальна й повноцінна	неповноцінна, є необхідність шукати опис наявних функцій на сторонніх сайтах
Можливості, що отримує розробник при використанні СУБД	PostgreSQL	MySQL
Підтримка стандартів SQL	відповідає стандартам SQL-92, SQL-98, SQL-2003, ведеться робота над SQL-2011	не відповідає жодним стандартам
Наявність оптимізатора запитів	наявний	відсутній, є лише парсер та нормалізації запитів

Продовження таблиці 4.2

Можливості, що отримує розробник при використанні СУБД	PostgreSQL	MySQL
Складність налаштування та адміністрування	Складне, за рахунок великої кількості налаштувань	просте, за рахунок невеликої кількості можливостей та налаштувань
Створення індексів неблокуючим шляхом (без блокування інших операцій)	наявне	відсутнє
Паралельне планування запитів	наявне	відсутнє
Часткові індекси	наявні	відсутні
Наслідування таблиць	наявне	відсутнє
Перевантаження функцій	наявне	відсутнє

Продовження таблиці 4.2

Можливості, що отримує розробник при використанні СУБД	PostgreSQL	MySQL
Підтримувані операційні системи	FreeBSD, HP-UX, Linux, NetBSD, OpenBSD, OS X, Solaris, Unix, Windows	FreeBSD, Linux, OS X, Solaris, Windows
Схема даних	наявна	наявна
Типи даних	підтримуються	підтримуються
Вторинні індекси	наявні	наявні
Підтримка SQL	наявна	наявна
Наявні API	ODBC, JDBC, нативна C бібліотека, стримінгове API, ADO.NET	Власне нативне API, ADO.NET, JDBC, ODBC
Підтримувані мови програмування	.NET, C, C++, Delphi, Java, JavaScript (Node.js), Perl, PHP, Python, Tcl	Ada, C, C#, C++, D, Delphi, Eiffel, Erlang, Haskell, Java, JavaScript (Node.js), Objective-C, OCaml, Perl, PHP, Python, Ruby, Scheme, Tcl

Продовження таблиці 4.2

Можливості, що отримує розробник при використанні СУБД	PostgreSQL	MySQL
Розділення даних	наявне розділення з використанням диапазонів, списків та хешів	горизонтальне розділення, шардування за допомогою кластера MySQL або фабрики MySQL
Мова написання	C	C та C++

Як видно з таблиці, у MySQL майже немає переваг над PostgreSQL, у той час як PostgreSQL має безліч переваг над MySQL [21].

У таблиці нижче наведено переваги та недоліки PostgreSQL, Oracle та MSSQL, з яких лише PostgreSQL є повністю безкоштовною (таблиця 4.3).

Таблиця 4.3 – Переваги та недоліки PostgreSQL, Oracle, MSSQL

Властивість СУБД	PostgreSQL	MSSQL	Oracle
Ціна	безкоштовна	$1,859 за версію SQL 2017, що буде запущена на одному ядрі процесора	$17,500 за стандартну ліцензію, що буде запущена на одному процесорі

Продовження таблиці 4.3

Властивість СУБД	PostgreSQL	MSSQL	Oracle
Реплікація	складається у формі звітів, має як вбудований безкоштовний варіант, так і платні сторонні. Вбудовано лише реплікацію формату master-slave	наявна можливість копіювання усіх видів даних. Це може бути реплікація логів, дзеркальне відображення, знімок, транзакція, злиття, наявна підтримка slave-серверів, що не працюють на SQL Server	підтримує як master-master, так і master-slave реплікацію
Розділення даних	наявне розділення з використанням диапазонів, списків та хешів	горизонтальне	горизонтальне
Скрипти на стороні сервера	функції, об’явлені користувачем	Transact SQL, .NET languages, R, Python та Java	PL/SQL скрипти
Наявні API	ODBC, JDBC, нативна C бібліотека, стримінгове API, ADO.NET	OLE DB, TDS, ADO.NET, JDBC, ODBC	JDBC ODBC Oracle call interface ODP.NET

Продовження таблиці 4.3

Властивість СУБД	PostgreSQL	MSSQL	Oracle
Вторинні моделі баз даних	документи, пари “ключ-значення”	документи, графи	документи, пари “ключ-значення”, сховище RDF, графи
Підтримка	безкоштовна, та розгляд проблеми, що виникла, може зайняти багато часу	входить до вартості ліцензії	коштує майже 25% від вартості ліцензії
Швидкість виходу нових версій	4-5 років	2-3 роки	2-3 роки
Масштабування	наявне безкоштовне масштабування з використанням кластерів	потребує Enterprise ліцензії у разі необхідності значного масштабування	потребує Enterprise ліцензії у разі необхідності значного масштабування
Швидкість виконання транзакцій	для більшості випадків не відрізняється від MSSQL та Oracle	для більшості випадків не відрізняється від PostgreSQL та Oracle	для більшості випадків не відрізняється від PostgreSQL та MSSQL

Продовження таблиці 4.3

Властивість СУБД	PostgreSQL	MSSQL	Oracle
Підтримувані операційні системи	FreeBSD, HP-UX, Linux, NetBSD, OpenBSD, OS X, Solaris, Unix, Windows	Windows, Linux	AIX, HP-UX, Linux, OS X, Solaris, Windows, z/OS
Підтримувані мови програмування	.NET, C, C++, Delphi, Java info, JavaScript (Node.js), Perl, PHP, Python, Tcl	C#, C++, Delphi, Go, Java, JavaScript (Node.js), PHP, Python, R, Ruby, Visual Basic	C, C#, C++, Clojure, Cobol, Delphi, Eiffel, Erlang, Fortran, Groovy, Haskell, Java, JavaScript, Lisp, Objective C, OCaml, Perl, PHP, Python, R, Ruby, Scala, Tcl, Visual Basic
Зберігання даних лише у пам’яті	відсутнє	наявне	наявне
Доступність тільки як облачного сервісу	ні	ні	ні
Мова написання	C	C та C++	C++
Підтримка XML	так	так	так

Продовження таблиці 4.3

Властивість СУБД	PostgreSQL	MSSQL	Oracle
Вторинні індекси	так	так	так
Підтримка різних типів даних	так	так	так
Підтримка схеми даних	так	так	так
Підтримка SQL	так	так	так
Підтримка тригерів	так	так	так
Концепція транзакцій	ACID	ACID	ACID
Підтримка паралельного виконання запитів	так	так	так
Концепція створення користувачів та надання користувачам прав доступу	згідно стандартів SQL	згідно стандартів SQL	згідно стандартів SQL

Як видно з таблиці 4.13, PostgreSQL володіє властивостями, що не набагато гірші, а у деяких випадках кращі, ніж MSSQL та Oracle, при цьому має вільну ліцензію та розповсюджується повністю безкоштовно [22]. Отже, PostgreSQL є оптимальним варіантом для використання у якості СУБД серверної частини розроблюваної системи пошуку елементів на веб-сторінках.

4.1.4 Вибір системи збірки проекту для клієнтської частини системи

Сьогодні використання засобів автоматизації робочого процесу, таких як Gulp або Grunt, є важливим для кожного проекта, що стосується розробки веб-застосунків. Вони допомагають вдосконалити швидкість розробки та роботи додатку та автоматизувати рутинні операції розробника, наприклад попередня обробка CSS, стиснення зображень, мінімізація коду тощо. На сьогодні існує такий інструмент, як Webpack, який є потужним менеджером залежностей проекту, а також корисний для поєднання різних файлів разом у граф залежностей. Однак він також містить функції виконання задач збірки проекту, що робить його конкурентом Grunt або Gulp, що також є інструментами для виконання задач при обробці коду розробника [23].

Існує велика кількість повсякденних завдань, з якими стикається кожен розробник. Але розробники можуть писати програми, що можуть автоматизувати повсякденні завдання, то автоматизація робочого процесу – це необхідна вимога для швидкого та зручного написання будь-якої великої системи.

Приклади повсякденних операцій, які має робити кожен розробник клієнтських частин веб-застосунків:

- обробка css-стилів: додавання властивостей для підтримки різних браузерів, а також компіляція коду з препроцесорів мови CSS (таких як SASS та LESS) до css-коду;
- мініфікація коду;
- видалення команд для дебагу та виводу в консоль;
- запуск тестів;
- перевірка стилю написання коду;
- стиснення зображень.

За допомогою засобів автоматизації, таких як засіб виконання команд Gulp, та досить простого коду, розробник системи може автоматизувати ці щоденні завдання. Хоча вони не є обов’язковими для розробки програмного забезпечення веб-застосунків, засоби виконання завдань можуть значно підвищити продуктивність

застосунків та заощадити час на розробку. Більше того, засоби виконання завдань допомагають спростити роботу та підвищити продуктивність розробки системи, а також зменшити кількість людських помилок.

Наприклад, як засоби виконання завдань, так і менеджери залежностей проекту, такі як Webpack, здатні упакувати декілька файлів JS в один файл. Це зменшує кількість запитів до сервера, що призводить до кращої продуктивності. Результат роботи даних засобів автоматизації – це оптимізований код, готовий до використання у виробничих умовах.

Нижче наведено основні відмінності між найбільш популярними засобами виконання завдань – Grunt та Gulp [23] (таблиця 4.4).

Таблиця 4.4 – Відмінності між Grunt та Gulp [23]

Характеристика	Grunt	Gulp
Інструмент виконання завдань	тимчасові файли на локальному диску	оперативна пам’ять
Формат вхідних даних	JSON	Javascript
Кількість одночасно виконуваних завдань	одне	багато
Вимагає знань у Node.js	ні	так
Дозволяє визначати залежності завдань одне від одного	ні	так

На рисунку 4.4 зображено процес виконання завдань при використанні Grunt. Як видно з рисунку, після виконання першого завдання у списку Grunt збережує проміжний результат у тимчасовій папці .tmp/, й наступна команда користується цими проміжними результатами, записаними на диск.

