МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КОРНІЛЕНКО ОЛЕКСАНДР СЕРГІЙОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
д. т. н., доцент
_____ Т. В. Нескородєва
«_____» _____ 2022 р.

ІНФОРМАЦІЙНА СИСТЕМА ПОШУКУ АВТОМОБІЛЬНИХ ЗАПЧАСТИН ЗА НЕПОВНИМ ОПИСОМ

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:
Штовба С.Д., професор кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____
(підпис)

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2022

АНОТАЦІЯ

Корніленко О.С. Інформаційна система пошуку автомобільних запчастин за неповним описом. Спеціальність 122 «Комп’ютерні науки», Освітня програма “Комп`ютерні технології обробки даних. (Data Science)” Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У магістерській роботі розроблено автоматизовану інформаційну систему з веб-інтерфейсом для пошуку автомобільних запчастин. Система забезпечує пошук автозапчастин за неповним описом. Відмінністю є алгоритм, що дозволяє виконати пошук автозапчастин за неповним описом швидко та не потребує великих ресурсів.

Ключові слова: інформаційна система, Python, Bootstrap, Flake8, Docker, Django, пошук за неповним описом, автозапчастини.

83 с.,12 табл.,31 рис., 47 джерел.

ABSTRACT

Kornilenko O. Information system for car repair partsearching under incompletedescription. Specialty 122 "Computer Science", Educational program "Computer Technologies of Data Processing (Data Science)" Vasyl' Stus Donetsk National University, Vinnytsia, 2022.

In the master's work, an automated information system with a web interface for searching for automotive spare parts was developed. The system provides search of auto parts by incomplete description. The difference is an algorithm that allows you to search for auto parts by incomplete description quickly and does not require large resources.

Keywords: information system, Python, Bootstrap, Flake8, Docker, Django, search for incomplete description, auto parts.

83 Pages, 12 Tables, 31 Fig.,47 Ref.

ЗМІСТ

ВСТУП

Сьогодні для багатьох компаній і підприємств, що працюють в Інтернеті, потреба визначити всю можливу інформацію про користувачів є надзвичайною. Одним із найефективніших способів отримання такого роду інформації є пошук за неповним описом. Використовуючи отриману інформацію, підприємства можуть регулювати обробку запитів клієнтів залежно від типу трафіку, мобільного чи комп'ютерного, можуть вирішувати, який вміст надсилати на пристрій, який запитує, і навіть налаштовувати вміст на ходу. На сьогоднішній день, коли використовується широкий спектр пристроїв – це особливо корисно, оскільки це дозволяє отримати максимальну віддачу від своєї стратегії націлювання. Окрім веб-оптимізації, це також має очевидне застосування у сфері реклами, тобто де пристрій користувача може використовуватися як критерій для визначення належності до цільового ринку. Аналітика є не менш важливим варіантом використання, і оскільки доступна найповніша інформація про клієнтів, подальший аналіз може значно покращити рішення щодо публікації вмісту, стратегії націлювання або оптимізації процесу перетворення користувачів веб-сайту на клієнтів.

Актуальність.

Дане дослідження і розробка алгоритмічного забезпечення - дуже актуальний для підприємств які працюють в мережі інтернет. Кількість таких підприємств з кожним роком тільки збільшується, і все більше уваги приділяється потребам індивідуальних користувачів, тому необхідно першочергово з`ясувати ці потреби.

Для цього був розроблений алгоритм, який буде частиною програмного додатку, тобто сайту каталогу автозапчастин.

Мета дослідження:

Розробити алгоритм та програмний додаток (сайт) для пошуку автозапчастини на основі збігів шаблонів пошуку з неповними описами.

Завдання дослідження:

– огляд досліджень і результатів, отриманих для вирішення проблеми зіставлення шаблонів за неповним описом;

– вибрати кілька алгоритмів і методів для вирішення проблеми зіставлення шаблонів пошуку на основі неповного опису та провести широке тестування;

– прискорення існуючих алгоритмів або розробка нових, на основі виконаного аналізу;

– розробити програмну реалізацію оптимального алгоритму;

– провести аналіз отриманих результатів.

Об’єкт дослідження.

Об’єктом дослідження є система обробки тексту.

Наукова новизна та методи розв’язання задачі.

Існує багато методів та алгоритмів для зіставлення шаблонів пошуку за неповним описом . Деякі з них спеціалізуються на роботі з парами шаблон-рядок, деякі мають справу з кількома шаблонами або кількома рядками. Дані алгоритми накладають певні обмеження на рядок або шаблон, такі як довжина, спеціальні символи, які можна використовувати. Крім того, ці алгоритми можна оптимізувати для найгіршого випадку тривалості виконання, середнього часу виконання, споживання пам’яті або інших показників. Але поки що не запропоновано алгоритм, який може вирішити поставлене завдання за задовільний робочий час.

Розроблений алгоритм вирішує проблему зіставлення довільної кількості шаблонів пошуку з текстовими екземплярами та забезпечує найкращу часову оцінку складності знизу серед існуючих шаблонів, що також підтверджується отриманими результатами.

Такий результат досягається шляхом поділу алгоритму на дві фази: попередня обробка пошукових шаблонів і фаза обробки вхідних запитів. На першому етапі використовуються додаткові модифіковані структури даних, такі як префіксні дерева та алгоритм Левенштейна. На другому етапі виконується

швидкий аналіз вхідних рядків з використанням структур даних, отриманих на першому етапі.

1. ОГЛЯД СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАНЬ РОЗРОБКИ

Спочатку оглянемо існуючі алгоритми в галузі порівняння шаблонів пошуку за неповним описом на предмет збігу.

Проведемо огляд літератури щодо методів порівняння шаблонів пошуку з рядками, тобто розбору регулярного виразу. Більшість методів зосереджено на обробці пар регулярних виразів і вхідних рядків. Цей підхід добре працює з парами, проте зі збільшенням кількості регулярних виразів їх продуктивність зростає лінійно з кількістю виразів. Найпопулярніші алгоритми, що використовують регулярні вирази, але, як виявилося, вони використовують багато пам'яті та часу для попередньої обробки регулярних виразів.

Регулярні вирази дорогі в обчисленнях. Тому використовувати їх рекомендується тільки в разі дійсної необхідності. Однак шаблони пошуку, тобто рядки символів підстановки, використовуються майже так ж часто, як регулярні вирази, проте їх легше використовувати та вони мають швидші алгоритми обчислень.

Переглядаючи дослідження, можна виявити, що досить багато роботи включає вивчення відображення великої кількості шаблонів пошуку в тексті.

Проблему легко вирішити, якщо застосувати аби-який алгоритм для зіставлення одного шаблону пошуку з представником тексту та всіх доступних шаблонів пошуку з даним текстом відповідно. Але зрозуміло, що це неефективний алгоритм для вирішення цієї проблеми, тому що інформація, яка отримується на кожному індивідуальному етапі порівняння шаблону пошуку та тесту – не використовується, а також не використовується інформація про “схожі” шаблони пошуку.

1.1 Інформаційні системи підбору запчастин як об`єкт автоматизації

Інформаційна система — це формальна соціотехнічна організаційна система, призначена для збору, обробки, зберігання та розповсюдження інформації [1]. З соціотехнічної точки зору інформаційні системи складаються з чотирьох компонентів: завдання, люди, структура (або ролі) і технологія. Інформаційні системи можна визначити як інтеграцію компонентів для збору, зберігання та обробки даних , дані яких використовуються для надання інформації, внеску в знання, а також цифрових продуктів , які полегшують прийняття рішень.

Автоматизована інформаційна система — це набір взаємопов’язаних даних, обладнання, програмного забезпечення, людей і стандартних процедур, призначених для збору, обробки, розповсюдження, зберігання та представлення інформації відповідно до цілей організації. В інформаційну епоху сьогодні майже всі інформаційні системи використовують комп’ютерні технології, тому в подальшому під інформаційними системами розумітимуть автоматизовані системи [2].

Основною функцією інформаційної системи є збір, зберігання, обробка та представлення необхідної інформації, необхідної для її подальшого застосування. Основним продуктом операції є необроблена інформація, тому процес функціонування інформаційної системи має забезпечувати якість вихідної інформації [3].

Загалом, основною метою функціонування інформаційної системи є задоволення потреб користувачів системи та забезпечення своєчасної та надійної доставки повної, правдивої та конфіденційної інформації, яка безпосередньо залежить від вимог користувачів. Досить часто запити є неповними, з чим більшість інформаційних систем не справляються, що дає в результаті негативну оцінку з боку користувачів.

1.2 Аналіз алгоритмів опрацювання неповних запитів

Алгоритми нечіткого пошуку (теж відомий як пошук подібності чи fuzzy string search) є базою систем перевірки орфографії та популярних пошукових систем, таких як Yahoo чи Google [4].

Методи пошуку нечіткої інформації дозволяють розширювати запити за допомогою введення тексту, що містяться в архівах документів, наприклад, в електронних бібліотеках [5].

Оригінальний алгоритм здатний знаходити всі лексикографічно близькі слова, які відрізняються заміною, пропуском, вставкою символу тощо.

Нечіткий пошук рекомендується використовувати при пошуку слів з друкарськими помилками, а також у разі сумнівів щодо правильності написання прізвищ, назв організацій тощо. Наприклад, запит «мультимедіа» можна розширити термінами: «сультимедіа», «мультиседіа».

Унікальний алгоритм, що реалізує нечіткий пошук, заснований на принципі асоціативного доступу до слів, що містяться в текстовому індексі повнотекстового архіву документа.

Алгоритм нечіткого пошуку характеризується метрикою - функцією відстані між двома словами, яка дозволяє нам оцінити, наскільки вони схожі в даному контексті. Математичне визначення метрики включає необхідність задовольнити умову нерівності трикутника: (X — множина, а p — метрика):

$$p(x, y) \leq p(x, z) + p(z, y), x, y, z \epsilon X, \quad (1.1)$$

де X – множина;

p – метрика.

Проте важливо зазначити, що міра відноситься до загального поняття відстані. Найчастіше використовуються метрики Левенштейна, Хеммінга та відстань Дамерау-Левенштейна. У той же час відстань Хеммінга є лише мірою багатьох слів однакової довжини, і це значно зменшує область її застосування.

Розглянемо існуючі алгоритми нечіткого пошуку.

1.2.1 Відстань Левенштейна

Відстань Левенштейна — це рядкова метрика для вимірювання різниці між двома послідовностями. Відстань Левенштейна між двома словами - це мінімальна кількість редагувань одного символу (вставок, видалень або замін), необхідних для зміни одного слова на інше [6].

Наприклад, відстань Левенштейна між «кошеня (kitten)» і «сидить (sitting)» дорівнює 3, оскільки наступні 3 редагування змінюють одне в інше, і немає способу зробити це менш ніж 3 редагуваннями:

1. k itten → s itten (заміна "s" на "k"),
2. sitt e n → sitt i n (заміна «і» на «е»),
3. sittin → sittin g (вставка «g» у кінці).

Відстань Левенштейна має кілька простих верхніх і нижніх меж. До них належать:

- це принаймні різниця розмірів двох струн;
- це щонайбільше довжина довшого рядка;
- він дорівнює нулю тоді і тільки тоді, коли рядки рівні;
- якщо струни мають однаковий розмір, відстань Хеммінга є верхньою межею відстані Левенштейна;
- відстань Хеммінга — це кількість позицій, у яких відповідні символи у двох рядках відрізняються. відстань Левенштейна між двома струнами не перевищує суму їхніх відстаней Левенштейна до третьої струни (нерівність трикутника) [7].

1.2.2 Відстань Дамерау-Левенштейн

Даний варіант додає ще одне правило до визначення відстані Левенштейна - транспонування двох сусідніх літер також вважається операцією, так само як і вставки, видалення та заміни.

Більшість друкарських помилок були транспозиціями. Тому саме цей показник на практиці дає найкращі результати.

Щоб обчислити таку відстань, необхідно лише трохи змінити алгоритм, який використовується для знаходження стандартної відстані Левенштейна, таким чином: зберегти три останні рядки матриці замість двох і зазначити відповідну додаткову умову - у випадку виявлення транспозиції, її значення також слід враховувати при розрахунку відстані.

Відстань Дамерау–Левенштейна відрізняється від класичної відстані Левенштейна тим, що, окрім трьох класичних операцій редагування одного символу (вставлення, видалення та заміни), вона включає дозвільне переміщення між операціями [8].

1.2.3 Метод N-грам

Цей підхід був винайдений давно і є найбільш широко використовуваним, оскільки він надзвичайно простий у реалізації та забезпечує хорошу продуктивність. Алгоритм відповідає такому принципу.

Алгоритм відповідає такому принципу : «Якщо слово А відповідає слову В з деякими помилками, то дуже ймовірно, що вони мають принаймні один спільний підряд довжини N».

Підрядки довжини N і називають N-грамами. Під час індексування слова розбиваються на N-грами, а потім слова вносяться до списку для кожної N-грами. Під час пошуку запит також розбивається на N-грами, і кожна N-грама шукається послідовно в списку слів, що містять такий підрядок [9].

1.3 Огляд популярних інтернет-магазинів автозапчастин

Проаналізуємо три популярні інтернет-магазини запчасти EXIST.UA, DOU.UA та AVTO.PRO:

1. EXIST.UA є одним із наймасштабніших автомобільних сайтів в Україні з великим асортиментом товарів, пов’язаних з автомобілем. Існуючі юзери є авторизованими представниками автомобільного бізнесу і можуть особисто додавати продукти на платформу[10].

На головній сторінці Exist користувачі можуть побачити насичену вмістом

механізм пошуку автозапчастин, яка включає категорії автозапчастин або марки та моделі автомобілів.

Автотовари: Exist.ua пропонує найбільший асортимент автотоварів в Україні. Exist.ua співпрацює з близько 400 постачальниками, які мають власні склади в Україні, а також відправляють замовлення клієнтів із закордонних складів.

Купівля запчастин - це тільки частина того, що ви можете зробити на Exist.ua, звичайно, після покупки деталі вам треба буде її встановити. Exist рекомендує партнерські СТО з найрізноманітнішими ціновими діапазонами, щоб гарантувати високу якість обслуговування.

Щодо дизайну веб-сайту, можна сказати, що він не дуже простий, навігація важлива для недосвідчених користувачів, з точки зору розробника, веб-сайт здається занадто логічним, але для такого типу веб-сайту це очікуваний результат. Панель навігації розташована у верхній частині екрана і містить посилання на основні розділи сайту. Відразу під навігацією праворуч є карусельний блок із новинами та списком поточних акцій (рис. 1.1).

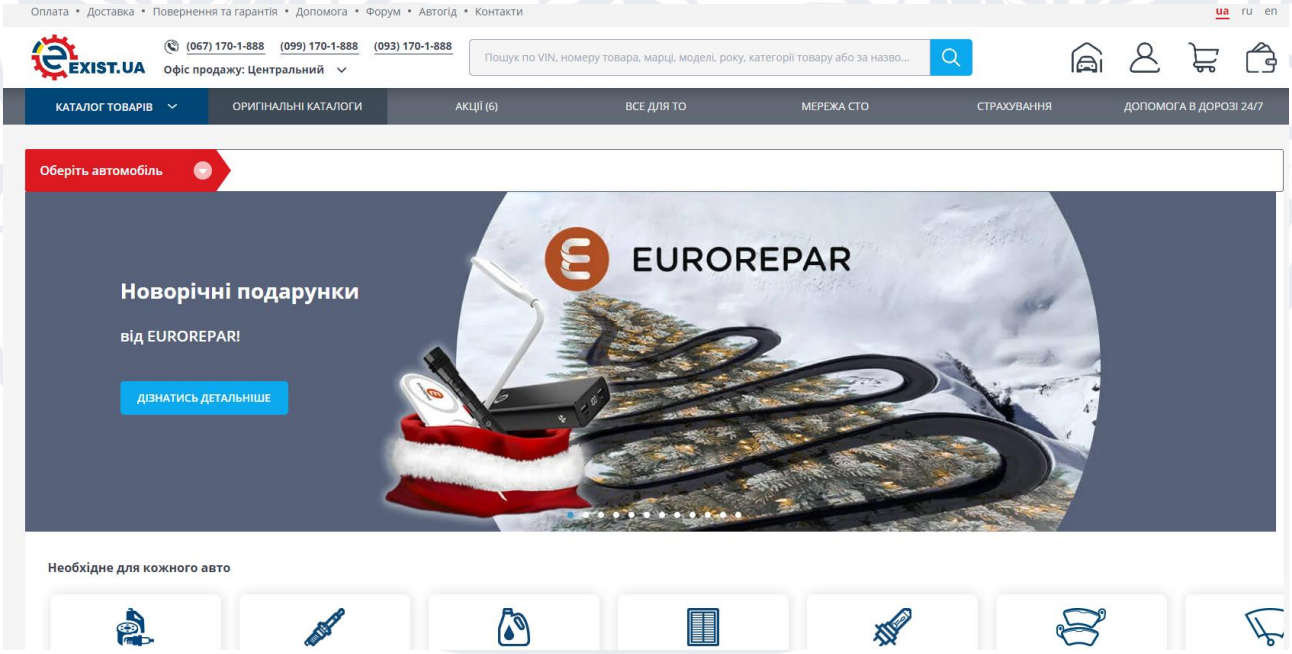


Рисунок 1.1 – Головна сторінка магазину «EXIST.UA»

Переваги даного магазину:

- Зручний поділ на тематичні підрозділи за категоріями;
- можливість швидкого відстеження будь-яких новин;
- зручний пошук — за допомогою добре розробленої системи пошуку необхідно задати параметри пошуку, як показано на рис 1.2;

Рисунок 1.2 – Пошукова система «EXIST.UA»

- на головній сторінці веб-сайту є можливість входу в особистий кабінет, що відкриває додаткові можливості, такі як збереження певної інформації, сповіщення електронною поштою тощо;
- зручний поділ автозапчастин (рис. 1.3);
- можливість додати свій конкретний автомобіль (марка, модель, державний номер.

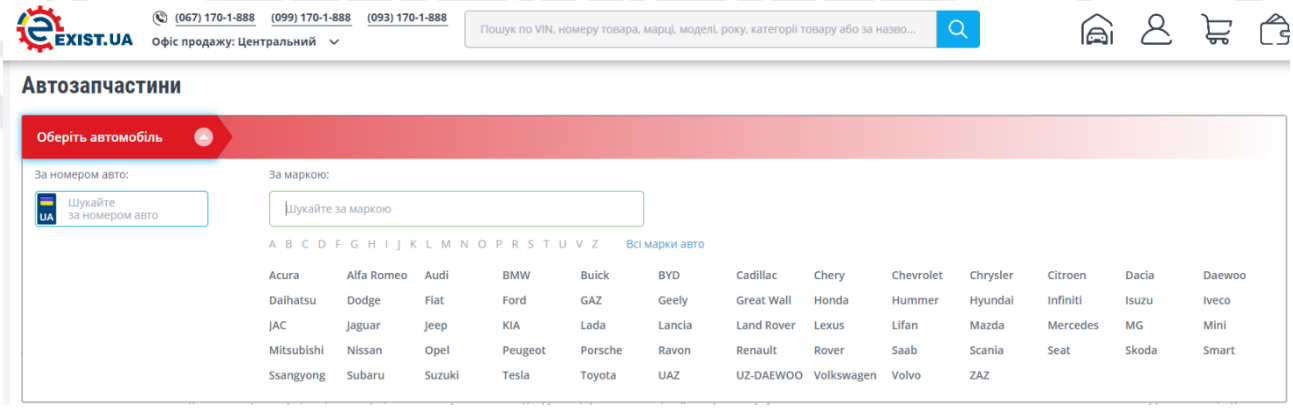


Рисунок 1.3 – Пошукова система «EXIST.UA»

Мінуси:

- сайт перенасичений контентом, різними інтуїтивно незрозумілими віджетами;
- відчутна швидкість оброблення запитів та завантаження сторінок.