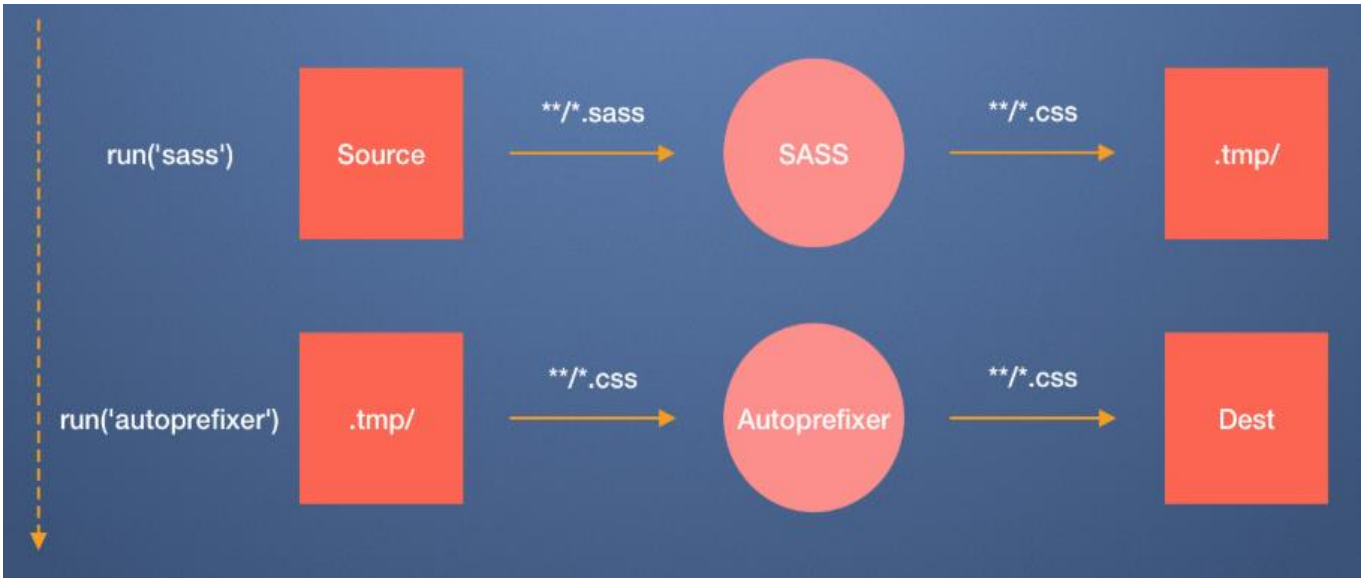

Рисунок 4.4 – Процес виконання завдань при використанні Grunt [24]

На відміну від Grunt, процес виконання завдань з використанням Gulp виглядає наступним чином (рисунок 4.5):

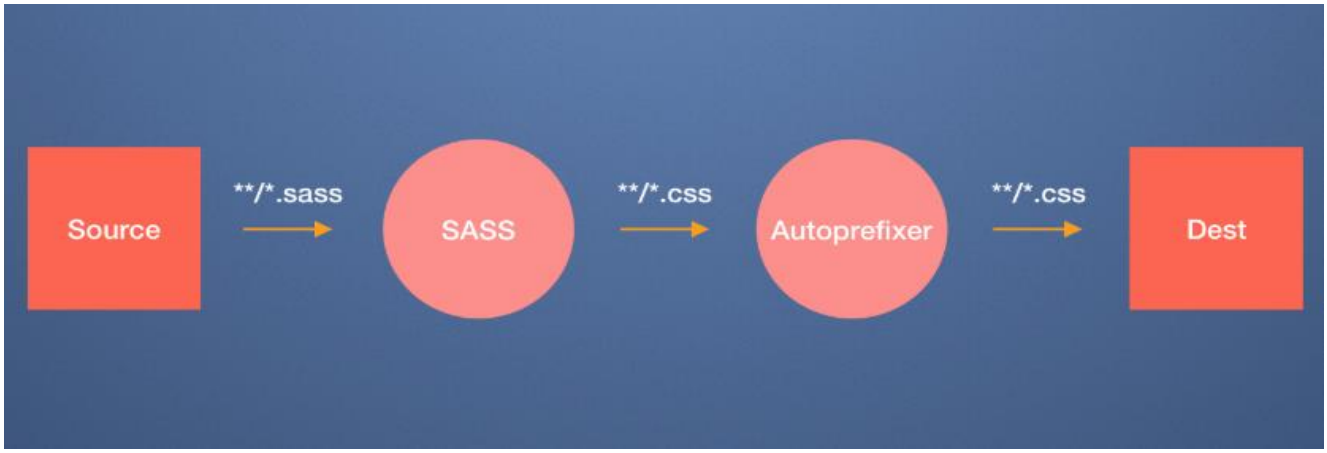

Рисунок 4.5 – Процес виконання завдань при використанні Gulp [24]

Як видно з рисунку, Gulp для збереження проміжних результатів виконання команд не потребує додаткового часу та ресурсів. Оскільки оперативна пам’ять є набагато швидшою за пам’ять на диску, Gulp є більш швидким засобом виконання завдань, ніж Grunt. Це також підтверджується порівнянням швидкості Grunt та Gulp, проведеним TMWtech [25].

Плагіни для Gulp орієнтовані на виконання єдиної мети, а також на кодування

для виконання того, що потрібно розробнику. Отже, як правило, зробити власне розширення для Gulp є набагато простішим для розробника. Крім того, код при використанні Gulp виглядає простіше, і його простіше читати. Однак, для використання Gulp необхідні знання Node.js.

Написання задач для Grunt схоже скоріше на задавання конфігурації, ніж на кодування, і його плагіни вирішують численні завдання. Налаштування плагінів Grunt може зайняти трохи більше часу.

Менеджери залежностей проекту дозволяють поєднати багато статичних файлів в один для зменшення кількості HTTP-запитів та оптимізувати продуктивність розроблюваної системи. Два найпопулярніших у даний час менеджера залежностей це Webpack та Browserify.

Browserify це інструмент для виконання Node.js коду в браузері, що дозволяє пакувати декілька статичних файлів в один. Він не має функцій виконання завдань, і для виконання завдань потребує взаємодії з інструментами виконання завдань, такими як Grunt або Gulp. Це робить незручним його використання, оскільки для кожного нового розроблюваного проекту в розробника є необхідність налаштувати спочатку Browserify, потім Grunt або Gulp, а потім налаштувати їх взаємодію. Набагато зручнішим є використання інструмента, що поєднує в собі функції менеджера залежностей проекту та інструмента виаонання завдань. Таким інструментом є Webpack [26].

Webpack приймає файли проекту на вході (наприклад, .js, .css., .scss, зображення тощо), які мають певні залежності між собою, і об'єднує їх у пакетну версію цих файлів, яку потім буде читати браузер. Порівняно з Gulp, Webpack виконує схожі функції, тому Webpack можна назвати інструментом виконання завдань. Таким чином, не потрібно використовувати Gulp та Webpack разом. Однак також Webpack здатен виконувати наступні функції:

- виявлення коду, що не використовується, й виключення даного коду з залежностей проекту;
- завантаження коду лише тоді, коли він потрібен користувачеві веб-сторінки;
- моніторинг змін файлів розроблюваної системи;

- перетворення коду Javascript з версії ES6 на версію ES5 для підтримки старими браузерами;
- використання конструкції require() для не кодових файлів;
- запуск веб-сервера.

Процес роботи Webpack проілюстровано на рисунку 4.6.

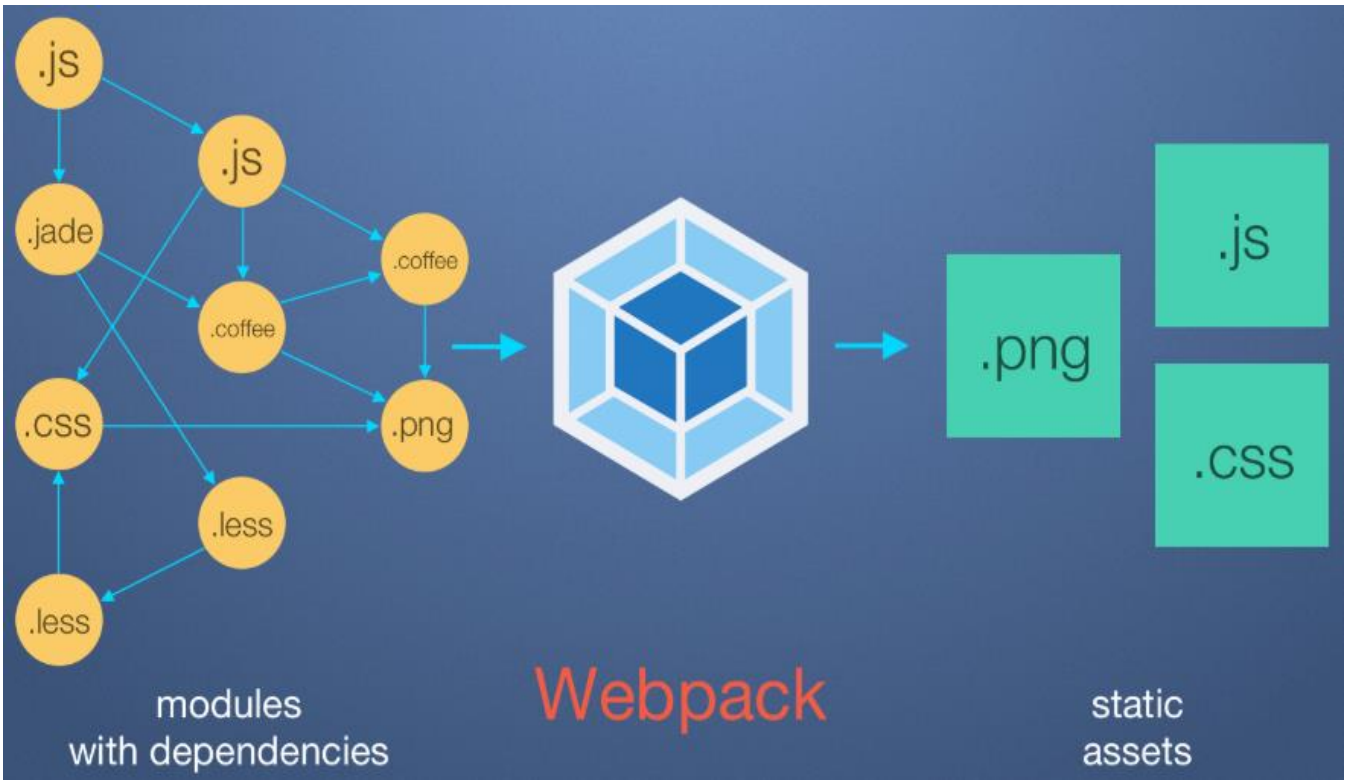

Рисунок 4.6 – Процес роботи Webpack [26]

Отже, Webpack є найбільш зручним інструментом, що поєднує в собі функції інструменту запуску завдань та менеджера залежностей проекту й що не потребує використання додаткових інструментів на відміну від альтернативних інструментів.

4.1.5 Вибір системи кешування для зняття навантаження з бази даних

Кешування даних необхідне для зняття навантаження з серверу шляхом перенесення навантаження з жорсткого диску та мережі до оперативної пам’яті, що, хоча й поступається жорсткому диску в надійності, значно виграє в швидкості у

порівнянні з ним [27].

На сьогодні існує велика кількість засобів, що дозволяють вирішити задачу кешування даних. У таблиці нижче наведено деякі з найбільш популярних інструментів для зберігання даних у вигляді “ключ-значення” у пам’яті (таблиця 4.5).

Таблиця 4.5 – Порівняння Ehcache, Memcached та Redis [28]

Характеристика	Ehcache	Memcached	Redis
Підтримувані операційні системи	Будь-які, що дозволяють запустити JVM	FreeBSD, Linux, OS X, Unix, Windows	BSD, Linux, OS X, Windows
Підтримка типізації	так	ні	часткова
Підтримка вторинних індексів	ні	ні	так
Характеристика	Ehcache	Memcached	Redis
Підтримувані мови програмування	Java	.Net, C, C++, ColdFusion, Erlang, Java, Lisp, Lua, OCaml, Perl, PHP, Python, Ruby	C, C#, C++, Clojure, Crystal, D, Dart, Elixir, Erlang, Fancy, Go, Haskell, Haxe, Java, JavaScript (Node.js), Lisp, Lua, MatLab, Objective-C, OCaml, Pascal, Perl, PHP

Продовження таблиці 4.5

Характеристика	Ehcache	Memcached	Redis
Підтримка тригерів	так	ні	ні
Реплікація	так	ні	ні
Шардування	так	ні	ні
Запис даних на диск	так	ні	ні
Підтримка паралельного маніпулювання даними	так	так	так

Як видно з таблиці 4.5, Memcached є досить легким інструментом, що, однак, поступається конкурентам за бататьма пунктами. Порівнюючи Ehcache та Redis, можна зазначити, що дані інструменти є досить схожими за рівнем можливостей, що вони надають розробнику, однак надоліком Ehcache є підтримка лише однієї мови програмування – Java [29]. Оскільки ж для розробки веб-серверу була обрана технологія Node.js, Redis є більш універсальним та підходящим рішенням для розроблюваної системи.

4.1.6 Вибір CDN для зменшення часу очікування завершення запитів зі сторони клієнта та зменшення навантаження на сервер

CDN (Content delivery network) це сукупність географічно розподілених серверів, які працюють разом, щоб забезпечити швидке завантаження інтернет-сторінок у будь-якій точці земної кулі. CDN дозволяє швидко передавати ресурси,

необхідні для завантаження ресурсів у мережі інтернет, включаючи HTML-сторінки, файли javascript, таблиці стилів, зображення та відео. Популярність служб CDN продовжує зростати, і сьогодні більшість веб-трафіку обслуговується через CDN, включаючи трафік з основних сайтів, таких як Facebook, Netflix та Amazon [30].

Правильно налаштований CDN також може допомогти для захисту веб-сайту від найбільш розповсюджених видів зловмисних атак, наприклад, таких як DDoS (Distributed Denial of Service) атак.

На відміну від веб-хостингу, CDN не вміщає в себе веб-сервер з логікою системи та базою даних, тому CDN не може замінити потребу в належному веб-хостингу. CDN допомагає кешувати веб-сторінки, що покращує ефективність веб-сайту. Багато веб-сайтів не здатні досягти потрібного їм рівня продуктивності за допомогою традиційних хостингових послуг, саме тому вони вибирають CDN.

Завдяки використанню кешування для зменшення навантаження на хостинг, запобіганню перебоям у обслуговуванні та підвищенню рівня безпеки, CDN став популярним інструментом для вирішення проблем, що виникають із традиційним веб-хостингом.

Переваги використання CDN різняться залежно від розміру та потреб веб-сайту, основні переваги для більшості користувачів можуть бути розбиті на наступні:

- прискорення завантаження веб-сайтів. За допомогою розташування кешованих даних ближче до користувачів веб-сайту шляхом розташування поблизу сервера CDN (а також впровадження інших оптимізацій), час завантаження сторінки значно зменшується. Оскільки користувачі веб-сайтів надають перевагу сайтам, що завантажуються швидко, CDN може знизити кількість незадоволених користувачів і збільшити кількість часу, який люди проводять на сайті;
- зменшення навантаження на веб-сервер. Витрати на пропускну здатність і потужність серверів це основна витрата веб-сайтів. Завдяки кешуванню та іншим оптимізаціям, CDN можуть зменшити кількість запитів, що повинен

обробляти веб-сервер, зменшуючи таким чином витрати на хостинг для розробника веб-сайту;

- зменшення кількості збоїв у роботі серверу. Велика кількість трафіку може порушити нормальну роботу веб-сайту. Завдяки своєму розповсюдженому характеру, CDN може обробляти більше трафіку та витримувати технічні несправності краще, ніж багато веб-серверів, що не використовують CDN;
- підвищення рівня безпеки. CDN може покращити безпеку, взяти на себе удар під час DDoS атак, а також покращити сертифікати безпеки у порівнянні з сертифікатами користувача.

За своєю суттю, CDN це мережа серверів, пов'язаних разом із метою постачання даних з більшою швидкістю, з меншими витратами, з більшою надійністю та безпекою. Для підвищення швидкості сервіси CDN розміщують сервери в місцях, де вони будуть доступні до певної категорії користувачів з мінімальними витратами часу.

Користувач CDN не взаємодіє напряму з веб-сервером, у більшості випадків користувач взаємодіє лише з проміжними CDN-серверами. CDN-сервери, у свою чергу, звертаються з запитами до основного веб-серверу у разі потреби.

Поломки обладнання та стрибки трафіку внаслідок зловмисних атак або просто збільшення популярності можуть потенційно перевантажити веб-сервер і перешкодити доступу користувачів до сайту чи послуги. Добре налаштований CDN має кілька функцій, які мінімізують час простою веб-сайту:

- балансування навантаження розподіляє мережевий трафік рівномірно на декількох серверах, що полегшує адаптацію до швидкого збільшення трафіку;
- інтелектуальна відказостійка система забезпечує безперебійне обслуговування навіть у тому випадку, якщо один або кілька серверів CDN переходять у режим офлайн через апаратну несправність; відмова може перерозподілити трафік на інші операційні сервери;

- у випадку, якщо весь центр обробки даних має технічні проблеми, маршрутизація передає трафік в інший доступний центр обробки даних, гарантуючи, що жоден користувач не втратить доступ до веб-сайту.

У таблиці 4.6 порівняно можливості, що надають користувачу деякі з CDN-сервісів, що здатні забезпечити найбільш малий час відповіді на запит клієнта.

Таблиця 4.6 – Порівняння характеристик CDN сервісів [31]

Властивість	Akamai	Verizon	Azure	Fastly	AWS CloudFront
Швидке налаштування	ні	ні	так	так	так
Відктиті ціни	ні	ні	так	так	так
Кількість точок присутності	більше 100	32	53	57	більше 100
Статистика у реальному часі	так	ні	ні	так	ні
Ціна за гігабайт трафіку	невідома	невідома	$0.087	$0.12	$0.085
Логування	так	так	так	так	так
GZIP-стиснення	так	так	так	так	так
Захист від DDoS-атак	так	так	так	так	так
Email-підтримка	так	так	ні	так	ні
HTTP/2	так	так	так	так	так
Запити у байтовому вигляді	так	так	так	так	так

Як видно з таблиці 4.16, у більшості характеристик наведені CDN інструменти

не відрізняються один від одного, оскільки усі вони здатні надати весь перелік необхідних послуг для зменшення навантаження на веб-сервер, захисту від зловмисних атак та зменшення часу очікування користувачем відповіді на запит. Отже, найбільш важливим параметром є відкритість вартості використання CDN сервісу та вартість його використання. Найбільш дешевим з наведених CDN інструментів є AWS Cloudfront, отже прийняте рішення використовувати його як CDN інструмент для системи ідентифікації елементів на веб-сторінках [32].

4.2 Розробка структурної схеми

Структурну схему системи наведено в додатку []. Розроблена система складається з наступних модулів:

- модуль скрипту кінцевого користувача;
- модуль інтерфейсу користувача та адміністратора;
- модуль CDN;
- модуль веб-сервера;
- модуль кешування даних на рівні сервера;
- база даних;
- база даних кешу.

Модуль скрипту кінцевого користувача вбудовується користувачем на сторінку власного сайту й слугує для взаємодії кінцевих користувачів (користувачів сайту користувача) з елементами, створеними користувачем.

Модуль інтерфейсу користувача та адміністратора слугує для маніпуляцій над елементами зі сторони користувача (наявні операції додавання, перевибору, видалення елементів), перегляду статистики, а також для взаємодії адміністраторів з елементами користувачів.

Модуль CDN слугує для кешування даних на рівні CDN, для зменшення навантаження на сервер, для прискорення відповіді на запити клієнта зі сторони сервера у різних країнах, а також для захисту сервера від DDoS атак.

Модуль веб-сервера реалізує логіку вдаємодії між системою, користувачем, кінцевим користувачем, адміністратором, кешем та базою даних (авторизація користувачів, збереження даних про елементи та активність кінцевих користувачів, надання даних про елементи). Цей модуль взаємодіє з базою даних та з базою даних кешу через модуль кешування даних.

4.3 Розробка функціональної схеми

Розроблена функціональна схема знаходиться у додатку []. Розроблена система складається з наступних модулів:

- модуль скрипту кінцевого користувача;
- модуль інтерфейсу користувача та адміністратора;
- модуль CDN;
- модуль веб-сервера;
- модуль кешування даних на рівні сервера;
- база даних;
- база даних кешу.