2. DOK.UA – інтернет-магазин автозапчастин, що займається на продажом автозапчастин, моторних мастил та акумуляторів в сегменті електронної комерції в Україні. Завдяки кваліфікованим працівникам, реалізована досить зручна система підбору запчастин та добре налагоджена система доставки товарів по усій Україні, вони мають велику кількість клієнтів, більшість з яких звертаюсь до сервісу на постійній основі [11].

З першого дня роботи компанія DOK орієнтована на безперервний ріст і розвиток, у зв'язку з чим корпоративна етика заснована на пильності і далекоглядності по відношенню до усіх клієнтів. Враховуючи високий рівень обслуговування, який вони обрали, DOK надає клієнтам найвищу якість і зручність, включаючи підбір консультацій кваліфікованих спеціалістів чи необхідних товарів.

Стосовно дизайну веб-сайту можна сказати, що він набагато зручніший за Exist.ua, немає перевантажень, користувачеві легко переглядати, а макет сторінки скомпонований з більшим простором між вмістом, тому стає легше сприймати інформацію. Панель навігації розташована у верхній частині екрана та містить посилання на основні розділи сайту, які є категоріями автозапчастин та обладнання. Відразу після навігації нижче користувач може вибрати потрібний автомобіль, вказавши марку, модель і рік випуску (рис. 1.4).

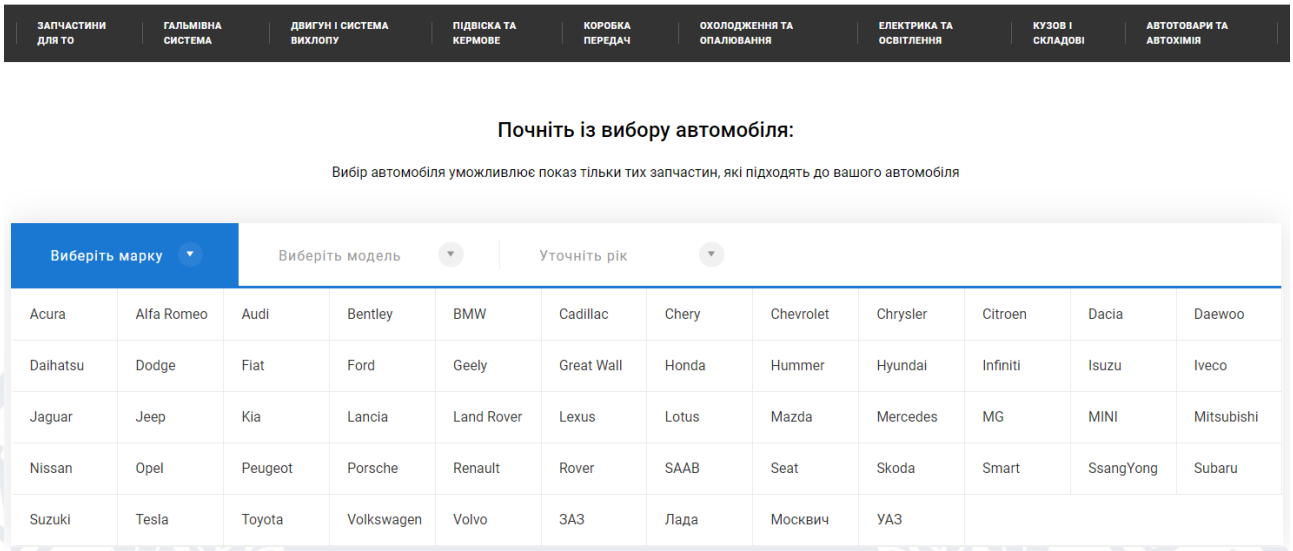


Рисунок 1.4 – Автомагазин запчастин «DOK.UA»

Плюси цього ресурсу:

- Зручний поділ контенту на тематичні розділи – розділи за категоріями;
- є змога швидкого переміщення по сайту завдяки зручній навігації;
- зручна система рекомендацій – за допомогою алгоритму рекомендації товарів користувачі можуть швидко знайти потрібне навіть без налаштування пошуку (рис. 1.5);
- вхід в особистий кабінет на сайті;
- адаптивний дизайн (рис. 1.6);
- FAQ, форум, розділ технічної підтримки;
- завдяки використанню новітніх технологій у розробці та кеш-менеджері веб-сайт дуже чуйно реагує на дії користувачів.

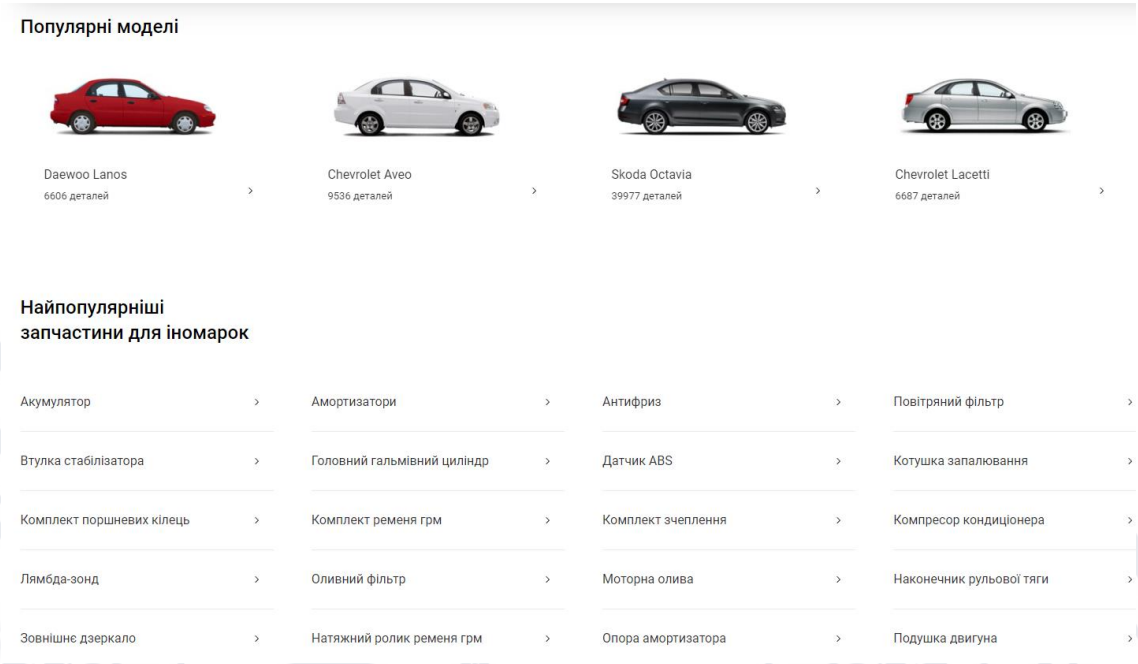


Рисунок 1.5 – Рекомендації магазину «DOK.UA»

Рисунок 1.6 – Головна сторінка «DOK.UA»

Мінуси:

- Розмір шрифту деяких елементів дизайну занадто малий, що ускладнює читання вмісту користувачам із вадами зору.

3. AVTO.PRO – головне завдання побудова прозорого цифрового ринку автомобільних товарів. Avto.pro зібрав велику кількість магазинів і продавців в одному місці. Тому все більше водіїв користуються цією онлайн-платформою, коли їм потрібні запчастини для ремонту. Головною перевагою є цифровий каталог з тисячами позицій, в якому доступні не тільки оригінальні запчастини, а й аналоги [12].

Потрібні запчастини легко знайти, широкий асортимент і широкий діапазон цін, тому ви завжди можете заощадити на покупці, взаємозамінність запчастин перевіряється фахівцями і відображається в результатах пошуку, покупці можуть знайти комплектні інформація про продавців, виробників запчастин і навіть автозвалище.

Що стосується маркування, то відразу можна сказати, що конструкція не дуже зручна. Перші 5 хвилин на сайті його дизайн здається абсолютно невідповідним будь-якому інтернет-магазину, але потім користувачі до нього звикають. На сайті також немає явної навігації, і користувачі повинні переміщатися по сайту, доки не знайдуть окреме посилання (рис 1.7).

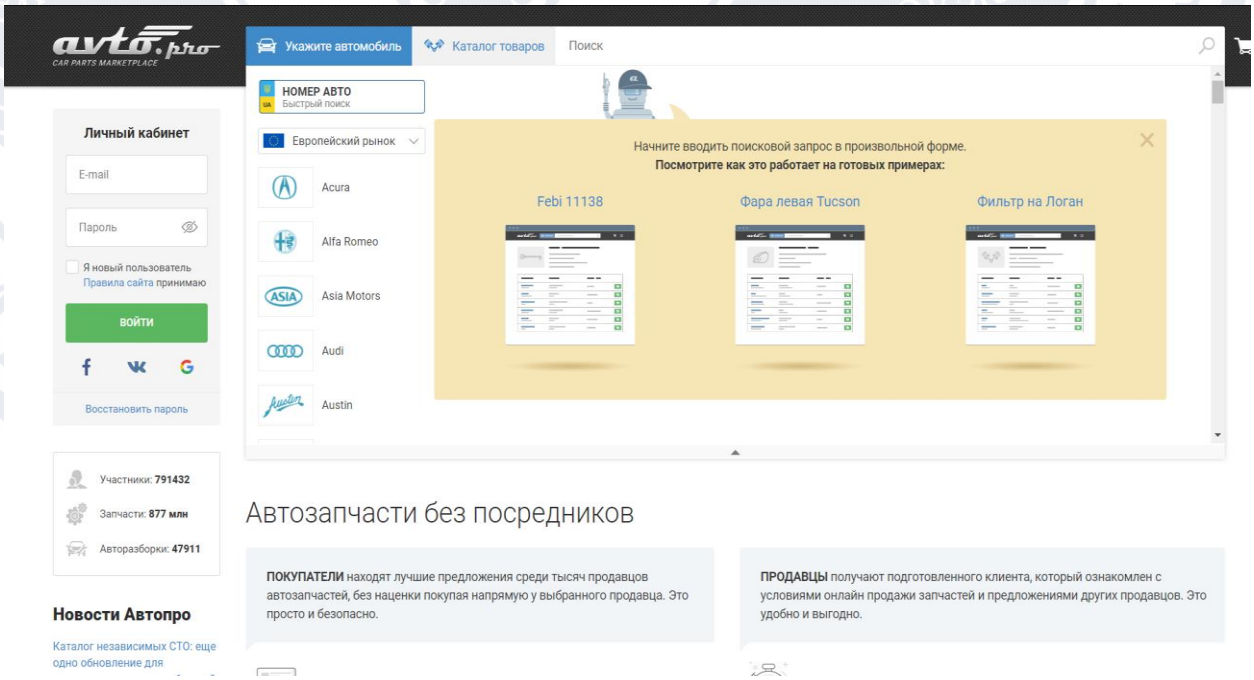


Рисунок 1.7 – Автомагазин запчастин «AVTO.PRO»

Переваги:

- Проста авторизація особистих рахунків, розташована помітно у верхньому лівому куті;
- можливість підбору автомобіля за номерним знаком;
- можливість купівлі та продажу автозапчастин;
- інформаційна таблиця товарів, включаючи спосіб доставки товару, місто, код, виробника та короткий опис;
- можливість швидкого пошуку контенту за тегами.

Мінуси:

- Незручний дизайн;
- жахливий каталог автозапчастин на головній сторінці (рис. 1.8).

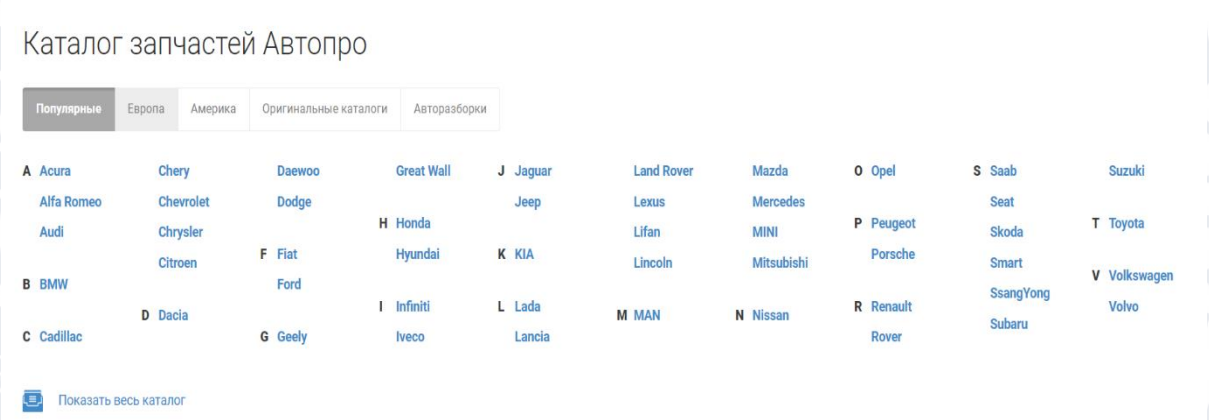


Рисунок 1.8 – Автомагазин запчастин «AVTO.PRO»

Існує також і багато інших аналогів, просто розглянувши вище перелічені системи пошуку автозапчастин можна зробити наступні висновки: існує досить велика кількість аналогів розробляємої системи зі своїми перевагами та недоліками, проте в жодному з них не реалізовано пошук за неповним описом. Також більшість інтернет-магазинів має застарілий та незручний дизайн, що доводить доцільність розробки інформаційної системи, яка зможе підбирати запчастини за неповним описом.

1.3 Постановка задачі

Шляхом порівняння та аналізу наявних інформаційних систем інтернет-магазинів автозапчастин, таких як EXIST.UA, DOC.UA, AVTO.PRO, отримано їх характеристики.

За їх характеристиками можна визначити критерії розвитку інтернет-магазину автозапчастин, який успадкує сучасні програмні рішення та передовий досвід, унікальний для цього ресурсу інтернет-магазину. Інтернет-портал повинен мати такі функції:

- Швидка обробка інформації;
- можливість створення нових продуктів;
- створення замовлення на продукцію;
- класифікація продукції;
- перегляд інформації, створеної на сайті;
- перегляд каталогу товарів;
- зручна класифікація;
- можливість перегляду окремих сторінок товару;
- можливість вибору індивідуальних параметрів
- додати товар у корзину;
- розрахувати вартість і кількість товарів у кошику;
- обробляти заявки клієнтів на стороні адміністратора;
- навігація на сайті має бути зручною та працювати належним чином;
- можливість перегляду умов інтернет-магазину;
- легкий доступ до довідкової інформації про роботу системи, яка може знадобитися користувачам;
- зручний та інтуїтивно зрозумілий інтерфейс, що відповідає сучасним стандартам розробки сайтів.

У процесі розробки проекту користувачі інтернет-магазину автозапчастин можуть виділити 2 типи користувачів:

Клієнти – будь-який відвідувач сайту, який не має доступу до

адміністрування сайту, вважається потенційним клієнтом. Користувачі можуть переглядати створений адміністратором вміст, списки продуктів тощо. Основні типи системних конструкцій інтернет-магазинів автозапчастин [13].

Адміністратор – можливість керувати сайтом, додавати та обробляти інформацію про клієнтів, переглядати додаток, маніпулювати асортиментом магазину тощо [14].

Висновки за розділом 1

Отже в усіх наявних сервісах пошуку автомобільних запчастин не реалізована функція пошуку за неповним пошуком. Данна функція є досить важливою, тому що іноді номер деталі може бути затертий, або деталь може бути зламана, що не дозволяє ввести її ідентифікатор повністю. Пошук за неповним описом вирішує цю проблему та може допомогти багатьом користувачам знайти потрібну деталь.

2. РОЗРОБКА АЛГОРИТМУ ЗІСТАВЛЕННЯ ШАБЛОНІВ ПОШУКУ З ВХІДНИМИ РЯДКАМИ

2.1 Формалізована постановка задачі

На вхід подається набір шаблонів пошуку:

$$P = \{p_1, p_2, \dots, p_m\}. \quad (2.1)$$

Кожен шаблон пошуку p_1 має рядкову форму символів, що належать до алфавіту, і спеціальних символів, що не належать до цього алфавіту, які також називаються символами підстановки:

«?» − відповідає будь-якому символу в алфавіті ;

"*" відповідає будь-якому рядку в алфавіті , включаючи порожній рядок. Тобто p_i можна виразити як:

$$p_i = c_{i1}k_{i1}c_{i2}k_{i2}c_{i3} \dots c_{il_i}k_{il_i} *, \quad (2.2)$$

де c_{ij}— символ підстановки, а k_{ij}— рядок ключа алфавіту, c_{i1} має додаткову умову - його можна опустити або дорівнювати "*".

Далі на вхід надходить запит такого вигляду: рядок S, що складається з n символів. Кожен такий рядок повинен знайти всі шаблони пошуку, що відповідають цьому рядку з P, тобто отримати набір:

$$A = \{p_{a_1}, p_{a_2}, \dots, p_{a_{anslen}}\}. \quad (2.3)$$

Кількість запитів не обмежена, потрібно відповідати на запити по мірі надходження.

2.2 Опис методів розв`язання задачі

Алгоритми для вирішення проблеми зіставлення шаблонів пошуку з вхідними рядками базуються на структурах даних, таких як використання префіксних дерев [15] і кінцевих автоматів, створених на основі модифікованого алгоритму Ахо-Корасіка [16].

Дерево префіксів будується поверх вхідних шаблонів, побудова цього дерева є нестандартною, подібною до побудови "стислого" дерева префіксів, оскільки кожен шаблон представлений у вигляді рядка ключових слів і спеціальних символ «*» і «?» між ними, тобто кожне ребро в цьому дереві

містить інформацію про ключ перетворення та спеціальний символ, який йому передує. Дерево будується тільки один раз при отриманні шаблону пошуку і залишається незмінним при подальшій роботі алгоритму.

Щоб зменшити споживання пам'яті та для простоти представлення, кожному ключовому слову в наборі всіх ключових слів W шаблонів P буде присвоєно символ в алфавіті Δ , тобто існує однозначне відображення, яке:

$$f: \delta_j \to w_j \,, \tag{2.4}$$

$$g: w_j \to \delta_j . \tag{2.5}$$

Для швидкого пошуку ключових слів у тексті вхідних запитів буде використовуватися автомат, побудований за алгоритмом Ахо-Корасіка. Автомати будуть попередньо зібрані на основі ключових слів із W і створені лише один раз. Щоб прискорити пошук ключових слів, які є префіксами інших ключових слів, до автомата буде додано посилання на термінали. Це дозволить знайти всі ключові слова в тексті S за час, лінійний кількості ключових слів у тексті. Щоб завчасно відсікати неможливі варіанти, до кінцевого стану автомата додаються посилання на вузли дерева префіксів, які можуть відповідати початку деякого шаблону пошуку. Крім того, коли надходить новий рядок S, він сканується зліва направо, створюючи набір префіксів можливих шаблонів пошуку використовуючи інформацію про переходи у префіксному дереві.