Модуль скрипту кінцевого користувача слугує для взаємодії кінцевих користувачів з елементами, створеними користувачем. Даний модуль складається з наступних підсистем:

- пошук елементів;
- відображення знайдених елементів;
- створення статистики.

Модуль інтерфейсу користувача та адміністратора слугує для маніпуляцій над елементами зі сторони користувача. Даний модуль складається з таких підсистем:

- додавання елементів;
- редагування та видалення елементів;
- перегляд статистики.

Модуль CDN слугує у якості проміжного шару між клієнтськими модулями та

веб-сервером. Цей модуль складається з таких підсистем:

- кешування даних на рівні CDN;
- прискорення відповіді серверу на запити в різних країнах.

Модуль веб-сервера реалізує логіку вдаємодії між системою, користувачем, кінцевим користувачем, адміністратором, кешем та базою даних. Підсистеми даного модулю:

- авторизація користувачів;
- збереження даних про веб-сторінки та елементи;
- надання даних про елементи;
- збереження та надання статистики.

4.4 Сценарії використання системи

У даному підрозділі наведено сценарії використання системи та розробка діаграми сценаріїв системи. Діаграму сценаріїв із функціональними вимогами до системи наведено у додатку []. На діаграмі сценаріїв присутні такі актори:

- користувач;
- кінцевий користувач;
- адміністратор.

Користувач це власник сайту, з яким взаємодіє система. Він вбудовує на сайт скрипт для кінцевого користувача й обирає елементи для подальшої ідентифікації, а також може здійснювати інші маніпуляції над елементами та переглядати статистику про елементи та активність кінцевих користувачів власного сайту.

Адміністратор має доступ до елементів користувачів й має змогу допомагати користувачам у роботі з системою. Також адміністратор має змогу робити усі дії, що може робити користувач, а отже роль адміністратора є розширенням ролі користувача.

Кінцевий користувач це людина, що заходить на сайт користувача й взаємодіє з елементами, що були знайдені системою на веб-сайті користувача. Він має змогу

переглядати елементи, що були обрані користувачем для ідентифікації, а також управляти збором статистики про свої дії.

У таблицях нижче наведено опис прецедентів (таблиці 4.7 – 4.16).

Таблиця 4.7 – Сценарій використання

Назва	Сплата підписки
ID	1
Опис	Після реєстрації або логіну користувач потрапляє на сторінку сплати підписки у разі, якщо підписка ще не була сплачена
Актори	користувач
Вигоди компанії	Можливість для користувача оплатити підписку у зручний спосіб
Частота користування	Після кожної реєстрації
Тригери	Користувач авторизується на не має сплаченої підписки
Передумови	Користувач проходить крок логіну або реєстрації
Постумови	-
Основний розвиток	Користувач сплачує підписку
Альтернативні розвитку	-
Винятки	-

Таблиця 4.8 – Сценарій використання

Назва	Створення елементу для ідентифікації
ID	2
Опис	Користувач має можливість обрати елемент, для якого будуть згенеровані шляхи від кореневого елементу, за допомогою яких у подальшому буде здійснюватися пошук даного елементу
Актори	користувач, адміністратор
Вигоди компанії	Сервіс неможливий без зручного вибору елементів для ідентифікації
Частота користування	Постійно
Тригери	Користувач натискає кнопку вибору елементу
Передумови	Користувач проходить крок логіну
Постумови	-
Основний розвиток	Користувач обирає елемент
Альтернативні розвитку	-
Винятки	-

Таблиця 4.9 – Сценарій використання

Назва	Перегляд створених елементів
ID	3
Опис	Користувач переглядає елементи, збережені для подальшої ідентифікації
Актори	користувач, адміністратор
Вигоди компанії	Важлива функція перегляду елементів, що потрібна для будь-яких користувачів
Частота користування	Періодично
Тригери	Користувач вибирає опцію “Перегляд збережених елементів”
Передумови	Користувач авторизувався
Постумови	Відправляється запит на наявні елементи
Основний розвиток	Користувач переходить на сторінку зі списком створених елементів
Альтернативні розвитку	-
Винятки	-

Таблиця 4.10 – Сценарій використання

Назва	Перевибір створеного елементу
ID	4
Опис	Перевибір елементу, що був створений раніше, зі збереженням існуючих даних
Актори	користувач, адміністратор
Вигоди компанії	Частина основного мінімальному функціоналу
Частота користування	Постійно
Тригери	Користувач натискає кнопку його перевибору елементу на сторінці елементу
Передумови	Користувач авторизувався й зайшов на сторінку списку елементів
Постумови	Користувач обирає новий елемент для ідентифікації
Основний розвиток	Дані про елемент в базі даних оновлюються
Альтернативні розвитку	Користувач здійснює перехід за прямим посиланням на сторінку елементу й натискає кнопку перевибору елементу
Винятки	-

Таблиця 4.11 – Сценарій використання

Назва	Видалення елементу
ID	5
Опис	Користувач може видалити раніше створений елемент
Актори	користувач, адміністратор
Вигоди компанії	Частина основного мінімального функціоналу
Частота користування	Періодично
Тригери	Користувач натискає кнопку видалення елементу
Передумови	Користувач сплачує підписку й переходить на сторінку списку елементів
Постумови	Видалення елементу
Основний розвиток	Елемент видаляється
Альтернативні розвитку	-
Винятки	-

Таблиця 4.12 – Сценарій використання

Назва	Перегляд статистики
ID	6
Опис	Користувач може переглядати статистику про кінцевих користувачів, що бачать створені ним елементи
Актори	користувач, адміністратор
Вигоди компанії	Задоволеність користувачів наявністю інформації про створені ними елементи, можливість переглядати статистику адміністраторами системи й відслідковувати тенденції іхньої активності.
Частота користування	Періодично
Тригери	Користувач заходить на головну сторінку
Передумови	Користувач сплачує підписку й авторизується
Постумови	-
Основний розвиток	Користувач бачить статистику
Альтернативні розвитку	Користувач переходить на головну сторінку з іншої сторінки сайту
Винятки	-

Таблиця 4.13 – Сценарій використання

Назва	Видалення елементів інших користувачів
ID	7
Опис	Адміністратор може видаляти елементи інших користувачів, що не є адміністраторами
Актори	адміністратор
Вигоди компанії	Наявність можливості зручно зменшити кількість інформації в базі даних, шляхом видалення непотрібних елементів.
Частота користування	Періодично
Тригери	Адміністратор натискає кнопку видалення елементу
Передумови	Адміністратор переходить на сторінку елементів певного користувача
Постумови	Адміністратор підтверджує видалення
Основний розвиток	Елемент видаляється
Альтернативні розвитку	-
Винятки	-

Таблиця 4.14 – Сценарій використання

Назва	Перевибір створеного елементу іншого користувача
ID	8
Опис	Перевибір елементу, що був створений іншим користувачем раніше, зі збереженням існуючих даних
Актори	адміністратор
Вигоди компанії	Наявність можливості допомогти користувачеві зі сторони адміністратора
Частота користування	Періодично
Тригери	Адміністратор натискає кнопку його перевибору елементу на сторінці елементу
Передумови	Адміністратор авторизувався й зайшов на сторінку списку елементів іншого користувача
Постумови	Адміністратор обирає новий елемент для ідентифікації
Основний розвиток	Дані про елемент в базі даних оновлюються
Альтернативні розвитку	Адміністратор здійснює перехід за прямим посиланням на сторінку елементу й натискає кнопку перевибору елементу
Винятки	-

Таблиця 4.15 – Сценарій використання

Назва	Перегляд створених елементів іншого користувача
ID	9
Опис	Адміністратор переглядає елементи, створені іншим користувачем
Актори	адміністратор
Вигоди компанії	Наявність можливості допомогти користувачеві зі сторони адміністратора
Частота користування	Періодично
Тригери	Адміністратор натискає кнопку переходу на сторінку елементів користувача
Передумови	Адміністратор авторизується
Постумови	-
Основний розвиток	Адміністратор бачить список елементів, створених користувачем
Альтернативні розвитку	-
Винятки	-

Таблиця 4.16 – Сценарій використання

Назва	Керування збором статистики
ID	10
Опис	Кінцевий користувач може відключити збор статистики про свої дії на сторінці
Актори	кінцевий користувач
Вигоди компанії	Задоволеність користувачів
Частота користування	Періодично
Тригери	Кінцевий користувач вводить в консолі браузера команду для відключення збору статистики
Передумови	Кінцевий користувач відкриває сторінку, на якій є скрипт клієнтської частини системи
Постумови	-
Основний розвиток	Відключення збору статистики для дій кінцевого користувача
Альтернативні розвитку	-
Винятки	-

4.5 Результати розробки

Користувач (власник сайту), як і адміністратор, взаємодіє з системою через веб-інтерфейс користувача. Кінцевий користувач (відвідувач сайту користувача) взаємодіє з системою через скрипт кінцевого користувача, що вбудовується на сайт користувача.

На рисунку 4.7 зображена головна сторінка інтерфейсу користувача. На даній сторінці користувач може переглядати статистику, а також переходити з неї на інші сторінки.

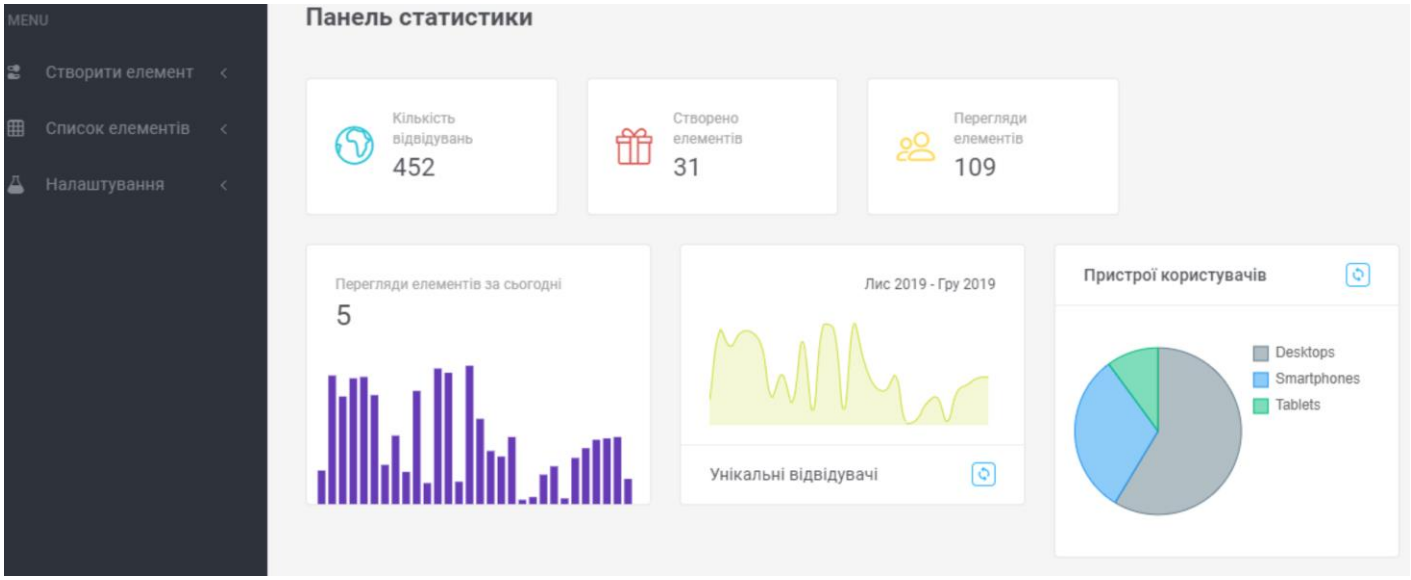

Рисунок 4.7 – Панель статистики інтерфейсу користувача

Також інтерфейс користувача дозволяє користувачу створити елемент, переглянути список елементів та задати налаштування.

На рисунку 4.8 зображено сторінку списку елементів, створених користувачем. На даній сторінці користувач може переглядати, перестворювати та видаляти елементи, що були створені раніше.

Для демонстрації роботи системи вбудуємо скрипт кінцевого користувача на сайт google.com, використовуючи командну строку. Це альтернативний варіант додавання скрипта на сторінку, що використовується у разі відсутності доступа

розробника до вихідного коду сайту.

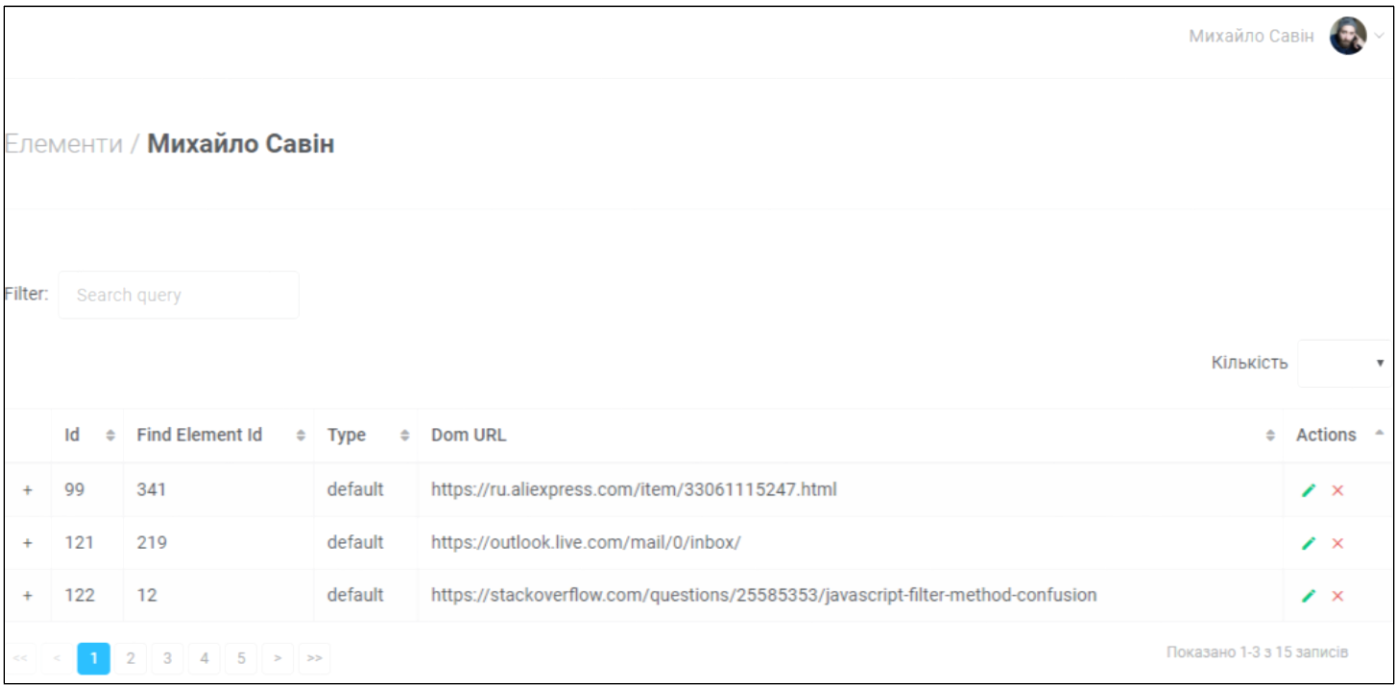

Рисунок 4.8 – Панель статистики інтерфейсу користувача

Головна сторінка пошукокого сайту Google без змін наведена на рисунку 4.9. Якщо обрати, наприклад, елемент, що є посиланням на пошту користувача, після чого змінити структуру сторінки, то, при заході кінцевого користувача на сторінку, елемент коректно знаходиться алгоритмом й виділяється як знайдений (рисунок 4.10).

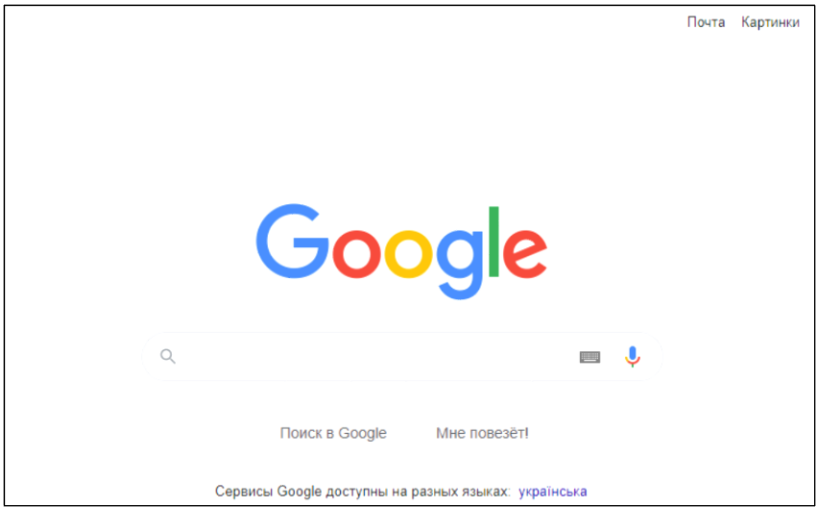

Рисунок 4.9 – Головна сторінка пошукового сайту Google без внесених змін [33]

На основі даних результатів роботи, можна зробити висновок, що алгоритм коректно шукає елемент на сторінці після значної зміни положення елементу на сторінці як відносно своїх попередніх координат, так і відносно попереднього положення у структурі сторінки. Вимоги до інтерфейсу користувача, що включають у себе наявність можливості керуванням станом елементів, а також перегляду статистики, також виконані.

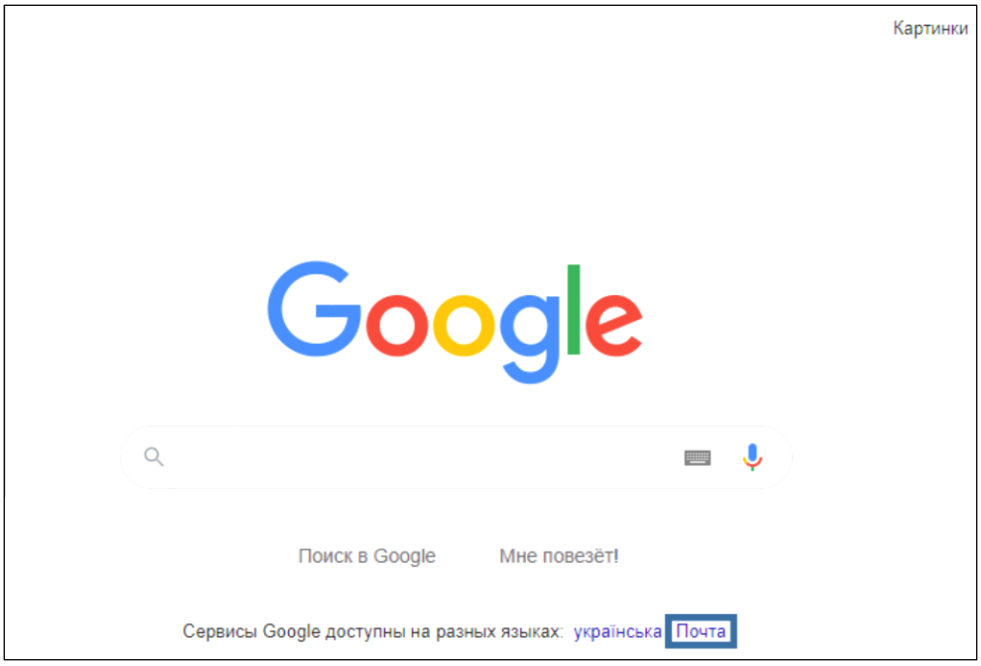

Рисунок 4.10 – Головна сторінка сайту Google після внесених змін та запуску алгоритму пошуку [33]

У даному розділі було проаналізовано існуючі елементи та технології, що можуть бути використані при розробці системи пошуку елементів на веб-сторінці. В результаті проведеного аналізу сформульовано наступний список технологій, які максимально відповідають вимогам до них:

- PostgreSQL у якості бази даних;
- Node.js у якості серверної мови програмування;
- Redis у якості бази даних кешу;
- AWS Cloudfront у якості CDN-сервісу;
- Webpack у якості інструмента збірки клієнтської частини системи.