2.3 Опис структури даних префінсного дерева

Prefix tree або trie — це дерево пошуку [17] — це структура даних, яка виглядає як дерево і може використовуватися для зберігання динамічного набору рядків. На відміну від інших дерев пошуку (наприклад, бінарних дерев пошуку), вузли префіксного дерева не зберігають ключі, а основну роль його вмісту відіграють ребра – усі переходи між вузлами позначаються якимись символами. Усі нащадки вузла мають загальний префікс рядка, пов'язаного з цим вузлом; корінь асоціюється з порожнім рядком. Ключі знаходяться в листках префіксного дерева, що відповідає певному рядку, і внутрішні вузли також

можуть містити ключ, якщо цей ключ є префіксом інших ключів у цьому дереві. Крім того, для оптимізації споживання пам’яті дерева префіксів можуть бути представлені в «компактній» формі, що досягається стисненням вузлів лише з одним нащадком, у цьому випадку перетворення може містити не один символ, а весь рядок символів який буде відповідати переходу.

Так як префіксні дерева побудовані таким чином, що всі ключі зі спільним префіксом мають спільний вузол, що відповідає цьому префіксу, можна шукати без повного ключа, а лише за його префіксом, і результатом буде набір можливих ключів.

2.3.1 Варіанти практичної реалізації

Основна відмінність між реалізаціями полягає в методі переходу між вузлами зберігання. Вибір методу залежить від того, що ви намагаєтеся оптимізувати: час виконання, споживання пам’яті чи щось інше.

Наприклад, для менших алфавітів може бути доцільним зберігати масив для кожного вузла, де будуть записані переходи для кожного символу алфавіту, що збереже час відвідування наступного вузла якомога коротшим - O(1) , проте збільшує споживання пам’яті, оскільки не всі символи алфавіту містять перетворення.

Також можна шукати в дереві за допомогою red-black, що не збільшує споживання пам’яті та надасть O(log x) часового логарифмічного доступу до наступного вузла.

Іншим варіантом є використання хеш-таблиці, яка може виконувати операції доступу до елемента в середньому за O(1) часу та з асимптотично порівнянними витратами пам’яті на звичайні масиви. Хоча пошукова операція може виконуватися в O(x),на що впливає заповненність таблиці та способів хешування.

2.3.2 Вимоги до пам`яті та швидкодія

У варіанті з використанням масиву час пошуку ключа довжиною r є лінійним за його довжиною r - O(r). Припускаючи набір усіх ключів M, споживання пам’яті залежить від загальної довжини всіх ключів і розміру алфавіту, у гіршому випадку вони можуть складати $O(|\Delta||M|)$.

У разі використання червоно-чорного дерева час на пошук ключа дорівнює $O(r \log x)$, а споживання пам’яті дорівнює $O(|M|)$.

Для нашої задачі найбільше підходить використання хеш-таблиці, тому що ми будемо виконувати кілька операцій пошуку в дереві, і нам потрібно оптимізувати час виконання алгоритму. Крім того, для нашої проблеми значення $|\Delta|$ та $|m|$ можуть бути дуже великими, тому використання пам'яті $O(|\Delta||M|)$ неприйнятно, оскільки нам потрібно зберігати ці дані в оперативній пам'яті сервера, яка дуже обмежена. Оскільки ми створюємо дерево префіксів лише один раз, воно не буде змінено знову, тобто хеш-таблиця не буде видалена та додана, що дозволяє кожному окремому вузлу вибрати відповідний розмір хеш-таблиці. У свою чергу, це дозволяє оцінити час виконання пошуку ключів $O(r)$ і $O(|M|)$ споживання пам'яті для зберігання дерева префіксів.

Час побудови автоматів лінійно залежить від загальної довжини слів у словнику - $O(|W|)$. Оскільки автомат структурований як дерево суфіксів, обсяг необхідної пам’яті також лінійно залежить від загальної довжини слів у словнику - $O(|W|)$, тому для кожного стану ми маємо лише один префікс і одне кінцеве посилання.

Час обробки кожного нового рядка введення залежить лише від самого рядка введення та кількості збігів і словників у ньому. Складність алгоритму лінійно залежить від довжини рядка, оскільки кожен символ рядка перебирається один раз, і лінійно залежний від загальної довжини відповіді - отриманих збігів зі словника. Це $O(n + L)$, де n — довжина рядка, а L — загальна довжина відповіді. Зазначимо, що загальну довжину відповіді можна оцінити як $O(n^2)$, так як у вхідному рядку може бути квадратична кількість збігів [18].

2.4 Побудова автомата за алгоритмом Ахо-Корасік

Алгоритм Ахо-Корасика — це алгоритм пошуку рядка, який дозволяє знаходити у вхідному рядку елементи деякого словника. Алгоритм знаходить збіг під час читання вхідного тексту та негайно повертає знайдений збіг. Тобто алгоритму, не потрібно заздалегіть бачити заздалегіть весь рядок[19].

Сочатку алгоритму на основі рядків у словнику побудуйте скінченний автомат . Ця машина схожа на префіксне дерево з додатковими зв'язками - так званими суфіксними посиланнями. Таке посилання вказує на найдовший суфікс лінії поточного стану, який існує в цьому автоматі, а за відсутності такого суфікса посилання вказує на початковий стан (корінь). Поведінка автомата може бути описана трьома функціями: функцією переходу, функцією відмови та функцією виходу. Функція переходу для кожного наступного символу введення в рядку вказує, до якого стану потрібно перейти, або повідомляє, що такого переходу не існує. Якщо переходу немає, використовується функція fail, яка працює так: поки немає переходу для стану за допомогою функції переходу, ми переходимо через ланцюжок суфіксів. Повторюйте ці переходи суфіксних посилань, якщо необхідно, до досягнення початкового стану. Функція виводу дозволяє вам перевірити, чи кожен стан відповідає рядку в словнику, і повернути його як результат.

Щоб прискорити пошук рядків словника у рядках введення, ви можете додати додаткові посилання - кінцеві (або дійсні) посилання. Кожен стан вказуватиме на інший стан, рядок якого є найдовшим суфіксом рядка поточного стану, що відповідає одному з рядків у словнику. Посилання, такі як суфіксні посилання, можуть бути попередньо обчислені під час створення автомата. Це дозволить швидко знайти всі слова в словнику, які закінчуються на цей символ у поточному рядку введення під час обробки наступного символу.

Зі словником, відомим заздалегідь, побудова автомата може бути виконана лише один раз, і автомат можна повторно використовувати, перемикаючись у початковий стан для кожного нового рядка введення.

2.5 Опис кроків алгоритму

Опис кроку алгоритму Алгоритм зіставлення шаблонів пошуку з вхідними рядками розроблено на основі структури даних «префіксного дерева» та скінченних автоматів, побудованих відповідно до алгоритму Ахо-Корасіка. Алгоритм можна розділити на дві основні частини - попередню обробку шаблону пошуку та інтерактивну частину відповіді на запит, вхідний рядок [20].

2.5.1 Алгоритм попередньої обробки шаблонів пошуку

Розглянемо кроки запропонованого алгоритму:

Крок 1. Створимо набір усіх унікальних ключових слів W. Для кожного шаблону пошуку p_i ми додаємо k_{ij} до набору:

$$W = \{w_j : j \in J\} = \bigcup_{i=1}^{m} \{k_{ij} : 1 \le j \le l_i\}. \quad (2.6)$$

Крок 2. Зіставимо кожне w_j з W з деяким числом δ_j, щоб отримати біективне відображення:

$$f : \delta_j \to w_j, \quad (2.7)$$

$$g : w_j \to \delta_j. \quad (2.8)$$

Крок 3. Побудуємо множину B – трансформований шаблон пошуку, замінивши ключові слова в p_i на символи алфавіту Δ, де $|\Delta| = |W|$, і видаливши c_{i1}, якщо він існує:

$$B = \{b_i = g(k_{i1})c_{i2} \dots g(k_{il_i}) : 1 \le i \le m\}. \quad (2.9)$$

Крок 4. На основі B побудуємо префіксне дерево T наступним чином: розглядаємо кожен $g(k_{ij})$ як звичайний символ алфавіту Δ , якщо c_{ij} дорівнює «*» або не існує - будемо ребро зі знаком $g(k_{ij})$, яке додається до цього дерева префіксів. Так як розмір алфавіту може бути дуже великим, у кожному вузлі будемо зберігати хеш-таблицю зі списком ребер, а символ $g(k_{ij})$, як ключ для пошуку. Це дозволить вам швидко здійснювати переходи вздовж певних країв - символу $|\Delta|$ в алфавіті. Якщо c_{ij} дорівнює "?", тоді ми приділяємо додаткову увагу таким ребрам, щоб відрізнити їх від "нормальних" країв.

КРОК 5. Ми побудуємо скінченний автомат на основі ключових слів у W відповідно до алгоритму Ахо-Корасіка. Для кожного стану, що відповідає деякому ключу w_j, ми будемо додатково зберігати δ_j. Переходи для кожного стану, позначені буквеним символом , будуть зберігатися в хеш-таблиці.

Крок 5.1. Для всіх станів автомата, з яких ми можемо перейти в стан, що відповідає певному ключовому слову, яке W переміщує вздовж суфіксного посилання, ми попередньо обчислюємо «остаточні» termLink до найближчого такого стану.

Крок 5.2. Для стану автомата, відповідного ключовому слову w_j, такого, що він зустрічається в будь-якому шаблоні пошуку на першій позиції та не передує символом узагальнення, ми додатково збережемо посилання headlink на відповідний вузол у дереві префіксів T, яке доступне через корінь та переміщення здійснюється по $g(w_j)$.

Крок 5.3. Для стану автомата, що відповідає ключовому слову w_j, яке з’являється на першій позиції будь-якого шаблону пошуку та передує символу узагальнення «*», ми додатково збережемо looseHeadLink до відповідного вузла в дереві префіксів T, яке доступне через корінь та переміщення здійснюється по $g(w_j)$.

2.5.2 Алгоритм обробки вхідних запитів

Після попередньої обробки шаблону пошуку (перші п'ять кроків алгоритму) алгоритм може прийняти запит на аналіз тексту - вхідний рядок S. Для кожного окремого рядка S необхідно зробити наступне:

Крок 6. Створимо порожній масив, де будемо зберігати пари значень: індекс рядка S і посилання на вузол префіксного дерева T. На початку операції стан автомата відповідає кореневому вузлу.

КРОК 7. Скануємо рядок S зліва направо та виконуємо наступні кроки для наступного символу s_{idx} , де $1 \leq idx \leq n$.

Крок 7.1. Скористаємося функцією переходу в алгоритмі Ахо-Корасіка,

щоб перейти з поточного стану автомата в новий стан за символом s_{idx}.

Крок 7.2. Якщо поточний стан відповідає якомусь ключу w_j і $|w_j|$=idx, тоді додаємо пару до q.

Крок 7.3. Якщо поточний стан має посилання looseHeadLink, додаємо пару (idx,curState.looseHeadLink) до q.

Крок 7.4. Якщо поточний стан відповідає ключовому слову w_j і |w_j|≠idx, тоді lb=idx-|w_j|. Для всіх пар із q і q_i.idx≤lb перевіримо, чи є перехід від вузла дерева T до $g(w_j)$

Крок 7.4.1. Якщо перехід існує, а край не позначений знаком «?», то ми виконаємо перехід уздовж нього, додавши пару idx і вузол до q.

Крок 7.4.2. Якщо перехід існує і ребро позначене знаком «?», ми додатково перевіряємо q_i.idx=lb-1 і додаємо пару idx і вузол (в який він був перетворений) до q.

Крок 7.4.3. Якщо до q додається нова пара значень, ми перевіряємо, чи відповідає вузол T дерева в цій парі певному шаблону пошуку, і якщо так, то додаємо цей шаблон пошуку до відповіді A.

Крок 7.5. Якщо в поточному стані є termLink, то ми використовуємо це посилання для переходу та повертаємося до кроку 7.3.

Крок 8. Повернутися до знайденого набору шаблонів пошуку A і завершити роботу алгоритму.

2.6 Контрольний приклад роботи алгоритму

Припустимо є алфавіт $\Sigma = \{c_1, c_2, c_3, c_4, c_5\}$ та на початку подається множина шаблонів пошуку з табл. 2.1.

Таблиця 2.1 – Шаблони пошуку

p_1	$* c_1 * c_1 c_3$
p_2	$* c_1$
p_3	$c_1 c_2 * c_4 c_2 c_5$

Продовження таблиці 2.1

p_4	$c_2c_5 * c_1c_3 * c_4c_2c_5$
p_5	$c_1c_2 * c_2c_5 * c_4c_2c_5$
p_6	$c_2c_5 * c_1 * c_2c_5$
p_7	$c_2c_5 * c_4c_2c_5$

Виділимо ключові слова з шаблонів пошуку та занесемо їх до табл. 2.2:

Таблиця 2.2 – множина ключових слів

w_1	c_1
w_2	c_1c_2
w_3	c_1c_3
w_4	c_2c_5
w_5	$c_4c_2c_5$

Побудуємо множину перетворених шаблонів та занесемо їх до табл. 2.3:

Таблиця 2.3 – множина перетворених шаблонів пошуку B

b_1	$* w_1 * w_3$
b_2	$* w_1$
b_3	$w_2 * w_5$
b_4	$w_4 * w_3 * w_5$
b_5	$w_2 * w_4 * w_5$
b_6	$w_4 * w_1 * w_4$
b_7	$w_4 * w_5$

Використовуючи В отримуємо превіксне дерево Т – рис. 2.1

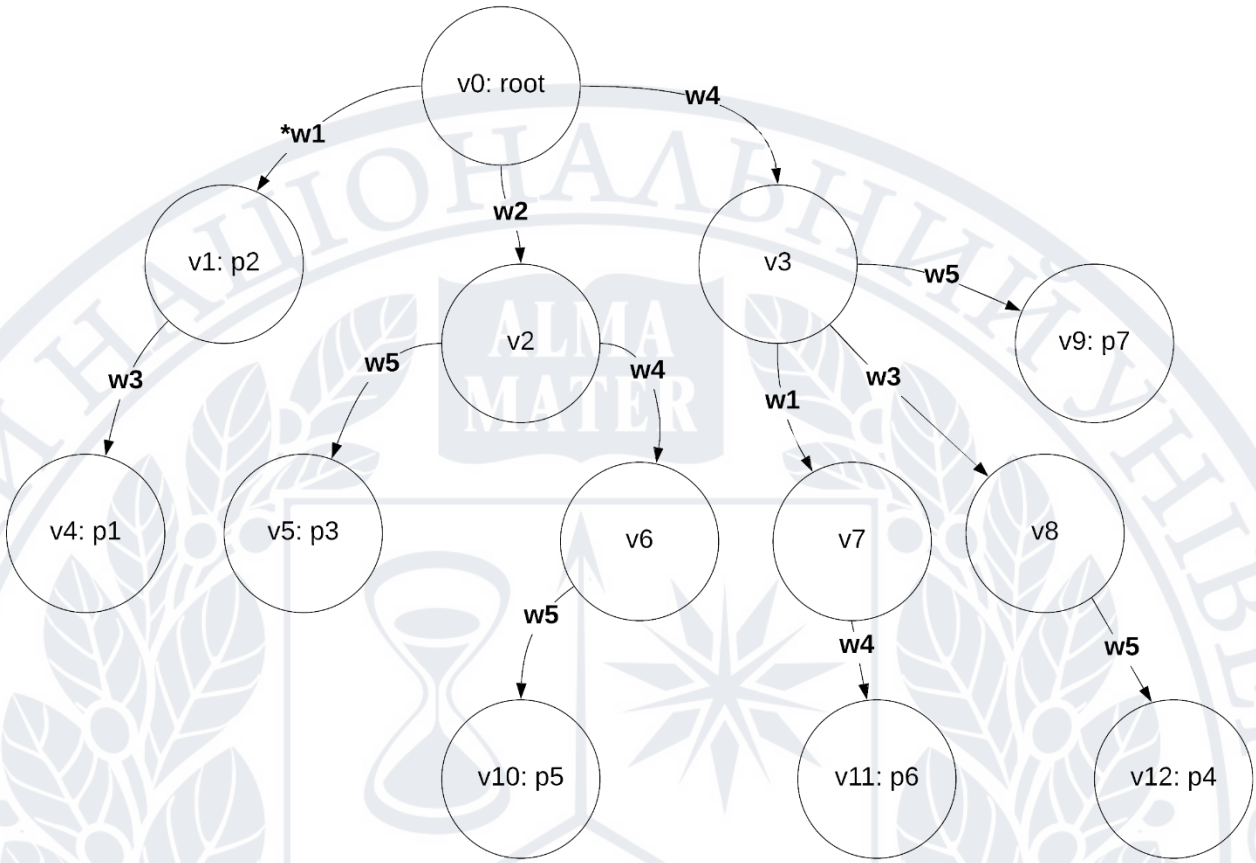


Рисунок 2.1 – Отримане префіксне дерево Т

Використовуючи набір ключових слів W і префіксне дерево Т, побудуємо скінченний автомат за модифікованим алгоритмом Ахо-Корасіка – рис. 2.2. Для спрощення картини автомата на рисунку не показано суфіксальних посилань на початковий стан.

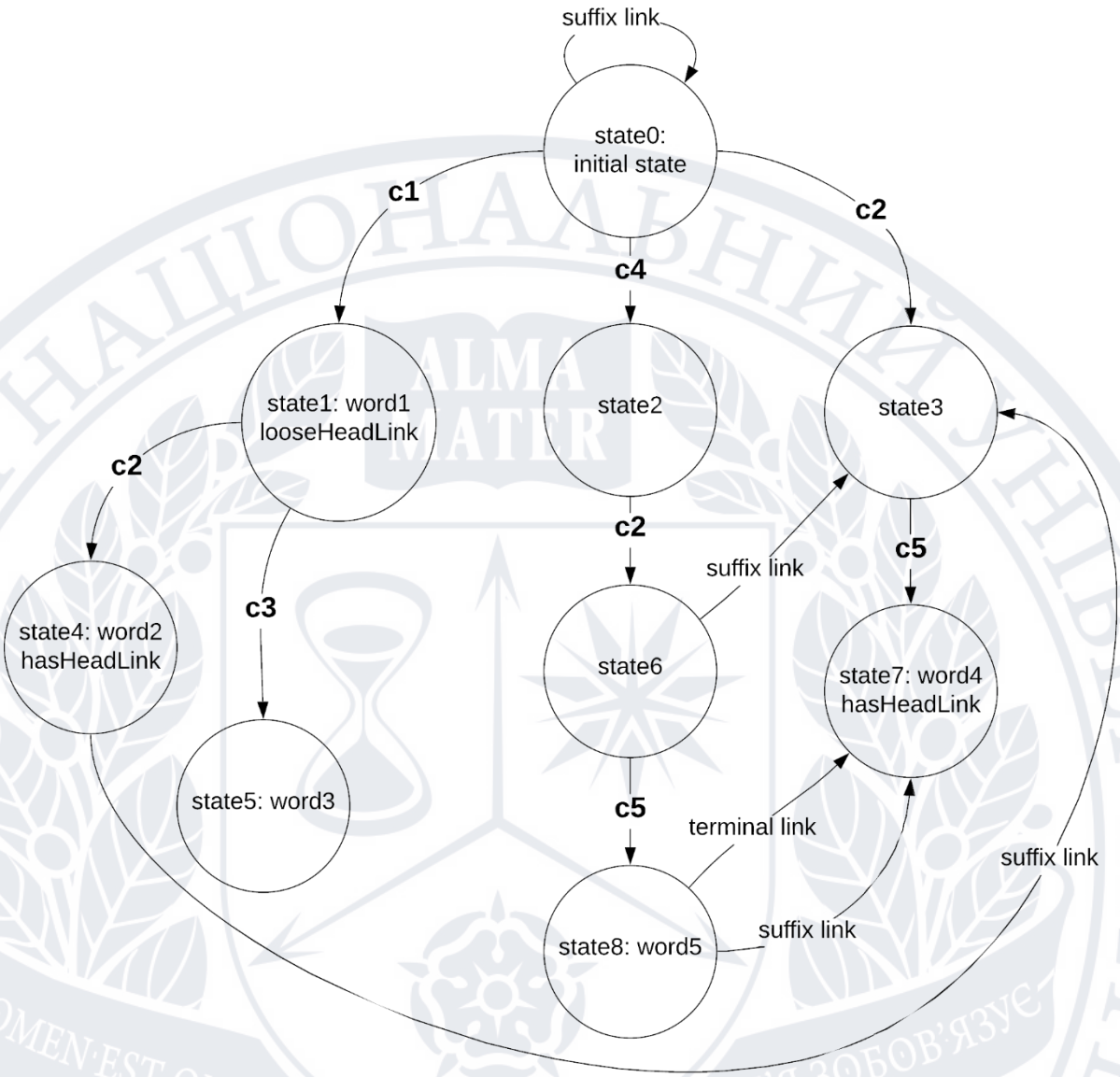


Рисунок 2.2 – Автомат за алгоритмом Ахо-Корасік

На наступному кроці отримаємо вхідний рядок:

$$S = c_2c_5c_5c_5c_1c_2c_4c_3c_3c_4c_2c_5c_2c_1c_3c_4c_2c_5 \quad (2.10)$$

Виділивши ключові слова, зобразимо його на рис. 2.3.