Також у даному розділі була обгрунтована розробка структурної та функціональної схеми системи, а також діаграми сценаріїв системи. Розроблена система складається з наступних модулів:

- модуль скрипту кінцевого користувача;
- модуль інтерфейсу користувача та адміністратора;
- модуль CDN;
- модуль веб-сервера;
- модуль кешування даних на рівні сервера;
- база даних;
- база даних кешу.

Також була розроблена система пошуку елементів на сторінках, структура яких зазнає змін та продемонстровані результати роботи даної системи. За результатами роботи системи можна зробити висновок, що система працює згідно вимог до неї й коректно знаходить елементи на змінених сторінках в автоматичному режимі.

5 РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

5.1 Опис ідеї проекту

Ідея проекту даної роботи – розробити систему, що дозволить ідентифікувати елемент на сторінці навіть тоді, коли її структура зазнала значних змін. Це є надзвичайно важливою складовою для автоматичної роботи системи прив’язання підказок або інших додатків, що спрощують роботу користувача на сайті, до елементів на веб-сторінках. Це є дуже позитивним фактором зменшення вартості підтримання системи прив’язання підказок до елементів на веб-сторінках, адже у разі автоматизації системи відпадає необхідність витрачати бюджет на робітників, що виконують ті задачі, з якими не здатен впоратися недосконалий алгоритм. Користувачі системи мають змогу обирати елемент, який має бути ідентифікований на сайті у майбутньому, навіть у разі зміни структури веб-сторінки.

Узагальнення цих ідей можна побачити в таблиці 5.1.

Основні вимоги до стартап-проектів, як правило, полягають у наступному:

- аналіз ринку на наявність аналогічних рішень;
- ознайомлення можливих споживачів та партнерів з стартап-проектом;
- вибір найкращого часу для подання інформації про стартап-проект;
- забезпечення процесів обробки інформації при реалізації проекту;
- збереження результатів обробки інформації в необхідному вигляді.

Обмеження на роботу з інформацією в загальному вигляді можуть бути представлені як група характеристик, які дійсно належать сторонам стартап-проекту, а саме:

- об'єкту (джерелу інформації);
- медіатору (засобу отримання, обробки, виробництва і передачі інформації, посереднику між суб'єктом і об'єктом);
- суб'єкту (активній стороні діяльності, одержувачу і перетворювачу
- інформації).

Таблиця 5.1 Опис ідеї стартап проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
		Скорочення часу, що витрачається на ідентифікацію елементу на веб-сторінці
		Скорочення кількості помилок та обмежень при ідентифікації елементів на веб-сторінках
		Відсутність необхідності переприв’язання елементів до підказок після кожної зміни структури веб-сторінки

Складність отримання інформації на сьогоднішній день обумовлена її розподіленням та фрагментарністю, а також різнопрофільним наповненням.

На сьогоднішній день на ринку існує декілька основних сервісів для прив’язання підказок до елементів, що використовують системи ідентифікації елементів на веб-сторінках зі змінною структурою, доведеться конкурувати і з ними, а не тільки з тими, що пропонують рекомендації. Тому із найкрупніших конкурентів можна виокремити WalkMe, Appcues та WhatFix. Порівняння даного проекту із ними можна побачити в таблиці 5.2.

5.2 Аналіз потенційних техніко-економічних переваг

Аналіз сильних, слабких та нейтральних сторін стартап-проекту визначено в таблиці 5.2. Перевагами системи є наявне рішення проблеми роботи з великою

кількістю даних.

Таблиця 5.2 Визначення сильних, слабких та нейтральних характеристик ідеї проекту

Техніко-економічні характеристики ідеї	Товари/концепції конкурентів				Слабка сторона	Нейтральна сторона	Сильна сторона
	Мій проект	WalkMe	Appcues	Whatfix			
Можливість пошуку елементу на сторінці у разі значної зміни структури	Так	часткова	ні	ні			+
Вартість експлуатації							+
Кількість доступних елементів для пошуку (на одній сторінці)	Велика	Велика	Середня	Середня			+

Продовження таблиці 5.2

Техніко-економічні характеристики ідеї	Товари/концепції конкурентів				Слабка сторона	Нейтральна сторона	Сильна сторона
Робота в автоматичному режимі	так	ні	так	так		+	
Вбудований аудіо та відео зв'язок	ні	так	так	ні	+		
Час відповіді на запити при малому навантаженні	0,4	0,35	0,4	0,5		+	
Час відповіді на запити при великому навантаженні	0,7	0,5	0,6	0,5	+		

Продовження таблиці 5.2

Техніко-економічні характеристики ідеї	Товари/концепції конкурентів				Слабка сторона	Нейтральна сторона	Сильна сторона
Можливість масштабування	+	+	+	-		+	

5.3 Технологічний аудит ідеї проекту

Список технологій реалізації програмного комплексу наведено в таблиці 5.3. Також у таблиці 5.3 вказана наявність та доступність наведених технологій. Основними технологіями, що були використані при написанні системи пошуку елементів на веб-сторінці із динамічною структурою, є Javascript, Node.js, PostgreSQL, Webpack та Babel.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Реалізація програмного забезпечення з використанням мови програмування Node.js.	Мова програмування Javascript, віртуальна машина V8, мова програмування Node.js	Наявні	Доступні

Продовження таблиці 5.3

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
2	Створення веб-інтерфейсу з використанням мови програмування Javascript	Мова програмування Javascript, інструмент Webpack, бібліотека Babel	Наявні	Доступні
3	Створення кросбраузерного скрипта для вбудови на сторінку веб-сайту клієнта	Мова програмування Javascript, DOM API, бібліотека js-combinatorics	Наявні	Доступні

5.4 Аналіз ринкових можливостей запуску стартап-проекту

Попередню характеристику потенційного ринку проекту наведено в таблиці 5.4.

Таблиця 5.4 – Попередня характеристика потенційного ринку проекту

№	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	500 грн / ум.од

Продовження таблиці 5.4

№	Показники стану ринку (найменування)	Характеристика
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Недискримінаційні якісні
5	Специфічні вимоги до стандартизації та Сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	70,1%

5.5 Аналіз ринкового середовища

У таблиці 5.5 наведені основні фактори загроз та можливі реакції, що можуть нейтралізувати дані загрози. У таблиці 5.6 наведені основні фактори можливостей та реакція компанії, що дозволить реалізувати дані можливості.

Таблиця 5.5 – Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Крадіжка інтелектуальної власності	Крадіжка ідеї або ключової інтелектуальної інновації	Відсудження прав інтелектуальної власності, забезпечення якісного захисту інформації, зміна методики групування даних

Продовження таблиці 5.5

№	Фактор	Зміст загрози	Можлива реакція компанії
2	Відмова компаній у співпраці	Керівництво компанії не погоджується на співпрацю	Пропозиція більш вигідних умов співпраці
3	Недостача стартових капіталовкладень	Недостача початкових інвестицій для реалізації мінімально життєздатного продукту	Пошук нових джерел інвестицій

Поточний розвиток галузі систем прив’язання підказок до елементів показує позитивні тенденції, тож потреба в системах, що можуть в автоматичному режимі ідентифікувати елементи на сторінці у разі значної зміни структури сторінки, є достатньо високою.

Таблиця 5.6 – Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Отримання необхідних інвестицій	Сформований початковий капітал, необхідний для реалізації мінімально життєздатного продукту	Розробка мінімально життєздатного продукту

Продовження таблиці 5.6

№	Фактор	Зміст можливості	Можлива реакція компанії
2	Висока Зацікавле-ність користува-чів	Обіг використання веб-служби становить більше 1000 запитів в день	Підтримка стабільної роботи системи та проведення масштабування системи Збільшення цін на використання сервісу
3	Успішна маркетин-гова політика	В результаті проведеної маркетингової політики отримана висока зацікавленість користувачів	Підтримка стабільної роботи системи та проведення масштабування системи Збільшення цін на використання сервісу Використання подібної маркетингової стратегії надалі для залучення нових користувачів
4	Ліквідація конкурента	Конкурент ліквідував свою компанію у результаті власного бажання або зовнішніх чинників	Проведення маркетингової кампанії для монополізації ринку

5.6 Визначення потенційних груп клієнтів

Характеристику потенційних клієнтів стартап-проекту наведено у таблиці 5.5. Також у таблиці наведена цільова аудиторія проекту, відмінності у поведінці різних потенційних груп й вимоги до споживачів товару. Основна потреба, що формує ринок – бажання зменшення часу роботи співробітника або клієнта при впровадженні нових веб-сервісів.

Таблиця 5.7 – Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Бажання зменшення часу роботи співробітника або клієнта при впровадженні нових веб-сервісів	1. Малий середній та великий бізнес галузі ІТ	1. Ведення бізнесу на платформах інтернет-ресурсів	Можливість роботи з великою кількістю елементів, можливість бачити підказки, прив’язані до коректних елементів

5.7 Аналіз пропозиції

У таблиці 5.8 наведено ступеневий аналіз конкуренції на ринку (особливості конкурентного середовища та його характеристики).

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип конкуренції – чиста	Немає чітко відкремленого лідера серед конкурентів,та відповідно мала їх кількість	Сприятливо впливає на розвиток проекту
Рівень конкурентної боротьби – міжнародний	Сфера систем прив’язання підказок до елементів зачіпає практично всі країни світу	Можливий швидкий вихід на світові ринки
За галузевою ознакою – внутрішньогалузева	Загроза появи нових конкурентів, ринкова влада споживачів, висока потреба у товарі	Інформування ринку щодо якості використовуваної новаторської технології, пропозиція гнучких цін
За видами товарів – товарно-родові	Надання різних сервісів одного виду	Маркетингова політика

Продовження таблиці 5.8

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Цінова політика	Використання цін для покращення економічних умов збуту	Зменшення вартості сервісу Використання нових каналів розподілу

Аналіз конкуренції за М. Портером надає більш детальну інформацію про стан ринку і можливості конкурентів (таблиця 5.9). Привабливість галузі, в даному контексті, має відношення до достатньої рентабельності галузі. Непривабливою галуззю є така, в якій поєднання сил знижує рентабельність. Найнепривабливішою є галузь, що наближається до досконалої конкуренції. Макросередовище складається з тих сил, які впливають на здатність компанії до обслуговування власних клієнтів і отримання прибутку. Достатня привабливість галузі не означає, що будь-яка компанія в ній буде отримувати однаковий прибуток.