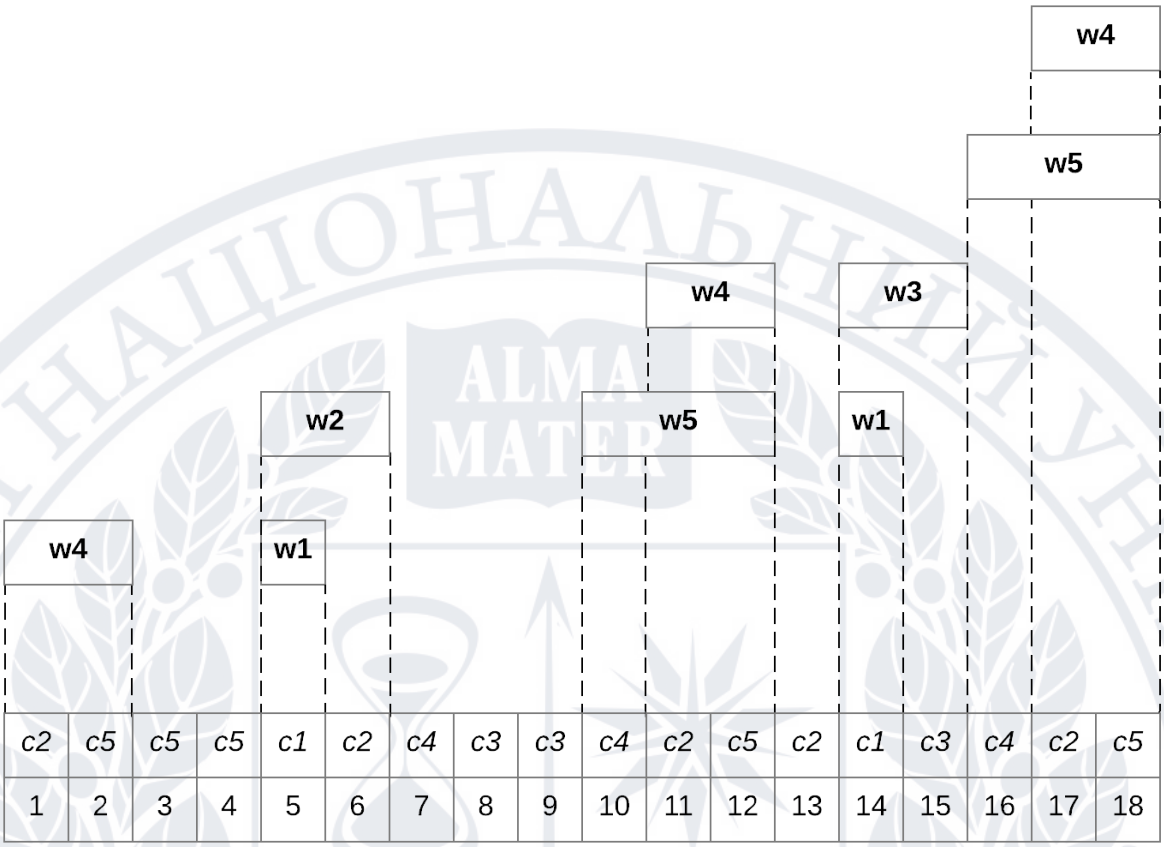


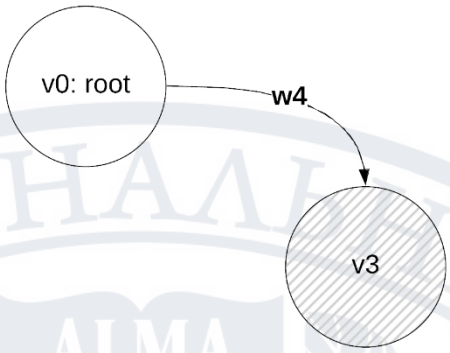
Рисунок 2.3 – Вхідний рядок S

Почнемо рухатися зліва направо по лінії S, спочатку масив q не має ніяких значень. Елементи, додані до q, можна розглядати як неявну конструкцію піддерева префіксного дерева T, для кожного наступного символу S, що містить усі префікси дерева T, побудованого з використанням перших символів S.

Коли зчитується другий символ, ми входимо в стан машини state7 і бачимо, що він відповідає ключовому слову. Оскільки це ключове слово має мітку hasHeadLink, це означає, що воно може бути у шаблоні пошуку першим ключовим словом та має бути на самому початку рядка. Оскільки автомат вже додатково зберігає посилання на відповідний вузол у префіксному дереві T – додамо цей вузол до масиву q.

З раніше побудованим префіксним деревом T - зберігаємо співпадіння за префіксом поточного рядка – рис. 2.4.

Рисунок 2.4 – Додані вузли після зчитування та обробки другого символу



При обробці п'ятого символу автомат знаходиться в стані1, що відповідає ключовому слову w_1. Цей стан має ключове слово terminalLink, позначене як looseHeadLink, і тому його можна використовувати як початок шаблону пошуку будь-де у вхідному рядку S. Тому додаємо вузол v_1, що відповідає цьому початковому ключу, до масиву q. Як ми бачимо, v_1 відповідає повному шаблону пошуку p_2 - тобто один із шаблонів пошуку вже знайдено в рядку S. Також перевіряємо, чи є перехід від вузла, доданого до q через ключ w_1. Такий перехід є від v_3 до v_7, додамо ці вузли до q (рисунок 2.5). На шостому символі маємо інший збіг, ключове слово w_2, але оскільки шаблон пошуку, що починається з w_2, не може передувати будь-який символ w_2, і вузол у q не має переходів за цим ключовим словом - тоді q залишається незмінним.

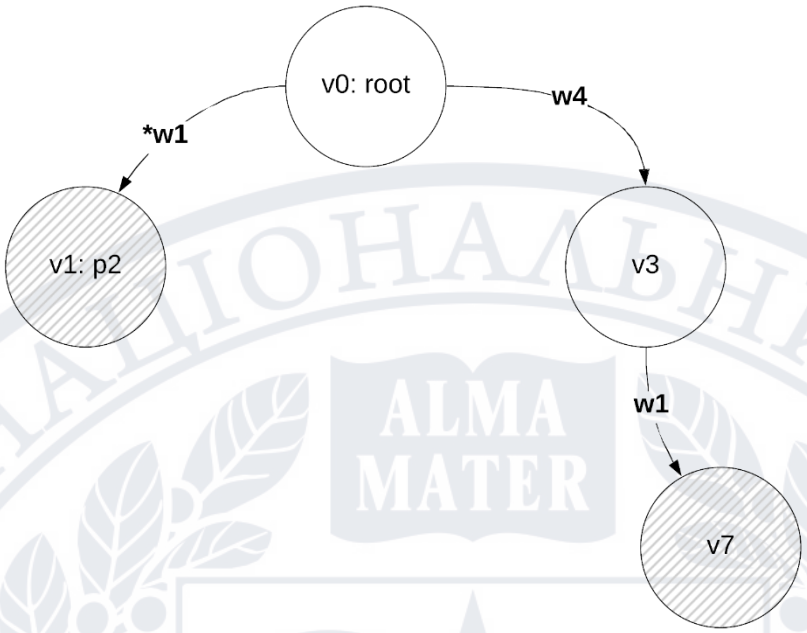


Рисунок 2.5 – Отримані вузли після оброки п`ятого сиволу

При обробці дванадцятого символу автомат знаходиться в стані8, що відповідає ключовому слову w_5. Цей стан має посилання (terminalLink) на стан state7, що відповідає ключовому слову w_4. По-перше, розберемося з w_5 - немає шаблону пошуку, що починається з цього ключового слова, але є вузол v_3 серед вузлів у q, який переходить до v_9 після w_5. Вузол v_9 відповідає шаблону пошуку p_7, і додаємо другий знайдений шаблон пошуку до відповіді. І передаємо ключове слово w_1. Оскільки переходів до w_5 більше немає, перейдемо до останнього посилання w_4. Як ми бачимо, у q є вузол v_7, з якого можна перейти до v_{11} за допомогою ключа w_4. Цей вузол відповідає третьому знайденому шаблону пошуку p_6. Після виконання всіх алгоритмічних операцій над дванадцятим символом масив виглядає так - рис. 2.6:

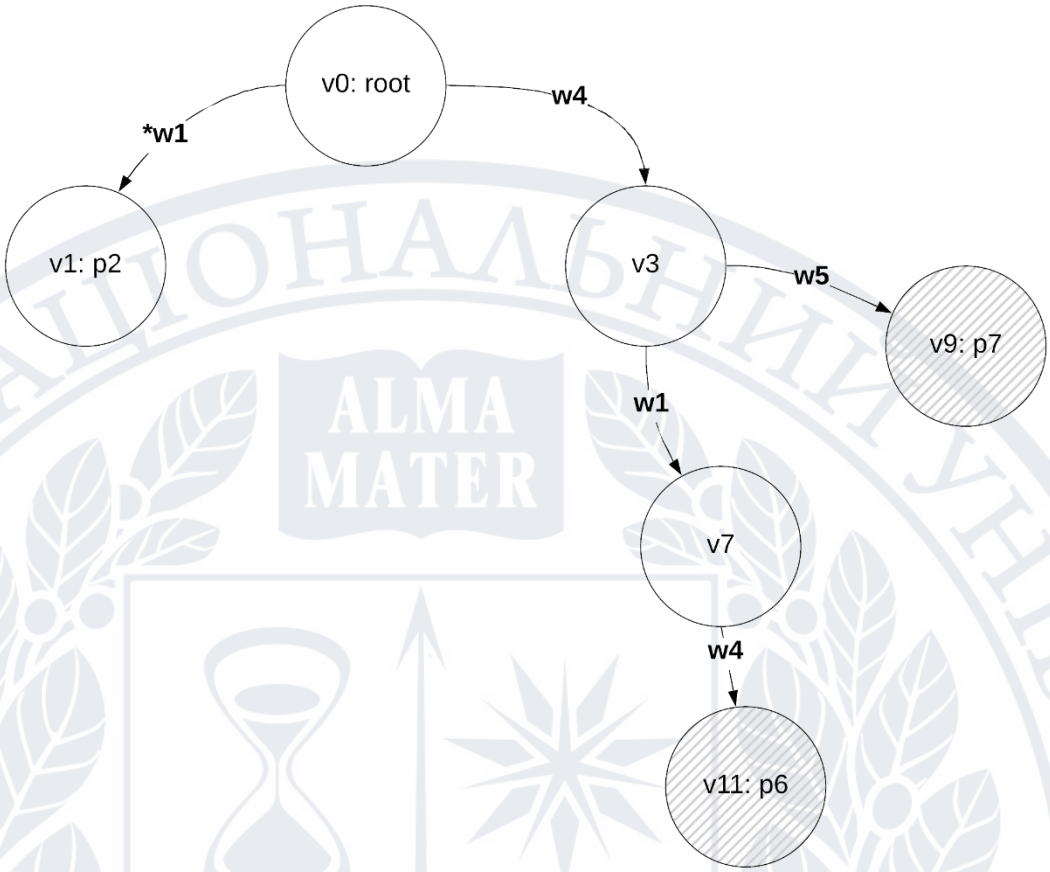


Рисунок 2.6 – Вузли після оброки дванадцятого символу

У символі 14 у нас є ключ w_1, але немає переходу на w_1 між вузлами в q, які ведуть нас до нового вузла, якого ще немає в q, тому до q не додаються дублікати.

Після обробки символу 15 - автомат знаходиться в стані5, що відповідає ключовому слову w_3. Немає шаблону пошуку, що починається з цього символу, але є вузли в q, які передаються з w_3, а саме вузли v_1 і v_3. Тобто ми отримаємо два нових вузла: v_4 і v_8. Вузол v_4 відповідає шаблону пошуку p_1. Після завершення платежу масив q має наступний вигляд - рис. 2.7:

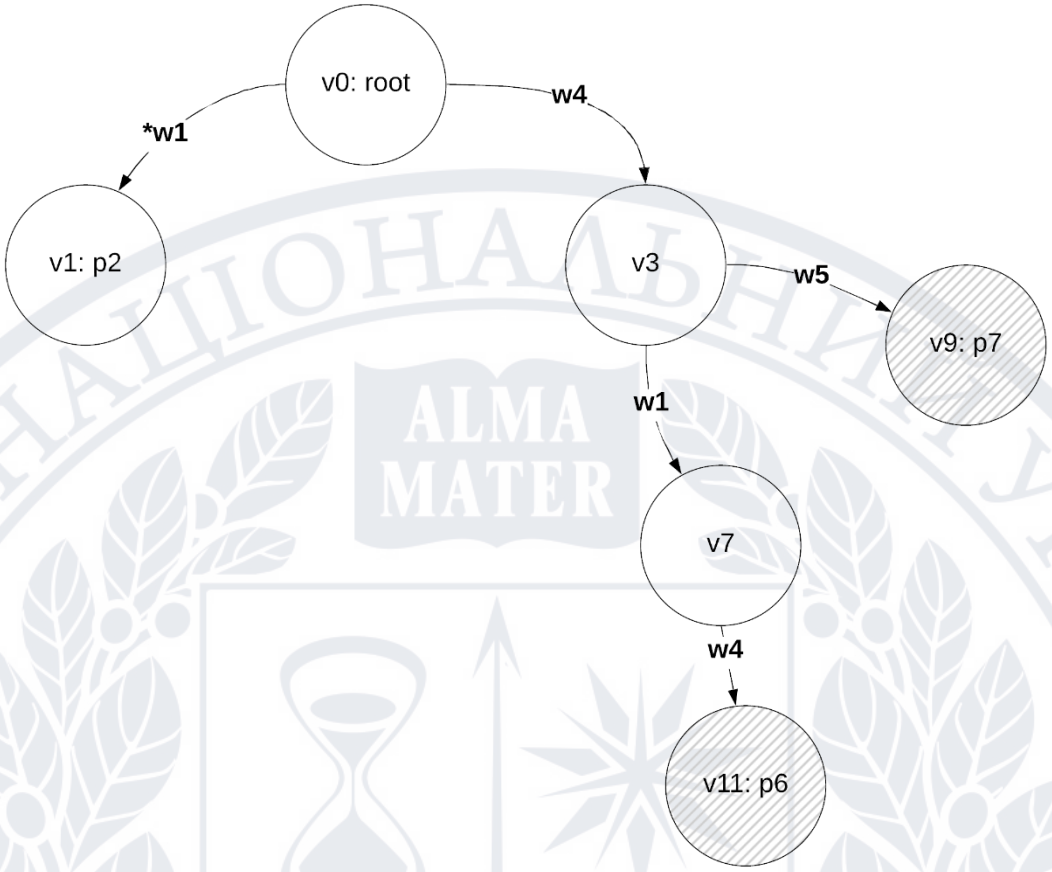


Рисунок 2.7 – Вузли після п`ятнадцятого символу

Під час обробки останнього символу рядка S автомат знаходиться в стані 8, що відповідає ключовому слову w_5. Серед елементів q є вузол v_8, з якого вузол v_{12} можна отримати через w_5. Вузол v_{12} відповідає шаблону пошуку p_4, тобто інший збіг знайдено в рядку S з більш ніж одним шаблоном пошуку. Отримуємо п’ять збігів.

Стан state8 нарешті посилається на стан state7, який відповідає ключовому слову w_4. Але серед елементів q немає такого вузла, щоб перейти від w_4 до нового, ще не відвіданого вузла. Після завершення алгоритмічної операції над останнім символом масив q має такий вигляд - рис. 2.8:

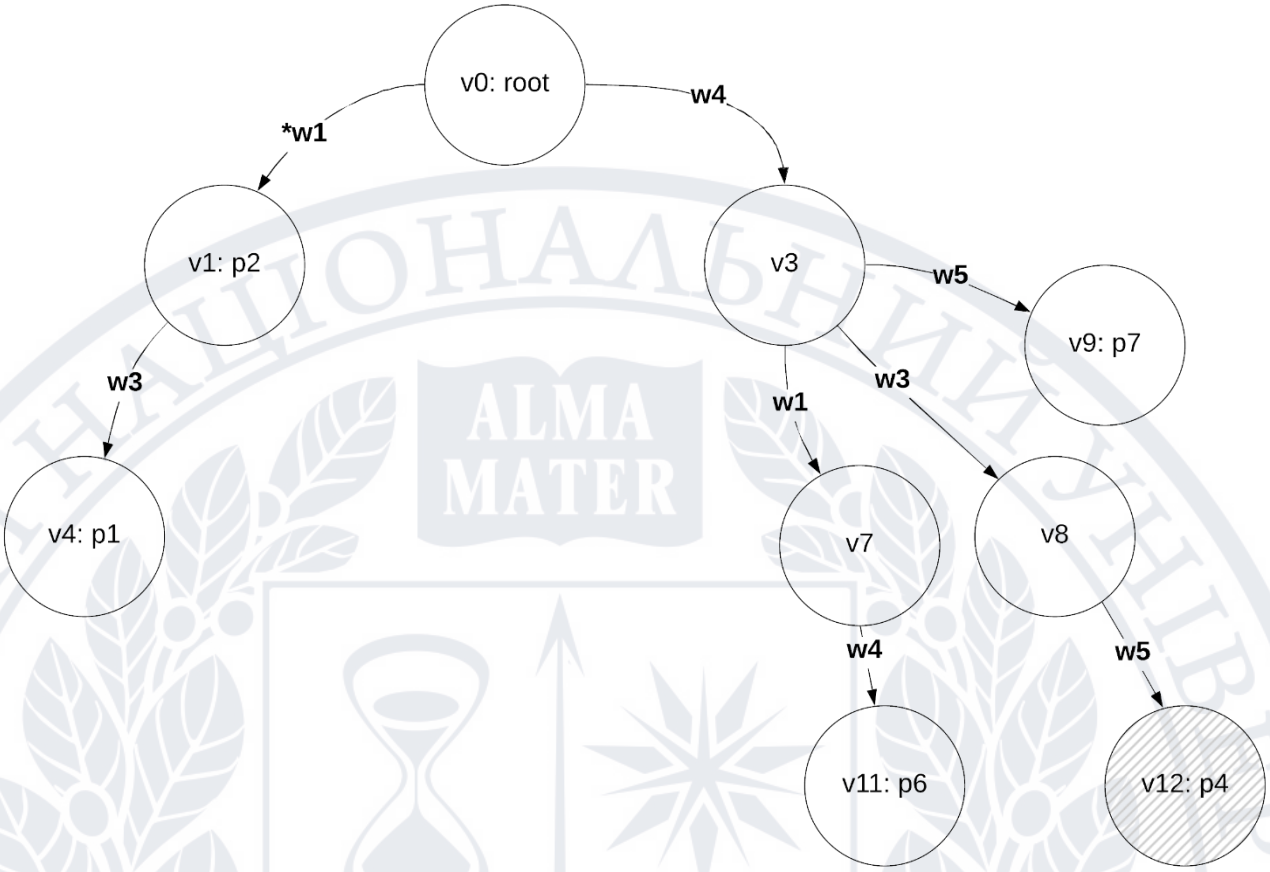


Рисунок 2.8 – Вузли після оброки 18 символу

Отже після виконання алгоритму видно, що автомат залишився без змін, було виконано лише перехід зі стану в стан. А також не змінювалось префіксне дерево, тому що було використано лише інформацію про переходи.

2.7 Обґрунтування безпомилкової роботи алгоритму

Так як під час роботи алгоритму для пошуку ключових слів у S-рядку використовується автомат, побудований за алгоритмом Ахо-Корасіка, то гарантовано буде проведено пошук усіх ключових слів, тобто в процесі аналізу S-рядка не буде пропущено ключове слово. Під час обробки рядка S неявно будується піддерево префіксного дерева T, перевіряючи, чи може кожен наступний ключ збільшити піддерево шляхом додавання вузла, наявного в T. Тобто під час роботи алгоритму ми проходимо всі можливі шляхи в дереві префіксів T, що гарантує знаходження всіх можливих шаблонів пошуку в S, оскільки всі вони знаходяться в T.

2.8 Оцінка складності алгоритму

Оцінку складності алгоритму можемо розділити на дві частини: оцінку обробки наступного рядка S, яка відбувається під час виконання програми так аналіз початкової обробки шаблону пошуку, яка виконується лише один раз під спочатку програми. У наступних двох розділах будуть розглянуті оцінки часу та оцінки споживання пам’яті для двох частин алгоритму відповідно.

2.8.1 Аналіз часу виконання алгоритму

Спочатку розглянемо етап попередньої обробки шаблону пошуку. Створюємо набір усіх ключових слів у O(|M|), де |M| — загальна довжина всіх шаблонів пошуку. На другому кроці потрібно відсортувати набір ключів і визначити для кожного символу ключа відповідне йому число, а для подальшого доступу до цих відображень треба- створити хеш-таблиці – загалом O(|W| log |W|). На кроці 3 будуємо B у вартості O(|M|+|W| log |W|). Побудова дерева префіксів коштуватиме O(|M|). Час побудови кінцевого автомата на кроці 5 оцінюється як O(|W|). Тобто, загалом, враховуючи залежності між змінними, загальна оцінка не перевищуватиме O(|M|+|W| log |W|).

Тепер перейдемо до оцінки часу обробки рядка S, що надходить під час виконання програми. Під час роботи алгоритму повторюємо всі екземпляри ключового слова, знайденого в рядку S, роблячи їх число рівним occ. Для кожного знайденого ключового слова та перевіряємо всі можливі стани в q, з яких можна здійснювати переходи до цього ключового слова. Загальна кількість станів — це число префіксів із T, знайдених у рядку, який перед ключовим словом. Отримуємо, що часова складність алгоритму може бути оцінена як O(occ*prefnum*log (|W|)+anslen). У гіршому випадку prefnum може мати значення O(|T|), якщо ми не враховуємо повторення в алгоритмі як можливі різні відповіді, оскільки для нас це не має сенсу, і оцінка може стати експоненціальною. Але, наприклад, у випадку, коли ми не збігаємося з префіксом T, нижня оцінка дорівнює (occ).

Отримуємо, що в середньому алгоритм працює швидко, коли вхідний

рядок S не має надто багато ключових слів і шаблони пошуку здебільшого не є підпослідовностями один одного.

2.8.2 Аналіз витрат пам`яті алгоритму

Попередня обробка шаблонів пошуку на перших двох кроках споживає O(|W|) пам'яті для створення ключа. Далі, третій крок споживає O(|M|) пам'яті для побудови перетвореного набору шаблонів пошуку B. Четвертий крок споживає O(|M|) пам’яті для побудови дерева префіксів, а п’ятий крок споживає O(|W|) пам’яті для побудови автомата згідно з алгоритмом Ахо-Корасіка. Таким чином, загальна вартість пам’яті лінійна із загальною довжиною шаблону пошуку, тобто оцінюється як O(|M|). Зауважимо, що фактичне споживання пам’яті може бути значно нижчим, та залежить від того, наскільки схожими є префікси цих шаблонів пошуку, а також залежно від кількості унікальних ключів у шаблонах пошуку та. При обробці рядка S пам’ять витрачається на зберігання масиву вузлів q– дублікати не враховуються. Тобто споживання пам'яті лінійно залежить від кількості префіксів з префіксого дерева, знайдених у рядку.

Висновки за розділом 2

Запропоновано алгоритм для пошуку за наповним описом. Алгоритм підходить для задачі пошуку за неповним описом, так як під час роботи алгоритму для пошуку ключових слів у S-рядку використовується автомат, побудований за алгоритмом Ахо-Корасіка, то гарантовано буде проведено пошук усіх ключових слів, тобто в процесі аналізу S-рядка не буде пропущено ключове слово. показана коректність роботи алгоритму на контрольному прикладі із запиту н 18 символів, при чому швидкість є високою.

3. ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Фреймворки для реалізації системи

Фреймворки — це структури, завдяки яким можна створювати програмне забезпечення. Вони служать основою, тому не доводиться починати з нуля. Фреймворки зазвичай пов'язані з конкретними мовами програмування і підходять для різних типів завдань [21].

Використання фреймворку економить час і знижує ризик помилок. Немає необхідності писати все з нуля, тому менше шансів на помилки. Крім того, фреймворк був протестований, тому немає про що турбуватися. Серед інших переваг [22]:

- Допомога у встановленні найкращих практик програмування;
- адаптивне використання шаблонів проектування;
- код більш безпечний;
- можна уникнути дублювання та надлишкового коду;
- сприяють послідовній розробці коду з меншою кількістю помилок;
- полегшують роботу над складними технологіями;
- кожен може створити власний фреймворк або зробити внесок у фреймворк із відкритим кодом. тому функціональність постійно вдосконалюється;
- кілька фрагментів коду та функціональні можливості попередньо зібрані та протестовані. це робить програму більш надійною;
- тестувати та налагоджувати код легше, це можуть зробити навіть розробники, які не займаються кодом;
- значно скорочується час розробки програм.

Деякі можуть думати про структуру програмного забезпечення як про набір бібліотек, так само як бібліотека — це набір попередньо скомпільованих процедур. Однак це не так, оскільки не вся інфраструктура програмного забезпечення використовує або покладається на бібліотеки [23]. Різниця між бібліотекою та фреймворком полягає в тому, що остання викликає код. Натомість код викликає бібліотеку.

Деякі проблеми щодо фреймворку [24]:

- Нестандартний характер фреймворку не дозволяє програмістам мати глибоке розуміння мови програмування;
- варіанти налаштування функцій обмежені;
- іноді розробка додатків є важкою за допомогою фреймворків;
- потрібно вибрати правильну структуру для масштабу вашої програми, інакше це може негативно вплинути на продуктивність і досвід користувача;
- чітко відокремити бізнес-логіку від рівня представлення в mvc іноді важко;
- важливо знати, що нового/застарілого в кожному випуску;
- на додаток до вищезазначеного, проблеми, пов'язані зі структурами веб-розробки, включають;
- залежність від версії браузера;
- фреймворк javascript запускаються лише в середовищі перегляду, які підтримують javascript;
- якщо під час розробки не дотримуватись встановлених інструкцій, це може призвести до вразливості системи безпеки.

Розробники повинні шукати структуру, яка найкраще відповідає їхнім потребам. Будь то розробка веб-сайтів, наука про дані, керування базами даних або мобільні додатки, програмна інфраструктура доступна для всіх типів програмування.

Існує багато типів інфраструктури програмного забезпечення, які полегшують розробку додатків у різних областях розробки додатків. Детально розглянемо деякі з популярних сьогодні фреймворків програмного забезпечення:

Популярні java фреймворки:

Spring Framework — це потужний легкий фреймворк розробки програм для Enterprise Java (JEE) [25].

Основні функції Spring Framework можна використовувати для розробки будь-якої програми Java. Він описується як повна модульна структура. Фреймворк можна використовувати для всіх реалізацій рівня додатків у реальному часі. Його також можна використовувати для розробки певного рівня програм реального часу, на відміну від інших фреймворків, але за допомогою Spring ми можемо розробляти всі рівні [26].

Spring і всі модулі, включаючи Spring MVC, Spring Core, Spring Security, Spring ORM тощо, використовуються для корпоративних програм.

Hibernate ORM — це стабільна структура об'єктно-реляційного відображення для Java. Це забезпечує кращий зв'язок між мовою програмування Java та системою керування реляційною базою даних (RDBMS) [27].

Коли використовується така об'єктно-орієнтована мова, як Java [28], стикаємося з проблемою, що називається невідповідністю об'єктно-реляційного імпедансу, також відомою як невідповідність парадигми. Це пояснюється тим, що мови ОО та RDBMS обробляють дані по-різному, що може призвести до серйозних невідповідностей. Отже, цей Hibernate дає вам структуру для подолання невідповідностей Java.

Google Web Toolkit (GWT) — це повністю безкоштовна структура з відкритим вихідним кодом, яка допомагає розробникам писати код Java на стороні клієнта та використовувати його як JavaScript. Багато продуктів Google, як-от AdSense, Google Wallet і Blogger, створено за допомогою GWT [29].

Використовуючи GWT, розробники можуть легко та швидко писати складний код браузерної програми. Це також дозволяє розробляти та налагоджувати програми Ajax за допомогою Java. Дивовижна річ у GWT полягає в тому, що ви можете писати складні додатки на основі браузера без необхідності володіти зовнішніми технологіями, такими як оптимізація JavaScript або адаптивний дизайн.

JavaServer Faces (JSF) був розроблений Oracle для створення інтерфейсів користувача для веб-додатків на основі Java. Це офіційний стандарт ініціативи Java Community Process (JCP). Досить стабільний каркахс [30].

Це основа для інтерфейсів користувача на основі компонентів. JSF базується на шаблоні розробки програмного забезпечення MVC і має архітектуру, яка повністю визначає відмінності між логікою програми та представленням.

Grails — це динамічна структура, розроблена з використанням мови програмування Groovy JVM [31]. Це об'єктно-орієнтована мова для платформи Java, призначена для підвищення продуктивності розробників. Синтаксис сумісний з Java і компілюється в байт-код JVM (віртуальна машина Java). Grails використовує технології Java, включаючи контейнери Java EE, Spring, SiteMesh, Quartz і Hibernate.

Популярні фреймворки Java Script:

Angular.js — це популярна інтерфейсна платформа розробки з відкритим кодом, яка в основному використовується для розробки динамічних односторінкових веб-додатків (SPA) [32].

AngularJS передає весь вміст із сервера в браузер і завантажує всі веб-сторінки одночасно. Після завантаження вмісту клацання будь-якого посилання на сторінці не перезавантажує весь вміст сторінки; натомість оновлюється лише розділ на сторінці.

Найпомітніша відмінність між традиційним веб-сайтом і веб-сайтом на базі Angular полягає в тому, що Angular очікує, що браузер відобразить сторінку. Це не створює великого навантаження на сервер і, отже, призводить до швидшого завантаження сторінок.

React — це бібліотека для розробки програм інтерфейсу користувача. Випущений у 2013 році, він став найшвидше зростаючим фреймворком JS у світі [33].

React.js є першим вибором, коли йдеться про надання користувачам високопродуктивних корпоративних програм. Інтерфейси користувача, такі як Instagram і Facebook, забезпечують це.

React більш гнучкий, ніж Angular, тому що розробникам доведеться використовувати автономні бібліотеки з відносно малим часом відповіді. React

чудовий, коли мова йде про невеликі функції без стану, які приймають вхідні та повертаючі елементи як вихідні.

Vue.js – це легкий прогресивний фреймворк JS, який запозичив багато концепцій з ReactJS і AngularJS [34].

Він має стиль шаблонів, схожий на Angular, із властивостями компонентів, схожими на ReactJS. Vue надає просте та швидке рішення для розробки програм, інтерфейсу користувача та інтерактивного веб-інтерфейсу. Він може запускати просунуті односторінкові веб-додатки.

Найважливішою перевагою вибору Vue над React є те, що за допомогою Vue залежності компонентів автоматично відстежуються під час візуалізації. Таким чином, система знає, який компонент повторно відобразити, коли стан змінюється.

Це запобігає додатковій роботі з оптимізації та дозволяє розробникам більше зосередитися на створенні програм.

Популярні Python фремворки:

Bottle – мікрофреймворк bottle створює вихідний файл для кожної розробленої з його допомогою програми [35]. Це один із найкращих веб-фреймворків Python. Мікрофреймворки Python спочатку були розроблені для створення API. Bottle не має залежностей, необхідних для створення невеликих веб-додатків, крім стандартних бібліотек Python. Однією з найважливіших переваг використання Bottle є те, що воно дозволяє розробникам бути ближче до апаратного забезпечення. Окрім створення простих програм для особистого користування, Bottle чудово підходить для навчання організації веб-фреймворків і створення прототипів.

Основні моменти:

- Підтримування адаптера для сторонніх механізмів шаблонів і серверів WSGI/HTTP
- Дає маршрути надсилання запитів із підтримкою URL-параметрів
- Вбудований HTTP сервер

- Надає простий доступ до файлів cookie, даних, завантажених файлів та інших пов’язаних із HTTP метаданих
- Підтримка плагінів для різних баз даних

CherryPy — популярна об’єктно-орієнтована структура Python з відкритим вихідним кодом із мінімалістичним підходом [36].

Будь-яка веб-програма на основі CherryPy є самостійною програмою Python із власним вбудованим багатопоточним веб-сервером, який працює на будь-якій операційній системі, яка підтримує Python. Таку програму можна розгорнути будь-де, де може працювати звичайна програма Python.

Програми, розроблені за допомогою CherryPy, можна запускати без сервера Apache. Мікрофреймворки дозволяють розробникам використовувати будь-які типи технологій для доступу до даних, створення шаблонів тощо.

Основні моменти:

• Багато готових інструментів для автентифікації, кешування, шифрування, сеансів, статичного вмісту тощо

• Забезпечує зручність одночасного запуску кількох HTTP-серверів

• Гнучка вбудована система плагінів

• HTTP/1.1-сумісний веб-сервер WSGI з пулом потоків

Потужна система конфігурації

• Доступно для Android

• Вбудована підтримка покриття, аналізу та тестування

Фреймворк повного стеку Django є одним із найпопулярніших фреймворків веб-розробки для розробки програм Python [37]. Фактично, це стало одним із найкращих фреймворків веб-розробки. Фреймворк Django дотримується принципу DRY (Don't Repeat Yourself).

На відміну від інших фреймворків, безкоштовні фреймворки Python із відкритим кодом містять велику кількість вбудованих функцій, а не надають їх як окремі бібліотеки. Django використовує свою ORM для відображення об’єктів у таблиці бази даних.

Це дозволяє коду працювати в різних базах даних, а також полегшує міграцію з однієї бази даних в іншу. Хоча Django має вбудовану підтримку баз даних MySQL, PostgreSQL, SQLite та Oracle, він може підтримувати інші бази даних за допомогою сторонніх драйверів.

Основні моменти:

• Великі готові до використання бібліотеки
• Підтримка автентифікації
• Міграція схеми бази даних
• Object Relational Mapper (ORM)
• Підтримка веб-сервера
• механізм шаблонів
• URL-маршрутизація

Flask — ще один популярний фреймворк Python. Flask дозволяє розробникам створювати міцну основу веб-додатків, на основі яких вони можуть використовувати будь-які необхідні розширення. Мікрофреймворк сумісний із Google App Engine [38].

Основні моменти:

- На основі Unicode
- Підтримка модульного тестування
- Відповідність WSGI 1.0
- Вбудований швидкий налагоджувач
- Обробка запитів HTTP
- Вбудований сервер розробки
- Шаблони Jinja2
- Надсилання запити RESTful
- Підтримує підключення до будь-якої ORM
- Підтримка безпечних файлів cookie для встановлення клієнтських сеансів

Twisted — це платформа розробки онлайн-додатків, розроблена для повного розділення логічного протоколу та фізичного транспортного рівня [39].

Зазвичай націлена на семантику з’єднання на основі потоку, наприклад HTTP або POP3. Зв'язок існує між логічними протоколами на транспортному рівні до того, як інформація буде передана до екземплярів логічного протоколу. Структура спрямована на уніфікацію. Усі функції доступні через усі протоколи.

Tornado — це фреймворк, який не блокується під час запитів і легко розширюється. Tornado ідеально підходить для тривалих запитів, веб-сокетів і програм, які вимагають постійних з’єднань з великою кількістю користувачів, оскільки він здатний масштабуватися до сотень тисяч відкритих з’єднань [40].

Pyramid — це фреймворк з відкритим кодом для Python. Однією з особливостей цього фреймворку є простота його використання програмістами під час створення веб-додатків [41].

3.2 Засоби розробки

Інформаційна система пошуку за неповним описом створюється з використанням кращих методів розробки сучасних засобів і програмного забезпечення.

Програмне забезпечення, яке буде використане при розробці цієї інформаційної системи:

- Мова Python, версія 3.9 і вище;
- Фреймворк Django;
- Flake8 – інструмент малювання;
- CSS фреймворк Bootstrap 5;
- Docker – засіб автоматичної компіляції проекту;
- Середовище розробки Visual Studio Code;
- Бібліотеки: django-allauth, pytest.

Далі опишимо детальніше програмне забезпечення та засоби розробки.

Python є динамічно типізованою мовою програмування [42]. Дана мова програмування приваблює розробників швидкістю розробки. Він має ефективні структури даних і є досить простим, але ефективним підходом до об’єктно-орієнтованого програмування. Python використовує інтерпретатор. Він

портований майже на всі типи існуючих платформ, і це є перевагою, оскільки код, написаний на ньому, дуже переносимий. Ця мова є основною мовою для написання інформаційних систем пошуку автомобільних запчастин.

Docker — це програмне забезпечення з відкритим кодом, яке дозволяє керувати віртуальними контейнерами [43]. Вирішує багато проблем, пов'язаних із розгортанням програмного забезпечення, створенням контейнерів і тестуванням компонентів. Головне завдання — ізолювати програмне забезпечення від інших процесів у системі. Docker є швидким і гнучким під час розгортання програмного забезпечення на серверах.