Згідно з Портером, модель п'яти сил потрібно використовувати тільки для галузі в цілому. Модель не призначена для використання для групи галузей або якоїсь частини однієї галузі.

П'ять сил Портера включають в себе:

- аналіз загрози появи продуктів-замінників;
- аналіз загрози появи нових гравців;
- аналіз ринкової влади постачальників;
- аналіз ринкової влади споживачів;
- аналіз рівня конкурентної боротьби.

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером.

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	WalkMe, Appcues, Whatfix	Цінність ідеї та її якісна реалізація	Немає	Споживачі прямо впливають на успішність продукту, оскільки сплачуються кошти за ліцензію користування	Конкуренти можуть стати дешевшими або стануть надавати якісніші послуги
Висновки	Конкурентів небагато, відсутність монополії, проте конкурентна боротьба значна за рахунок хорошої якості проектів конкурентів	Є можливість виходу на ринок	Немає постачальників	Так, реалізація потреб клієнтів вирішує успішність проекту	Достатньо важко предентувати на клієнтів, які вже використовують інший готовий продукт

5.8 Перелік факторів конкурентоспроможності

Навіть при врахуванні конкурентів розроблена система має низку переваг, які наведені в таблиці 5.10.

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	У конкурентів наявні функціональні проблеми реалізації	Алгоритму пошуку елементів на веб-сторінці, що справно працює для будь-яких змін структури веб-сторінки в автоматичному режимі
2	Ідея	Система пошуку елементів на веб-сторінці, структура якої зазнала значних змін з часу вибору елементу
3	Швидкий вихід на ринок	Достатньо випустити мінімальну базову працездатну версію продукту на ринок для його використання
4	Цінова політика	Отримання прибутку здійснюється за рахунок гнучкої моделі оплати

5.9 Аналіз сильних та слабких сторін стартап-проекту

За наведеними факторами конкурентоспроможності в таблиці 5.10 було виконано порівняльний аналіз сильних та слабких сторін програмного рішення (стартап-проекту), який відображено в таблиці 5.11.

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін системи ідентифікації елементів на сторінці з динамічною структурою

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні із системою ідентифікації елементів на сторінці						
			-3	-2	-1	0	+1	+2	+3
1	У конкурентів наявні функціональні проблеми реалізації	17							+
2	Ідея	12					+		
3	Швидкий вихід на ринок	14					+		
4	Цінова політика	13					+		

5.10 Формування базових стратегій розвитку

Базовою стратегією для стартап-проекту є стратегія диференціації, що означає надання программному комплексу властивостей, які будуть відрізняти його від конкурентів. В таблиці 5.12 наведено визначення базової стратегії розвитку.

Таблиця 5.12 – Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції	Базова стратегія розвитку
Пропонування програмного комплексу для ІТ команд	Диференційо-ваний маркетинг	Здатність пропонувати унікальний функціонал	Стратегія диференціації

5.11 SWOT-аналіз

Складений SWOT-аналіз на основі отриманих даних відображено в таблиці 5.13.

Таблиця 5.13 – SWOT-аналіз стартап-проекту

Сильні сторони: – автоматизованість системи; – мала залежність від інвестицій; – можливість роботи з великою кількістю елементів; – підтримка значних змін структури сторінки; – невелика кількість сильних – конкурентів.	Слабкі сторони: – вузьке коло клієнтів; – потреба в рекламі.
Можливості: – інвестиції; – невелика кількість конкурентів.	Загрози: – зміна тенденцій розвитку ринку; – вихід на ринок нових конкурентів.

5.12 Опис цільових груп потенційних клієнтів

В таблиці 5.14 наведено характеристики потенційних клієнтів.

Таблиця 5.14 – Характеристики потенційних клієнтів стартап проекту.

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Компанії, що розробляють системи прив’язання підказок до елементів на веб-сторінках	Висока	80%	Середня	Складно
2	Компанії, що використовують системи прив’язання підказок до елементів на веб-сторінці	Висока	70%	Середня	Середньо
Які цільові групи обрано: компанії, що розробляють системи прив’язання підказок до елементів на веб-сторінках					

5.13 Вибір стратегії конкурентної поведінки

В таблиці 5.15 наведено базові стратегії конкурентної поведінки.

Таблиця 5.15 – Визначення базової стратегії конкурентної поведінки

Чи є проект першопрохідцем на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Ні	Залучати нових та забирати існуючих	Можливість використання системи у системі прив’язання підказок до елементу на веб-сторінці	Стратегія вилику лідера

5.14 Розробка стратегії позиціонування

На основі потреб споживачів з обраних сегментів, а також в залежності від обраної базової стратегії розвитку в таблиці 6.16 наведено визначені стратегії позиціонування.

Таблиця 5.16 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Коректний пошук елементіва на сторінці при значній зміні її структури Підтримка одночасного пошуку великої кількості елементів Збір статистики про знайдені елементи	Стратегія диференціації	Розширення базового функціоналу	Система ідентифікації елементів Обрати елемент для ідентифікації в один клік Збір статистики про ідентифіковані елементи

5.15 Розробка маркетингової програми стартап-проекту

Першим кроком до формування маркетингової концепції товару, який отримує споживач. Для цього у таблиці 5.17 підсумовано результати попереднього аналізу конкурентоспроможності товару.

В таблиці 5.17 визначено ключові переваги концепції потенційного товару.

Таблиця 5.17 – Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Коректний пошук елементіва на сторінці при значній зміні її структури	Алгоритм пошуку елементів на веб-сторінці за будь-яких змін структури веб-сторінки	Робота алгоритму в автоматичному режимі
2	Підтримка одночасного пошуку великої кількості елементів	Швидкий алгоритм та оптимізований веб-сервер, що може обробити велику кількість запитів одночасно	Швидкість алгоритму, відсутність потреби чекати на прийняття рішення людиною
3	Збір статистики про знайдені елементи	Збір статистики без необхідності додаткових налаштувань	Відсутність необхідності додаткових налаштувань

Наступним кроком є визначення цінових меж представлених в таблиці 5.18, якими необхідно керуватись при встановленні ціни на товар.

Таблиця 5.18 – Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
Товарів-замінників немає	Товари замінники мають високий рівень цін (від $5000 на рік)	Цільова група споживачів має високий рівень доходів	Нижня межа – це найменша ціна рішення найближчого найдешевшого конкурента, верхня межа, це на 1% більша ціна на найближчого найдорожчого конкурента

5.16 Висновки

Під час проведення дослідження магістерської дисертації у якості стартап-проекту було створено відповідну аргументовану документацію. Протягом усього процесу дослідження було досягнуто повне розуміння цілей проекту, його переваг та недоліків, які в свою чергу дали можливість точніше визначити мету проекту та користувацьку аудиторію програмного застосунку, визначити можливі методи монетизації, маркетингову стратегію для ІТ команд та власного використання.

На даний час попит на такий програмний комплекс може бути високим через активний розвиток інтернет-технологій та потребу великих компаній у збільшенні швидкості освоєння персоналом нових веб-сайтів.

Перевагами проекту є:

- робота алгоритму в автоматичному режимі;
- збір статистики без додаткових налаштувань;
- коректна робота алгоритму за значних змін структури веб-сторінки.

Унікальність проекту полягає в наявності алгоритму, що здатен в автоматичному режимі знаходити елементи на веб-сторінках, що зазнали значних змін з часу вибору елементу. Дана властивість системи реализована за допомогою використання DOM API та комбінаторики.

ВИСНОВКИ

У даній магістерській дисертації здійснено аналіз існуючих систем та алгоритмів пошуку елементів на веб-сторінках із динамічною структурою. Було проведене порівняння таких існуючих алгоритмів, як:

- алгоритм пошуку елементу на сторінці за його атрибутами;
- алгоритм пошуку елементу на сторінці за його положенням відносно лівого верхнього кута сторінки;
- алгоритм пошуку елементу на сторінці за його CSS-стилями.

На основі проведеного аналізу було зроблено висновок про нездатність існуючих алгоритмів задовільняти існуючі потреби у пошуку елементів в автоматичному режимі.

Також був проведений аналіз існуючих методів об’єктної моделі документа, що можуть бути використані для написання алгоритму пошуку елементу на сторінці. Було встановлено, що:

- метод document.querySelector() найкраще підходить для пошуку елементу за його положенням відносно інших елементів на сторінці;
- властивості елемента element.childNodes, element.firstChild, element.nextSibling, element.previousSibling дозволяють отримати доступ до елементів, що розташовані поруч з поточним елементом.

З використанням наведених методів об’єктної моделі документа було розроблено алгоритм пошуку елементу на сторінці після зміни структури сторінки за допомогою прокладання різних шляхів до елементу від кореневого елементу. При написанні цього алгоритму були використані такі елементи комбінаторики, як булеан та декартовий добуток множин.

Також були сформульовані функціональні та нефункціональні вимоги до системи, на основі яких був проведений аналіз існуючих елементів та технологій розробки веб-сайтів.

В результаті проведеного аналізу для розроблюваної системи пошуку елементів на сторінці було обрано такі технології:

- PostgreSQL у якості бази даних;
- Node.js у якості серверної мови програмування;
- Redis у якості бази даних кешу;
- AWS Cloudfront у якості CDN-сервісу;
- Webpack у якості інструмента збірки клієнтської частини системи.

Також в магістерській дисертації була обгрунтована розробка структурної та функціональної схеми системи, а також діаграми сценаріїв системи. Розроблена система складається з наступних модулів:

- модуль скрипту кінцевого користувача;
- модуль інтерфейсу користувача та адміністратора;
- модуль CDN;
- модуль веб-сервера;
- модуль кешування даних на рівні сервера;
- база даних;
- база даних кешу.

Також було досліджено магістерску дисертацію у якості стартап-проекту, в результаті чого було створено відповідну аргументовану документацію. Протягом усього процесу дослідження було досягнуто повне розуміння цілей проекту, його переваг та недоліків, які в свою чергу дали можливість точніше визначити мету проекту та користувацьку аудиторію програмного застосунку, визначити можливі методи монетизації, маркетингову стратегію для IT команд та власного використання.

Також була розроблена система пошуку елементів на сторінках, структура яких зазнає змін та продемонстровані результати роботи даної системи. За результатами роботи системи можна зробити висновок, що система працює згідно вимог до неї й коректно знаходить елементи на змінених сторінках в автоматичному режимі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is WalkMe? [Електронний ресурс] – Режим доступу до ресурсу: https://www.walkme.com/technology

2. WalkMe solution architectural white paper [Електронний ресурс] – Режим доступу до ресурсу: https://docplayer.net/1734901-Walkme-solution-architectural-white-paper.html

3. How to Use Walk Me Through in simPRO [Електронний ресурс] – Режим доступу до ресурсу: https://helpguide.simprogroup.com/Content/WalkMe-in-simPRO.htm

4. Appcues Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.appcues.com/article/46-welcome-to-appcues

5. Appcues Integration [Електронний ресурс] – Режим доступу до ресурсу: https://segment.com/integrations/appcues/

6. Whatfix [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Whatfix

7. Accelerate User Adoption Using Whatfix [Електронний ресурс] – Режим доступу до ресурсу: https://whatfix.com/accelerate-user-adoption

8. Configuring the embedded self-help | Whatfix [Електронний ресурс] – Режим доступу до ресурсу: https://docs.bmc.com/docs/smartit1908/configuring-the-embedded-self-help-886823590.html

9. CSS Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/css/css_intro.asp

10. What is the DOM? | CSS-TricksFlywheel [Електронний ресурс] – Режим доступу до ресурсу: https://css-tricks.com/dom/

11. DOM tree [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.info/dom-nodes

12. Dynamic HTML: The Definitive Reference, 3rd Edition / Danny Goodman // O'Reilly Media, – 2009. – С. 134–382

13. The HTML DOM Element Object [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/jsref/dom_obj_all.asp

14. Document.querySelector() [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/API/Document/querySelector

15. Basics of Combinatorics Tutorials & Notes [Електронний ресурс] – Режим доступу до ресурсу: https://www.hackerearth.com/practice/math/combinatorics/basics-of-combinatorics/tutorial

16. C++ Programming Examples on Combinatorial Problems & Algorithms – Sanfoundry [Електронний ресурс] – Режим доступу до ресурсу: https://www.sanfoundry.com/cpp-programming-examples-combinatorial-problems-algorithms

17. Mastering Node.js / Sandro Pasquali // Packt Publishing, – November 2013. – С. 250–300

18. Professional Node.js / Pedro Teixeira // Wrox, – 2012. – С. 286–347

19. PostgreSQL 9 Administration Cookbook LITE: Configuration, Monitoring and Maintenance / Simon Riggs, Hannu Krosing // Packt Publishing, – 2011. – С. 49–61

20. 2019 Database Trends – SQL vs. NoSQL, Top Databases, Single vs. Multiple Database Use [Електронний ресурс] – Режим доступу до ресурсу: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use

21. Why PostgreSQL is better than MySQL – 2ndQuadrant | PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.2ndquadrant.com/en/blog/postgresql-better-mysql-1

22. Mastering PostgreSQL 9.6 / Hans-Jurgen Schonig // Packt Publishing, – 2017. – С. 49–98

23. What are the differences between NPM, Bower, Grunt, Gulp, Webpack, Browserify, Slush, Yeoman, and Express? – Quora [Електронний ресурс] – Режим доступу до ресурсу: https://www.quora.com/What-are-the-differences-between-NPM-Bower-Grunt-Gulp-Webpack-Browserify-Slush-Yeoman-and-Express?share=1

24. Webpack vs Gulp vs Grunt vs Browserify: Choosing the Best Tool to Boost Developer Productivity [Електронний ресурс] – Режим доступу до ресурсу: https://www.cleveroad.com/blog/gulp-browserify-webpack-grunt

25. Speedtesting Grunt and Gulp – Режим доступу до ресурсу: http://tech.tmw.co.uk/2014/01/speedtesting-gulp-and-grunt

26. NPMCompare.com – Comparing bower vs. browserify vs. grunt vs. gulp vs. webpack [Електронний ресурс] – Режим доступу до ресурсу: https://npmcompare.com/compare/bower,browserify,grunt,gulp,webpack

27. Memory Systems / Spencer Ng, David Wang, Bruce Jacob // Elsevier, – 2010. – С. 442–508

28. Redis Cookbook / Tiago Macedo, Fred Oliveira // O'Reilly Media, – 2011. – С. 38–45

29. Instant Effective Caching with Ehcache / Daniel Wind // Packt Publishing, – 2013. – С. 74–76

30. CDN Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.cdnoverview.com

31. CDN comparison | The best CDN review [Електронний ресурс] – Режим доступу до ресурсу: http://cdncomparison.com

32. CDN Pricing Comparison – CDN Reviews [Електронний ресурс] – Режим доступу до ресурсу: https://cdn.reviews/cdn-pricing-comparison

33. Google [Електронний ресурс] – Режим доступу до ресурсу: https://www.google.com

ДОДАТКИ

Додаток А – Публікація

Міжнародна наукова інтернет-конференція "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 43)" / Збірник тез доповідей: випуск 43 (м. Тернопіль, 14 листопада 2019 р.). – Частина 1. – Тернопіль. – 2019. – 141 с.

УДК 001 (063)
ББК 72я431

ISSN 2522-932X

Збірник тез доповідей підготовлено за матеріалами Міжнародної наукової інтернет-конференції (випуск 43) від 14 листопада 2019 р.

Збірник матеріалів науково-практичної інтернет-конференції включаються до наукометричної бази даних "РІНЦ/RSCI".

Тексти матеріалів конференції подаються в авторській редакції. Відповідальність за точність, достовірність і зміст поданих матеріалів несуть автори.

Наша адреса: Оргкомітет МНІК "Конференція онлайн"
а/с 797, м. Тернопіль 46005
тел. моб. 068 366 0 525
e-mail: inetkonf@ukr.net

URL Інтернет-конференції: http://www.konferenciaonline.org.ua/

Савін М.С.
Національний Технічний Університет України "Київський політехнічний інститут ім. Ігоря Сікорського", м. Київ
Кафедра автоматики та управління в технічних системах, студент

АЛГОРИТМ ПОШУКУ ЕЛЕМЕНТІВ НА СТОРІНЦІ ЗІ ЗМІНЮВАНОЮ СТРУКТУРОЮ

Задача пошуку елементів на сторінках із динамічною структурою виникає з потреби прив'язання певних сторонніх елементів (підказок, тощо) до елементів веб-сторінки й збереження цього прив'язання зі зміною структури сторінки. Для цього необхідно мати змогу ідентифікувати той елемент на зміненій сторінці, що якого було раніше прив'язано підказку.

Для вирішення поставленої задачі необхідно розробити алгоритм, що враховує всі можливі шляхи знайдення певного елементу на сторінці від кореневого елементу, оскільки положення кореневого елементу на сторінці завжди відомо, а врахування всіх можливих шляхів від кореневого елементу до обраного елементу дозволить врахувати будь-які переміщення обраного елементу на сторінці відносно кореневого, адже його все ще можна буде знайти на сторінці за деякими шляхами, за якими його можна було знайти раніше.

Наприклад, якщо на сторінці знаходиться кореневий елемент <body>, всередині якого знаходиться елемент , що має атрибут class="myDivClass", всередині якого знаходиться інший елемент з класом "myDivClass2", всередині якого знаходиться шуканий елемент <a>, то дійти до елементу <a> від кореневого елементу, використовуючи такий метод, як document.querySelector, можна наступними шляхами:

- body > div.myDivClass > div.myDivClass2 > a;
- body > div.myDivClass > a;
- body div.myDivClass > a;
- body div.myDivClass a;
- body div.myDivClass2 > a;
- body > div > div > a;
- body > div.myDivClass > div.myDivClass2 > *;
- body > div.myDivClass > div> *;
- body div.myDivClass2 > *;
- body > a;
- body > div.myDivClass > div.myDivClass2 > a;
- body > div.myDivClass > div.myDivClass2 > a.

У деяких з цих можливих шляхів від кореневого елементу до шуканого відсутній перший елемент , у деяких з них відсутній другий елемент , у деяких з них відсутні обидва елементи. Це означає, що частина з

можливих шляхів дозволить знайти шуканий елемент за будь-яких змін структури сторінки.

Також, частина шляхів до елементу може відкидати певні атрибути чи тег обраного елемента або його батьків. Наприклад, розглянемо наступні два шляхи від кореневого елементу body до обраного елементу a:

– body > div > div > a;
– body > div > div > *.

У першому випадку елемент буде знайдено на сторінці лише за умови якщо він не змінить свій тег з <a> на будь-який інший. Другий шлях знайде елемент навіть якщо тег елементу буде змінено на, наприклад, .

Використовуючи ідею про прокладення усіх можливих шляхів від кореневого елементу до обраного елементу, розглянемо етапи роботи алгоритму, що прокладає дані шляхи, перевіряє, чи усі вони ведуть до шуканого елементу, й після зміни сторінки використовує усі ці шляхи для знайдення елементу, обираючи той елемент, на який вказує більшість з згенерованих шляхів.

Нижче наведені етапи роботи алгоритму побудови усіх можливих шляхів до елементу:

– знайти усі елементи між кореневим елементом та шуканим елементом;
– використовуючи усі можливі комбінації множини проміжних елементів, скласти множину можливих шляхів, частина з яких включає лише частину проміжних елементів;
– знайти усі атрибути для шуканого елементу та для кожного з проміжних елементів;
– використовуючи комбінації множини для атрибутів можного з проміжних елементів та для атрибутів шуканого елементу, а також декартовий добуток множин, розширити множину можливих шляхів до шуканого елементу таким чином, що частина зі шляхів буде використовувати лише частину з атрибутів елементу;
– продублювати шляхи для кожного з проміжних елементів та для шуканого елементу, відкидаючи вімоги відносно тегу елементу.

Список використаних джерел:

1. Document.querySelector [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/ru/docs/Web/API/Document/querySelector
2. Document Object Model [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Document_Object_Model
3. What is the Document Object Model? [Електронний ресурс] – Режим доступу: https://www.w3.org/TR/DOM-Level-2-Core/introduction.html