Bootstrap 5 є безкоштовним набором інструментів щоб створювати веб-додатки, що включає шаблони CSS і HTML для кнопок і форм [44]. Типографіка та інші компоненти інтерфейсу. Bootstrap 5 спрощує і прискорює розробку веб-додатків. Даний набір інструментів є клієнтським фреймворком, тобто інтерфейс користувача.

Visual Code Studio — комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від Microsoft. Вихідний текст Community Edition поширюється за ліцензією Apache 2.0. Двійкові випуски підготовлені для Linux, Mac OS X і Windows. У ньому є багато програм, які можна використовувати для зручного тестування, створення графіків і порівняльних показників [45].

PyTest — це платформа для тестування програм на Python. PyTest відіграв важливу роль у створенні TDD (Test Driven Development) як способу написання програм. є одним із найпопулярніших фреймворків тестування. У даній роботі він використовується для порівняння результатів програм на тестовому корпусі [46].

Opencsv — це проста у використанні бібліотека для аналізу файлів CSV (розділених комами) для Java. Він використовується для обробки даних - зчитування схеми та екземплярів тестування user-agent [47].

3.3 Архітектура програмного забезпечення

3.3.1 Діаграма класів

У ході розробки створено структуру класів інформаційної системи відповідності шаблонів пошуку автозапчастин.

Основна програма розділена на 3 пакети: Trie, Parser і Capabilities.

Пакет Trie відповідає за створення та представлення стисненої інформації про всі шаблони введення у вигляді дерева, іншими словами, це автономні кроки алгоритму.

Пакет Parser відповідає за основну роботу програми, що відповідає онлайн-етапам алгоритму.

Пакет Capabilities містить класи та інтерфейси для представлення інформації про можливі типи результатів.

3.3.2 Діаграма компонентів

Систему розділена на п’ять основних компонентів, як показано на рис. 3.1:

- Pattern Trie - компонент, що реалізує та підтримує структуру даних префіксного дерева, яка будується на етапі попередньої обробки шаблону пошуку та використовується пізніше при обробці вхідних рядкових запитів;
- User Agent DataBase – компонент, відповідальний за представлення та інтерфейс можливостей і функцій браузера, який також зберігає зв’язок між шаблонами пошуку та їхніми відповідними браузерами;
- Automata – компоненти, які реалізують і підтримують кінцеві автомати, побудовані за алгоритмом Ахо-Корасіка, побудовані на етапі попередньої обробки пошукового шаблону і потім використовуються при обробці вхідного рядка запиту;
- User-Agent processor – компонент, відповідальний за обробку вхідного рядка запиту, який працює з компонентами шаблону trie та automaton для пошуку збігів шаблону пошуку з вхідним рядком.

- Patterns Preprocessor – компонент, відповідальний за початкову обробку шаблонів пошуку, який будує префіксні дерева та автомати;

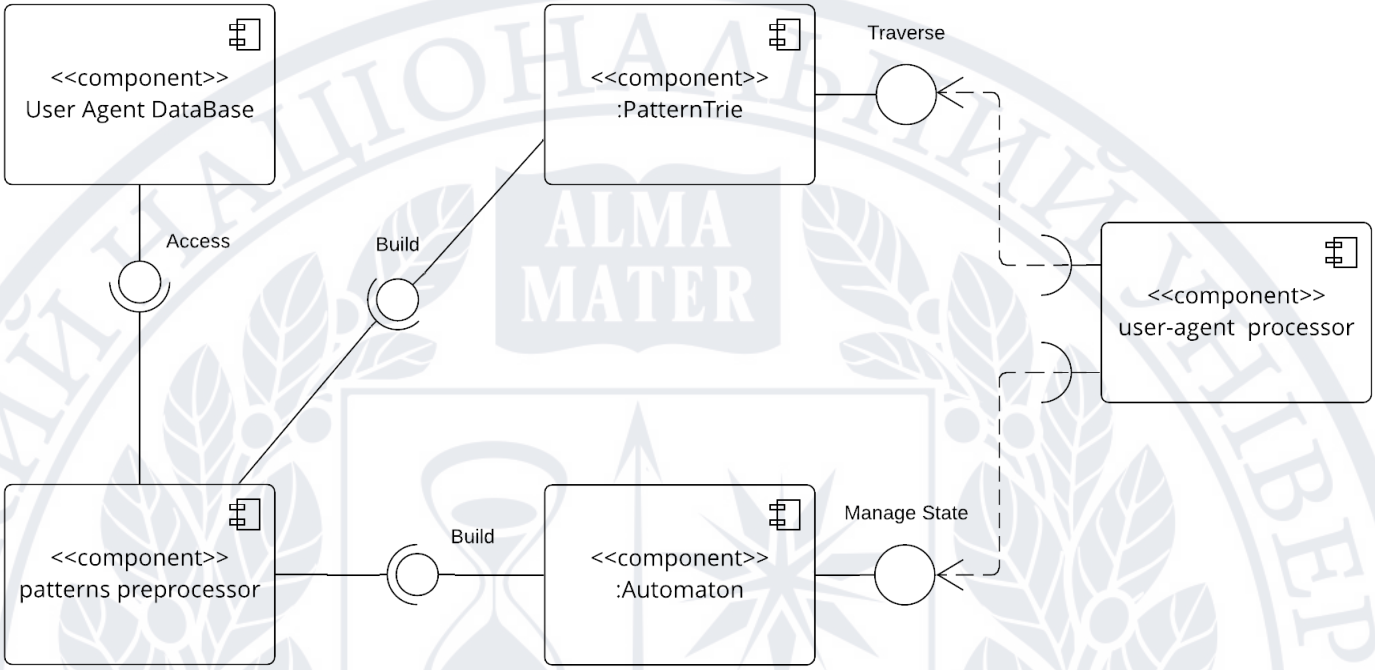


Рисунок 3.1 – Компоненти системи

3.4 Специфікація функцій

У таблиці 3.1 наведено функції класів інформаційної системи, які відповідають шаблонам пошуку для визначення можливостей браузера.

Таблиця 3.1 – Функції класів програмного забезпечення

Клас	Функція	Опис функції
Parser	parse(String): Capabilities	приймає на вхід рядок user-agent та вертає характеристики браузера, що йому відповідають

Продовження таблиці 3.1

	getPatternByString(String, PatternTrie): ?Capabilities	описує основні кроки онлайн частини алгоритму, приймає на вхід Trie та рядок user-agent
	getToken(PatternTrie, Int): String	використовуючи дерево та номер токена вертає його символьне представлення
	buildVertexInfo(String, Function<String, Integer>): List<VertexInfo>	за вхідним рядком вертає усі можливі VertexInfo екземпляри, відповідає кроку (6) алгоритму
	buildAllPossibleWordsGraph(List<VertexInfo>): List<List<Pair<Int, Int>>>	за результатом buildVertexInfo вертає DAWG, відповідає кроку (7) алгоритму
VertexInfo	begin: int	відповідає індексу першого символа у вхідному рядку, що відповідає деякому токену
	end: int	відповідає індексу останнього символа у вхідному рядку, що відповідає деякому токену
	key: int	номер отриманого токена
PatternTrie	getRoot(): PatternTrieNodeRoot	вертає корінь дерева, з якого можна почати спуск

Продовження таблиці 3.1

	getTokenNumber(String) : int	вертає номер токена за його символьним представленням
	getFrequency(int, int): int	вертаю частоту переходів від одного токена до іншого у дереві

3.5 Структурно-функціональне моделювання процесу.

IDEF — це графічна техніка моделювання процесу, яка використовується для застосування процесів та інженерних програм. Метод функціонального моделювання IDEF0 призначений для моделювання рішень, дій і діяльності організації або системи. IDEF0 дуже корисний для визначення обсягу аналізу, особливо для функціонального аналізу. Як механізм співпраці, IDEF0 використовує спрощені інструменти візуалізації для полегшення участі та спільного прийняття рішень експертами галузі.

IDEF0 є дуже простим підходом у багатьох відношеннях. Як і в інших методах, кожне поле представляє окремий процес, але IDEF0 відрізняється використанням і розташуванням стрілок. На додаток до звичайних входів і виходів, є ще два типи стрілок, що представляють «контроль» і «механізм».

Перелік основних функцій цієї інформаційної системи:

1. Зареєструватися на сайті.
2. Авторизація на сайті.
3. Налаштування особистого кабінету.
4. Написати повідомлення адміністратору.
5. Оформлення розділу Навички.
6. Перегляд облікових записів СТО або окремих виконавців.
7. Перегляд контактних даних.
8. Використовування вбудованого калькулятора, щоб розрахувати ціну на ремонт автомобіля.

9. Замовлення послуг або консультації.

10. Написати відгук.

11. Можливість спілкування з користувачами інформаційної системи (СТО/інші виконавці).

Елементи керування є формою введення, але використовуються для керування діяльністю в процесі. Іноді існує певна невизначеність щодо того, чи є елемент вхідним елементом чи елементом керування. Розглянемо інформаційну систему IDEF0 (рисунок 3.2).

Рисунок 3.2 – IDEF0 інформаційної системи

Метод IDEF1 поділяє елементи структури інформаційної індустрії, їх атрибути та зв’язки на кілька категорій. Основною концепцією методології IDEF1 є концепція сутності. Клас сутності — це група інформації, накопиченої та збереженої всередині підприємства, яка відповідає об’єкту або групі об’єктів у реальному світі.

Основними концептуальними властивостями сутностей в IDEF1 є:

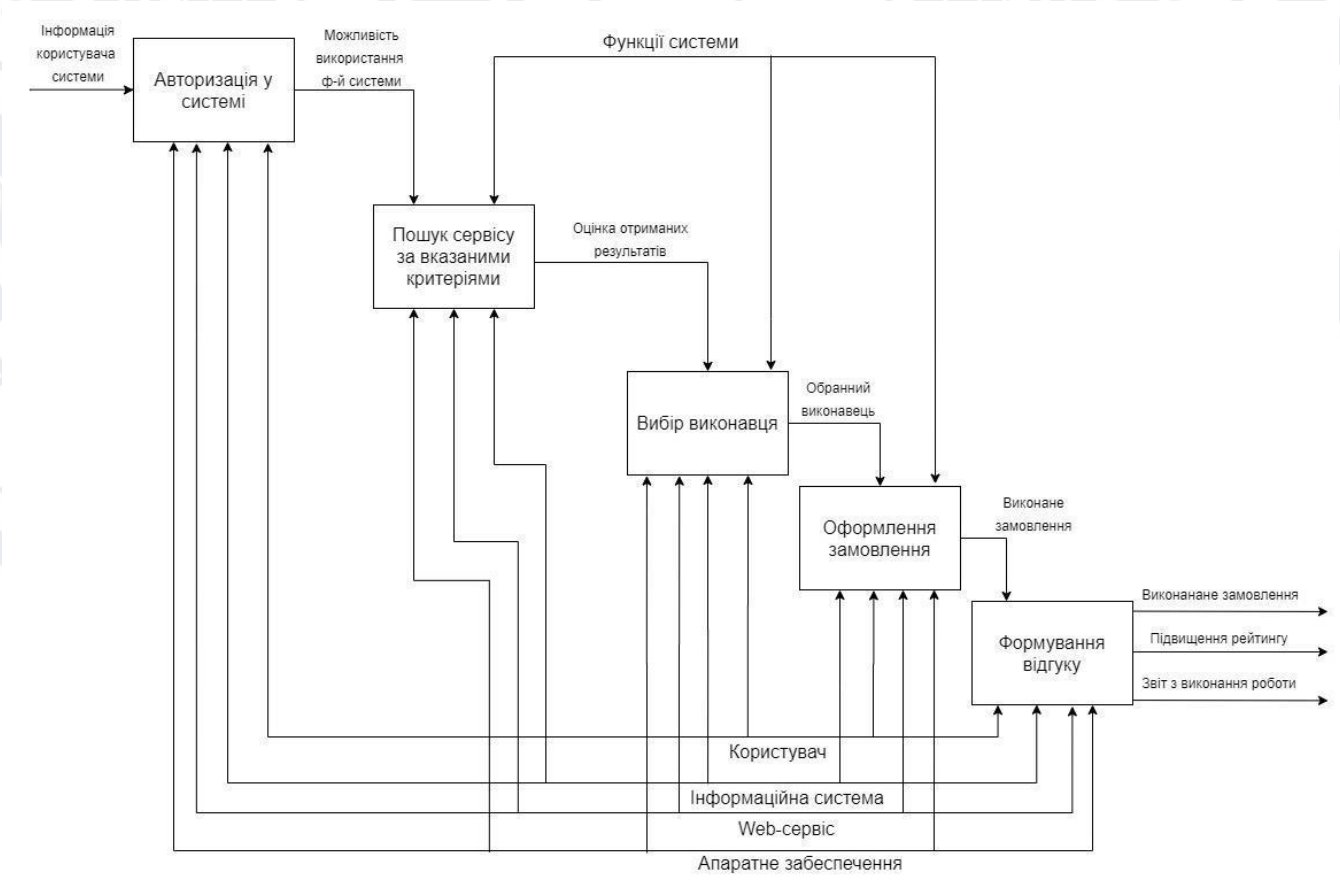
- Довговічність. Інформація, що стосується тієї чи іншої сутності, постійно накопичується.

- Унікальність. Будь-яка сутність може бути однозначно ідентифікована з іншої сутності.

Кожна сутність має власну назву та властивості. Властивості — це характеристики об’єктів у реальному світі, пов’язані з сутністю. Клас атрибутів — це набір пар, що складається з імені атрибута та його значення для певної сутності. Атрибути, за якими можна однозначно відрізнити одну сутність від іншої, називаються ключовими атрибутами.

Кожну сутність можна охарактеризувати кількома ключовими властивостями. Клас зв’язку в IDEF1 представляє набір зв’язків між сутностями. Кажуть, що зв’язок між двома незалежними сутностями існує, якщо клас атрибутів однієї сутності містить ключовий атрибут іншої сутності. Кожен із зазначених вище класів має власне умовне графічне відображення відповідно до підходу IDEF1. Розглянемо IDEF1 для інформаційних систем (рисунок 3.3).

Рисунок 3.3 – IDEF1 проекту



3.6 Діаграми використання

Мета використання діаграм — показати динамічні аспекти системи. Однак це визначення є надто загальним, щоб описати його мету, оскільки інші чотири діаграми (діаграма діяльності, діаграма послідовності, діаграма співпраці та діаграма стану) також служать тій же меті. Ми розглянемо деякі особливості використання, які відрізняють від інших чотирьох діаграм.

Діаграми варіантів використання використовуються для збору системних вимог, включаючи внутрішні та зовнішні впливи. Ці вимоги є перш за все вимогами до дизайну. Тому, аналізуючи систему, щоб зібрати її можливості, підготуйте варіанти використання та визначте учасників.

Діаграма використання моделюється для представлення зовнішнього вигляду після виконання початкового завдання.

Метою діаграми випадків використання може бути:

– Використання для збору системних вимог.

– Використання для отримання оболонки системи.

– Визначення зовнішніх та внутрішніх факторів, що впливають на систему.

Розглянемо детальніше акторів та варіанти використання в табл. 3.2 та табл. 3.3.

Таблиця 3.2 – Опис акторів

Назва	Опис
Виконавець	Юзер із можливістю виконання замовлення, створення портфоліо та спілкування з клієнтами.
Замовник	Юзер, який має змогу створити замовлення в інформаційній системі.
Адміністратор	Юзери з найбільшою можливістю використання сайту, тобто редагування та видалення даних.

Таблиця 3.3 – Опис варіантів використання

Назва	Опис
Авторизація	Цей модуль дозволяє авторизуватися на сайті
Реєстрація	Цей модуль дозволяє зареєструватись новим користувачам на сайті.
Редагування даних на сторінці	Кожен користувач може редагувати персональну інформацію на сайті.
Редагування інформації на сайті	Дозволяє змінити загальну інформацію.
Перегляд замовлень та користувачів	Цей модуль дозволяє переглядати зареєстрованих юзерів та замовлення на сайті.
Формування відгуку	Зворотній зв`язок після виконання замовлення.
Додавання послуг	Модуль для додавання послуг для особистого кабінету.
Створення замовлення	Дозволяє клієнтам створювати нове замовлення.

Після формування представлення щодо інформації та функціональності системи, було сформовано діаграму варіантів використання (рис 3.4).

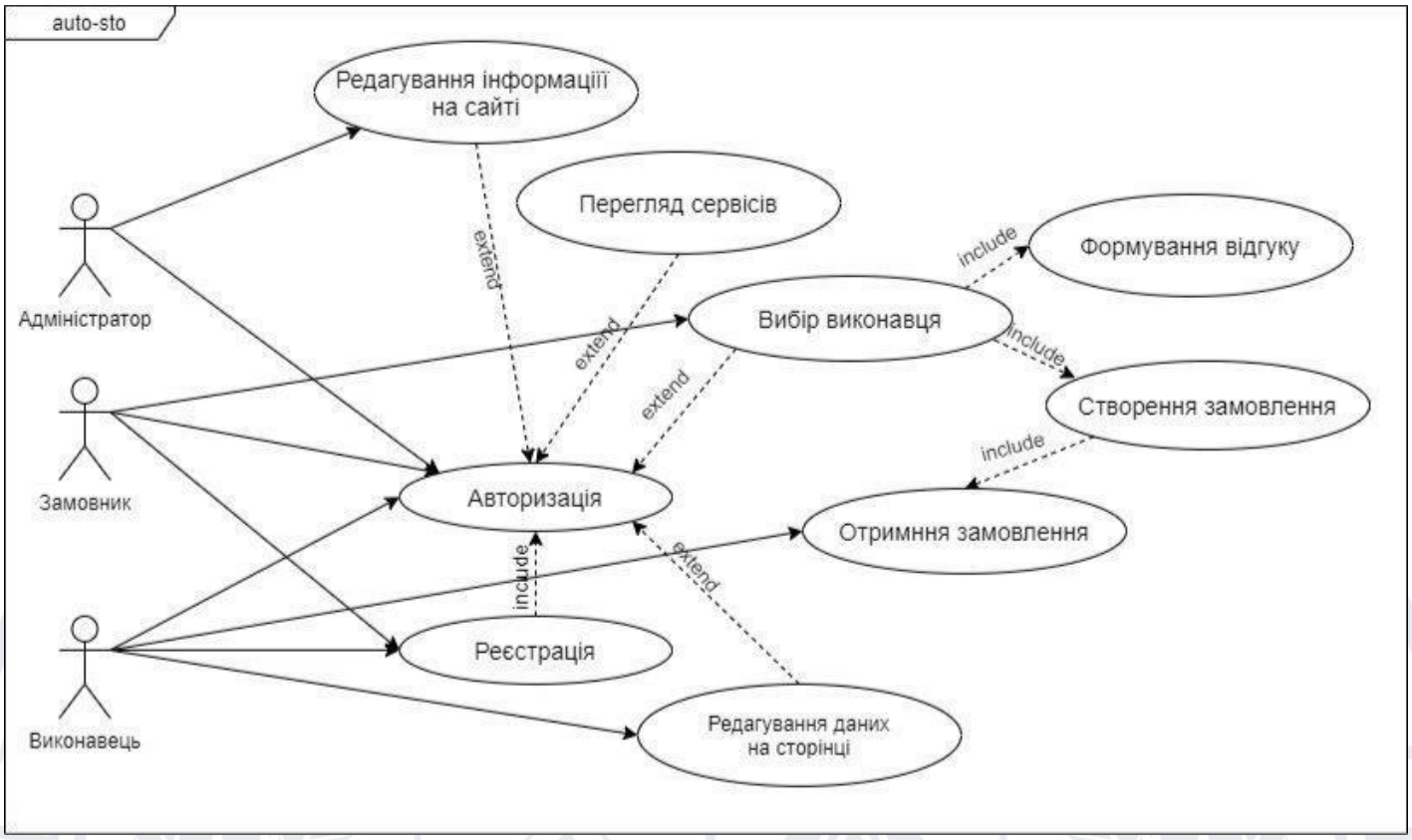


Рисунок 3.4 – Діаграма варіантів використання

3.7 Проектування бази даних

Основними користувачами системи є клієнти та фахівці. За допомогою цієї системи вони можуть отримувати різноманітну інформацію про виконавців і замовників, послуги тощо.

Опишемо цю залежність на ER-діаграмам (рис. 3.5 – рис. 3.7). Проаналізувавши суть використаної в моделі інформаційної системи, перейдемо до реалізації структури бази даних. Для цього в таблиці покажемо імена, атрибути, типи, їх призначення та обмеження у таб. 3.5.

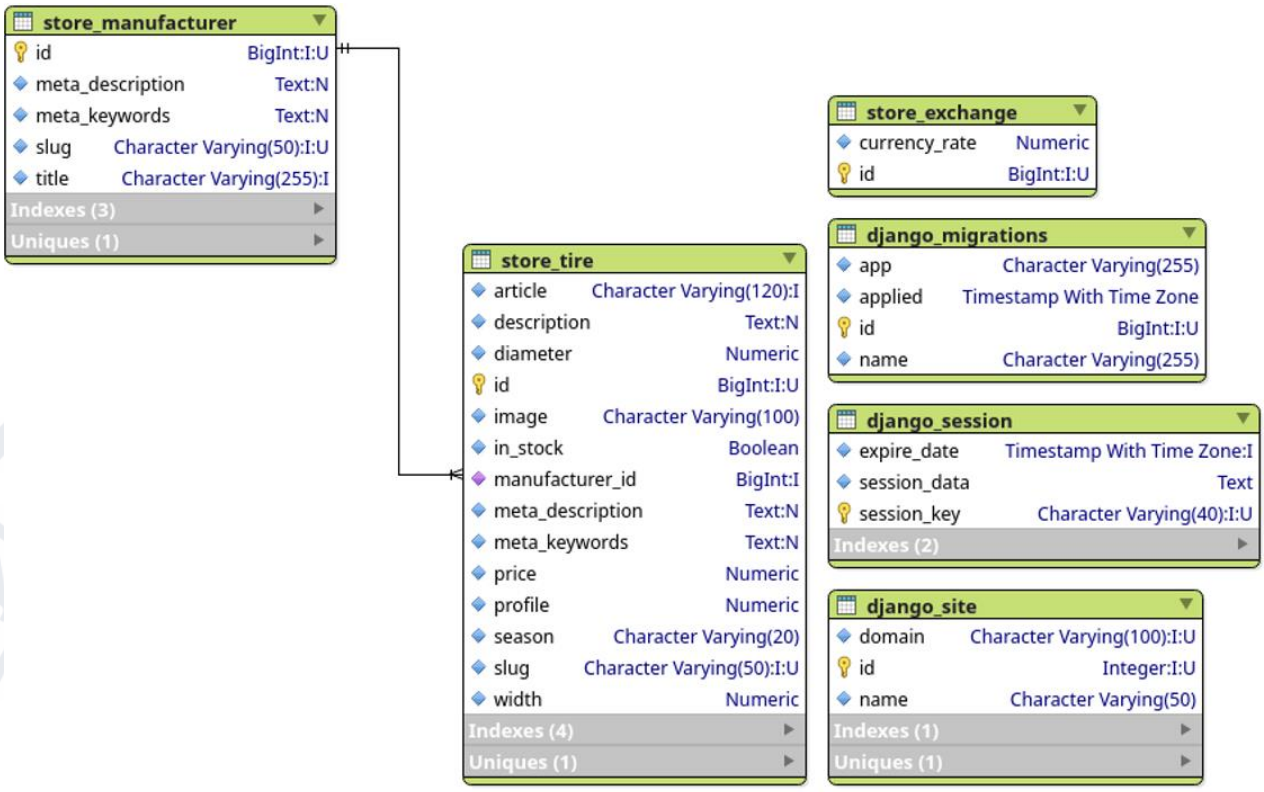


Рисунок 3.5 ER- діаграма

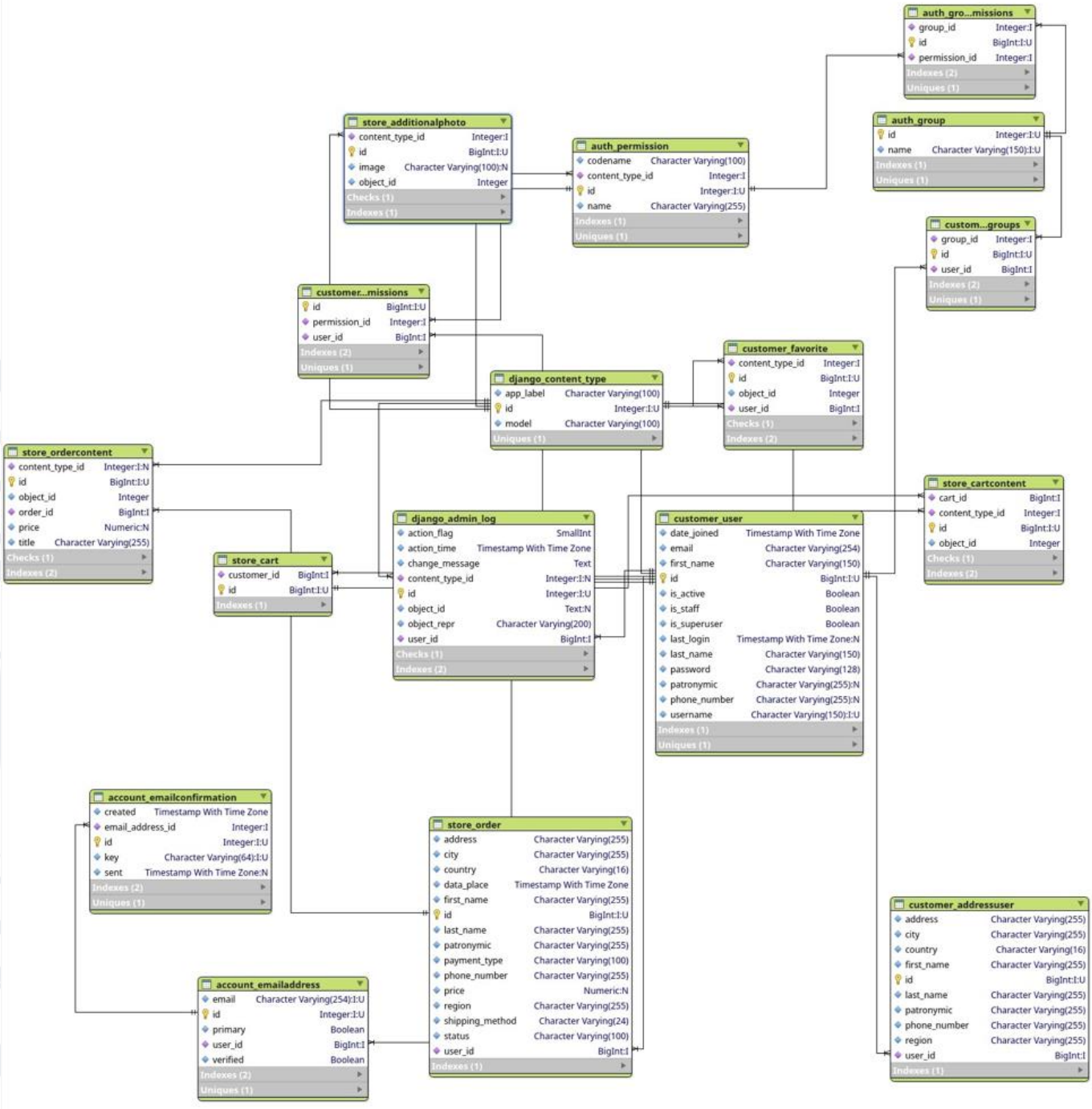


Рисунок 3.6 ER- діаграма

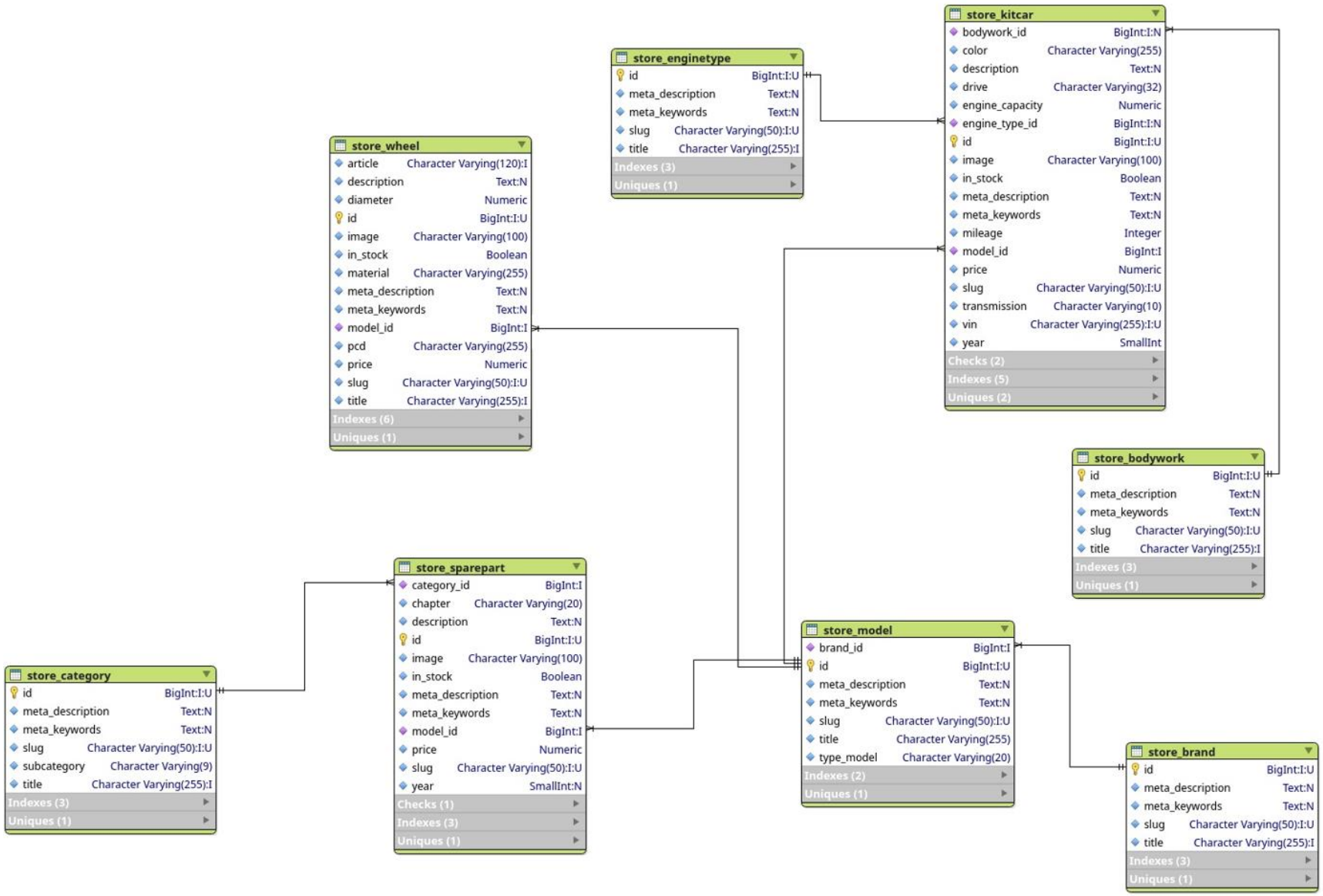


Рисунок 3.7 ER- діаграма

Таблиця 3.4 – Інформація за таблицями ER-діаграми

№	Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
1	account	account_id	Ідентифікатор аккаунта	INTEGER	PK	Не пустий
		email	Електронна пошта	VARCHAR(100)		Не пустий
		password	Пароль від аккаунта	VARCHAR(100)		Не пустий
		photo	Фото аккаунта	VARCHAR(100)		
		address	Адреса	VARCHAR(100)		
		register	Дата реєстрації	DATE		Не пустий
		account_status_id	Ідентифікатор статусу аккаунта	INTEGER	FK	Не пустий
		contacts_id	Ідентифікатор контактів	INTEGER	FK	Не пустий
2	account_status	account_status_id	Ідентифікатор статусу аккаунта	INTEGER	PK	Не пустий
		account_status_title	Назва статусу	VARCHAR(100)		Не пустий
3	person	peson_id	Ідентифікатор спеціаліста	INTEGER	PK	Не пустий
		name	Ім’я	VARCHAR(100)		Не пустий
		surname	Прізвище	VARCHAR(100)		Не пустий
		account_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
4	service	service_id	Ідентифікатор сервісу	INTEGER	PK	Не пустий
		title	Назва сервісу	VARCHAR(100)		Не пустий

Продовження таблиці 3.4

№	Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
		account_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
5	photo	id_photo	Ідентифікатор зображення	INTEGER	PK	Не пустий
		putch	Посилання на файл	VARCHAR(100)		Не пустий
		account_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
6	contacts	contacts_id	Ідентифікатор контакту	INTEGER	PK	Не пустий
		phone_id	Ідентифікатор телефону	INTEGER	FK	Не пустий
		socials_links_id	Ідентифікатор посилання соціальної мережі	INTEGER	FK	Не пустий
7	phones	phone_id	Ідентифікатор телефону	INTEGER	PK	Не пустий
		number	Номер телефону	VARCHAR(100)		Не пустий
8	socials_links	socials_links_id	Ідентифікатор посилання соціальної мережі	INTEGER	PK	Не пустий
		link	Посилання на соціальну мережу	VARCHAR(100)		Не пустий
		socials_id	Ідентифікатор соціальної мережі	INTEGER	FK	Не пустий
9	socials	socials_id	Ідентифікатор соціальної мережі	INTEGER	PK	Не пустий
		title	Назва соціальної мережі	VARCHAR(100)		Не пустий
		logo	Логотип соціальної мережі	VARCHAR(100)		Не пустий

Продовження таблиці 3.4

№	Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
10	account_services	services_id	Ідентифікатор спеціалізації	INTEGER	FK	Не пустий
		account_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
11	services	services_id	Ідентифікатор спеціалізації	INTEGER	PK	Не пустий
		services_title	Назва спеціалізації	VARCHAR(100)		Не пустий
12	sub_s ervices	sub_services_id	Ідентифікатор послуги	INTEGER	PK	Не пустий
		sub_services_title	Назва послуги	VARCHAR(100)		Не пустий
		services_id	Ідентифікатор спеціалізації	INTEGER	FK	Не пустий
		price	Ціна спеціалізації	INTEGER		Не пустий
13	account_car	account_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
		car_specialization_id	Ідентифікатор марки авто	INTEGER	FK	Не пустий
14	car_specialization	car_specialization_id	Ідентифікатор марки авто	INTEGER	PK	Не пустий
		mark	Марка авто	VARCHAR(100)		Не пустий
15	reviews	reviews_id	Ідентифікатор відгуку	INTEGER	PK	Не пустий
		text	Текст відгуку	TEXT		Не пустий
		account_id	Ідентифікатор аккаунта	INTEGER	FK	Не пустий
		comment_account_id	Ідентифікатор аккаунта що залишив відгук	INTEGER	FK	Не пустий

Продовження таблиці 3.4

№	Таблиця	Поле	Зміст	Тип	Ключі	Обмеження
16	info	info_key	Ключ запису	VARCHAR(100)	PK	Не пустий
		value	Значення	TEXT		Не пустий

3.8 Діаграми діяльності

Діаграми діяльності — це техніка, яка дозволяє описати логіку програм, бізнес-процесів і робочих процесів. Багато в чому вони нагадують блок-схеми, але фундаментальна відмінність між діаграмами діяльності та нотацією блок-схем полягає в тому, що перша підтримує паралельні процеси.

Діаграми діяльності дозволяють кожному, хто виконує процес, вибрати курс дій. Інакше кажучи, діаграма лише надає правила обов'язкової послідовності дій, які я повинен виконати. Це є важливим для моделювання бізнес-процесів, так як ці процеси часто реалізуються паралельно. Такі діаграми також є корисними при розробці паралельних алгоритмів, де незалежні потоки мають можливість виконувати роботу паралельно.

Діаграма діяльності кожного модуля інформаційної системи фріланс-біржі наведена на рис. 3.8-3.12.

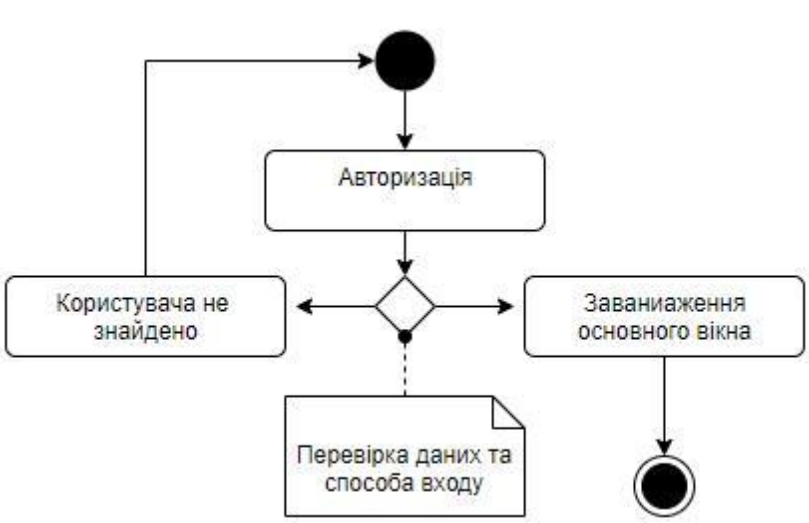


Рисунок 3.8 – Діаграма діяльності модулю авторизації

Рисунок 3.9 – Діаграма діяльності модулю реєстрації

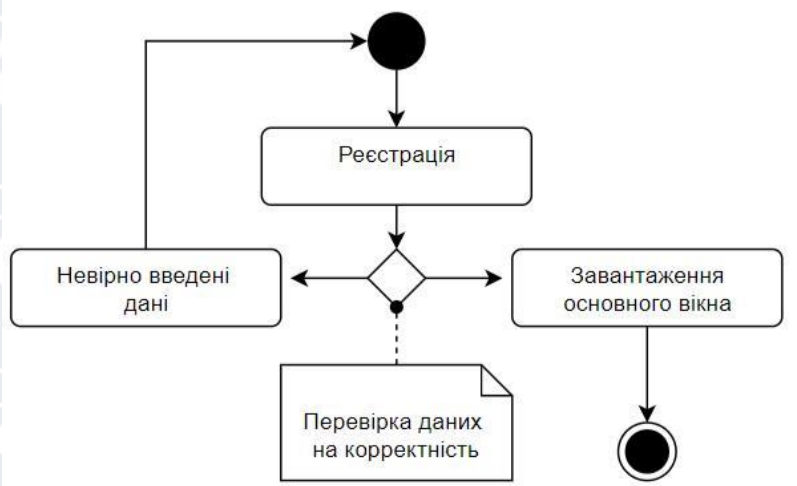


Рисунок 3.10 – Діаграма діяльності модулю редагування інформації на сайті

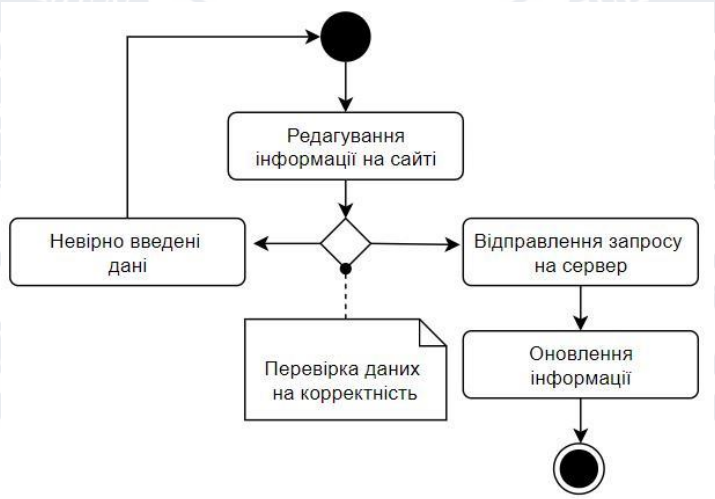


Рисунок 3.11 – Діаграма діяльності модулю перегляду інформації

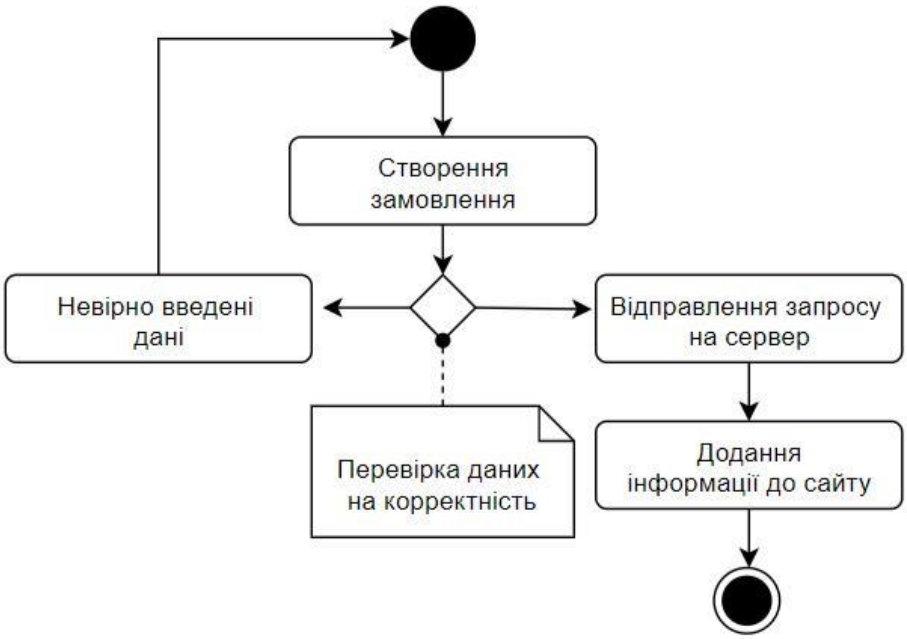


Рисунок 3.12 – Діаграма діяльності модулю створення замовлення

Висновки за розділом 3

Розглянуто технічні аспекти програмного забезпечення інформаційних систем. Описано технологію, обрану для написання системи, і вимоги до технічної підтримки системи.

Архітектура системи також описана і представлена у вигляді діаграм класів і діаграм компонентів. Одночасно надається функціональна специфікація та її детальний опис.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

4.1 Програмна реалізація

Програмна реалізація програми виконана за допомогою фреймворку Django, так як він має всі необхідні інструменти для розробки додатку. Розглянемо докладніше структуру інформаційної системи (табл. 4.1).

Таблиця 4.1 – Таблиця рівнів доступу

Пакет	Короткий опис
«app»	Містить код ядра додатку.
«bootstrap»	Містить файли, які завантажують фреймворк і налаштовують автозагрузку.
«config»	Містить всі конфігураційні файли додатку
«database»	Містить міграції і класи для наповнення початковими даними БД.
«public»	Містить файл index.php, який є вхідною точкою для всіх запитів, що надходять в додаток. Також ця папка містить ресурси, такі як зображення, JavaScript, CSS.
«resources»	Містить початковий код, а також сирі, некомпільовані ресурси, такі як LESS, SASS, JavaScript.
«routes»	Містить всі визначення маршрутів додатку.
«storage»	Містить скомпільовані Blade-шаблони, файл-сесії, кешу файлів і інші файли, створені фреймворком.
«tests»	Містить автотести.

Інформаційна система, яка надає послуги з технічного обслуговування автомобіля, має певну кількість типів користувачів, а саме адміністратори та «звичайні користувачі».

Розглянемо розподіл функціоналу більш детально в табл. 4.2.

Таблиця 4.2 – Функціонал за групами користувачів

Користувач	Можливості користувача
Адміністратор	Перегляд загальної інформації.
	Підтримка сайту.
	Блокування користувачів.
Звичайний користувач	Створення власної інформаційної сторінки;
	Редагування даних;
	Листування з користувачами системи;
	Використня вбудованоо калькулятру для обрахування ціни ремонту автомобіля;
	Оформлення замовлення послуги або консультації;
	Залишити відгук виконавцю за виконану роботу;
	Написання повідомлення адміністратору.

Для детального ознайомлення з програмою докладно опишемо кожну сторінку для подальшого прототипування.

4.1.1 Головна сторінка

Сторінка з базовою довідкою та функціями інформаційної системи надання послуг з ремонту автомобілів. Ця сторінка є першою сторінкою, на яку потрапляють користувачі.

Автолюбителів з кожним роком стає все більше, як і попит на ремонт, тому відкриваються нові СТО. Користувачі будуть проінформовані про останні новини та ознайомлені з переліком ключових переваг сервісу.

4.1.2. Реєстрація

Сторінка реєстрації нового користувача. Для створення облікового запису користувачі повинні ввести певний список даних. При створенні облікового

запису для інтернет-магазину для реєстрації необхідно заповнити наступні поля:

- E-mail
- Пароль

Якщо обліковий запис належить користувачеві, список полів змінюється та розширюється.

Перелік даних необхідних для реєстрації:

- Ім’я;
- Прізвище;
- E-mail;
- Пароль;
- Дата народження;
- Стать;
- Вид роботи.

4.1.3. Авторизація

Сторінка авторизації. Незалежно від типу сторінки, користувач повинен ввести такі дані:

- E-mail
- Пароль

4.1.4. Особистий кабінет

Сторінка інформації про користувача. На цій сторінці можна переглянути основні розділи, а саме контактну інформацію, досвід роботи, напрямок роботи для спеціалістів або СТО та ціни на перелік послуг.

4.1.5. Налаштування акаунту

Налаштування облікового запису. Сторінки для редагування основної інформації користувача та його додаткові налаштування.

4.1.7 Онлайн-калькулятор

Онлайн калькулятор - це додаткова функція для розрахунку трудомісткості авторемонтних робіт. На сторінці можна ввести виконавця роботи, а потім перелік необхідних робіт. Ціна буде розрахована за ціною, яку вкаже сам експерт на власній сторінці, залежно від типу та складності завдання.

4.1.8 Листування

За допомогою листів можна вирішити багато важливих завдань. Ця функція допоможе користувачам розвивати партнерські стосунки, інформувати про результати та завдання, ділитися новинами тощо.

4.2 Використання програмного додатку

Розглянемо головну сторінку інформаційної системи підбору авто запчастин 4.13.

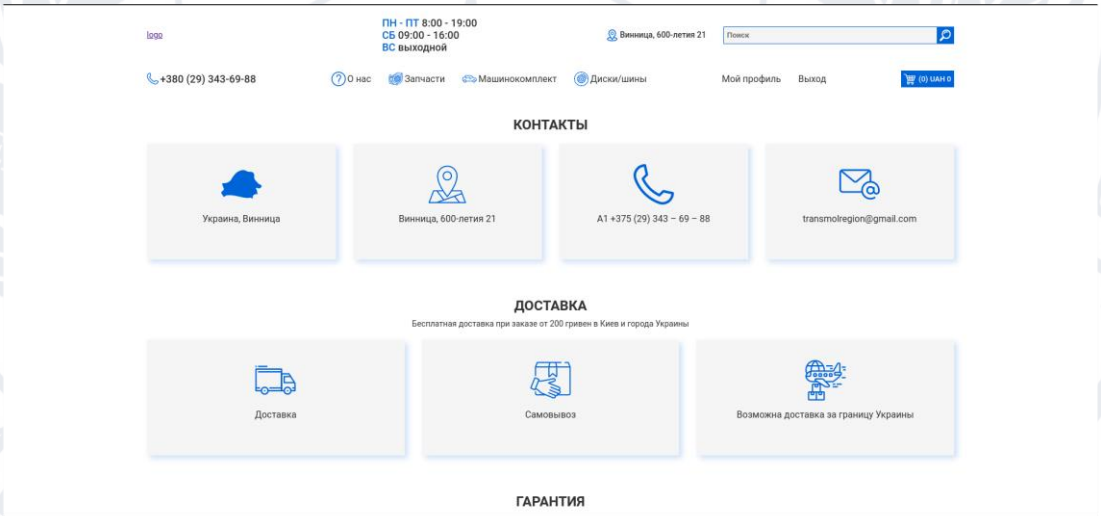


Рисунок 4.1 – Головна сторінка сервісу

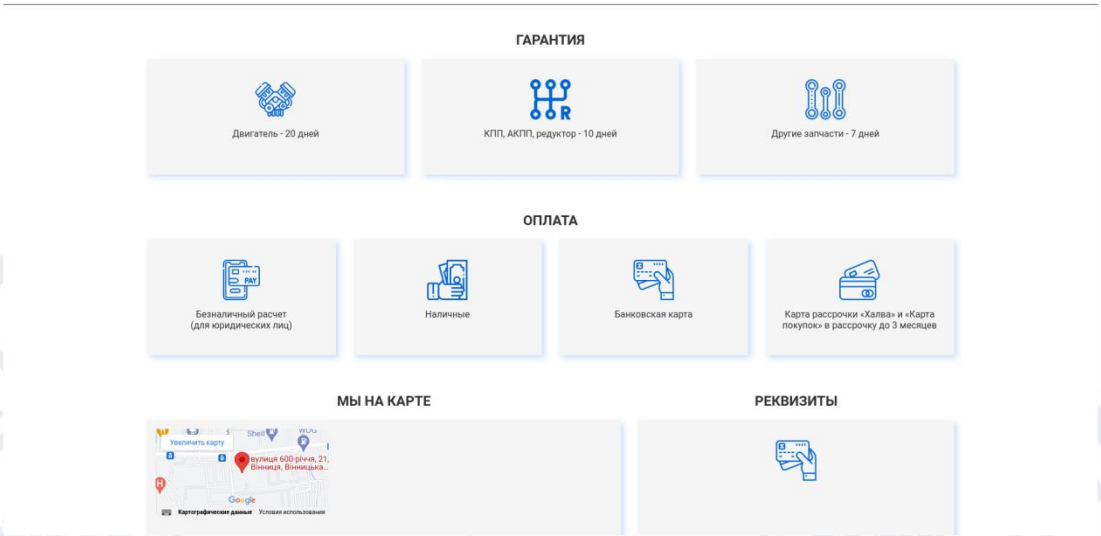


Рисунок 4.2 – Загальна інформація про інформаційну систему

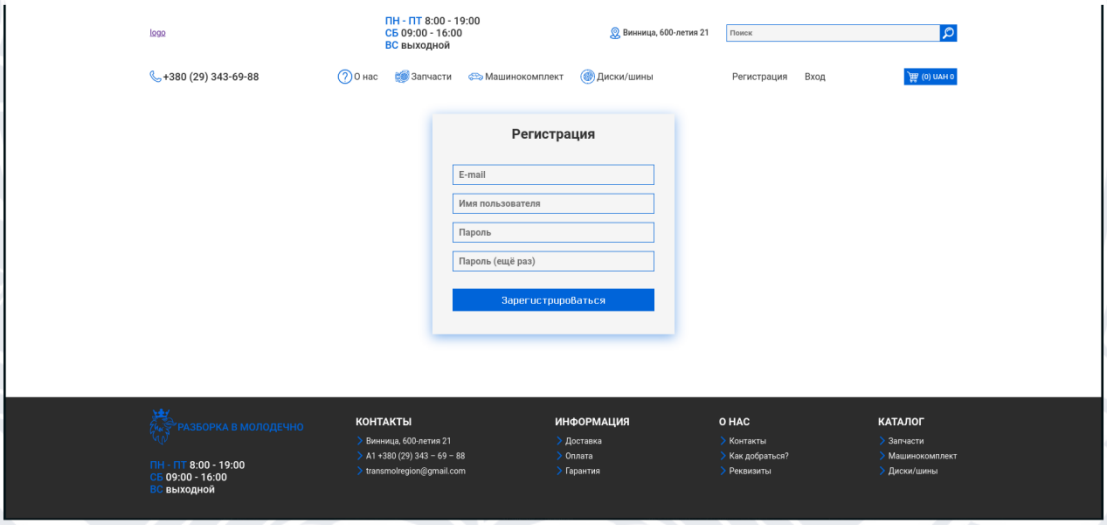


Рисунок 4.3 – Реєстрація як сервісний центр

4.3 Керівництво користувача

Будь-яка сумісна система, за допомогою якої сервер wsgi автоматично збирає елементи, може бути використана для запуску інформаційної системи та її подальшого використання в проектах Python. Оскільки інформаційна система поширюється як набір файлів, локальна інсталяція не потрібна.

Команда запуску сервера проекту: python manage.py runserver.

Команда запуску докера: docker-compose up.

4.4 Тестування

Тестування – це перевірка на сайті з метою виявлення відхилень, тобто процес виявлення помилок (багів). Перевірки можуть проводитися спільно підрядником і замовником. Оптимальним варіантом є – спеціаліст-тестувальник.

Час роботи системи особливо важливий, оскільки кожен клієнт, який не може створити замовлення, є грошовою втратою для компанії, тому в цій частині дизайну веб-сайт перевіряється за всіма потрібними стандартами.

Тестування – це виснажливий процес. Завдяки цьому етапу ми готові до того, що на момент запуску все буде працювати як потрібно, і якщо проігнорувати цей етап, комерційна кампанія може повністю провалитися.

4.4.1 План тестування

План тестування — це документ, який описує всю роботу, яку команда тестувальників виконуватиме в проекті. Він може містити вказівки щодо процесу тестування, такі як методологія, тестові завдання, екологічні вимоги, вимоги до ресурсів, графік і обмеження.

Для тесту системи створено план тесту, у якому визначено вид тестування, призначення виду тестування та характеристика процесу (табл. 4.3).

Таблиця 4.3 – План тестування

Тип тестування	Мета	Опис процесу
Тестування верстки	Перевірка коректного відображення елементів дизайну.	Перевірка коректності верстки на всіх сторінках сайту.
Функціональне тестування	Перевірка повного функціоналу системи з метою визначення помилок, що можуть заважати користувачеві або блокувати його.	Перевірка за раніше визначеним тест-планом, прохід по кожному пункту
Тестування кросбраузерності	Перевірка коректної роботи та дизайну проекту в різних браузерах.	Тестування верстки та функціоналу в браузерах: Safari, Chrome, Firefox, Opera.
Регресійне тестування	Набір тестів, спрямованих на виявлення дефектів у вже протестованих ділянках додатка.	Перевірка на виправлення дефектів, прохід по вразливим місцям.

4.4.2 Тестування верстки інтерфейсу

Виконаємо процес тестування інтерфейсу користувача, щоб забезпечити відповідність специфікації. Дизайн кожного елемента важливий, оскільки він не тільки приваблює користувачів, а й виконує в деяких випадках має особливу функцію. Інтерфейс користувача – це точка входу, через яку користувач може «спілкуватися» з системою. Таблиця 4.4 показує контрольний список для тестування макета інтерфейсу користувача.

Таблиця 4.4 – Результати тестування

	Chrome	Safari	Firefox	Opera
Коректне зображення хедера	Passed	Passed	Passed	Passed
Наявність логотипу	Passed	Passed	Passed	Passed
Коректне зображення футера	Passed	Passed	Passed	Passed
Наявність іконки кошика	Passed	Passed	Passed	Passed
Правильний шрифт	Passed	Passed	Passed	Passed
Правильність стилізації форм	Passed	Passed	Passed	Passed
Блоки не вилазять за свою ширину	Passed	Passed	Passed	Passed
Анімації	Passed	Passed	Passed	Passed
Коректне зображення кнопок	Passed	Passed	Passed	Passed
Всі компоненти відображаються коректно	Passed	Passed	Passed	Passed
Правильна стилізація посилань	Passed	Passed	Passed	Passed

4.3 Керівництво користувача

Будь-яка сумісна система, за допомогою якої сервер wsgi автоматично збирає елементи, може бути використана для запуску інформаційної системи та її подальшого використання в проектах Python. Оскільки інформаційна система поширюється як набір файлів, локальна інсталяція не потрібна.

Лістинг 3.1 – Приклад запуску сервера проекту.

```
python manage.py runserver
```

Лістинг 3.2 – Приклад запуску докера.

```
docker-compose up
```

4.4.4 Функціональне тестування

Функціональне тестування базується на функціях і особливостях, так само і на співдіянні з іншими системами, та може бути проведеним на усіх рівнях тестування: модуль чи компонент, інтеграція, система та прийняття.

Метою функціонального тестування є визначення невідповідностей між фактичною та гаданою поведінкою реалізованої функціональності згідно з специфікаціює і вимогами. Функціональне тестування має обхвачувати весь реалізований функціонал з врахуванням найбільш ймовірних видів помилок. Тестові сценарії в поєднанні з індивідуальними тестами спрямовані на перевірку якості вирішення функціональних завдань.

Таблиця 4.5 – Результати тестування

	Chrome	Safari	Firefox	Opera
Список продуктів береться з API	Passed	Passed	Passed	Passed
Компоненти рендеряться правильно	Passed	Passed	Passed	Passed
Товари потрапляють у кошик	Passed	Passed	Passed	Passed
Товари сортуються по категоріях	Passed	Passed	Passed	Passed
Правильність зображення полів товару	Passed	Passed	Passed	Passed
Ціна та назва відповідає товару на API	Passed	Passed	Passed	Passed
У кошик потрапляє саме той товар який обрав користувач	Passed	Passed	Passed	Passed
Модальне вікно замовлення з`являється	Passed	Passed	Passed	Passed
Пустий кошик має інший вигляд	Passed	Passed	Passed	Passed
Видалення товару з кошику	Passed	Passed	Passed	Passed
Збільшення кількості товарів	Passed	Passed	Passed	Passed
Зменшення кількості товарів	Passed	Passed	Passed	Passed

Таким чином, у результаті тестування виявляються та виправляються помилки та невідповідності між фактичною поведінкою системи та очікуваною поведінкою згідно з специфікацією та вимогами. Система працює нормально.

Висновки за розділом 4

Розроблено базу даних, тестування алгоритму для пошуку за неповним описом. Розроблена архітектура додатку та взаємодія модулів програми. Виконана та продемонстрована практична реалізація веб-додатку для пошуку автозапчастин за неповним описом.

ВИСНОВКИ

Отже проведено дослідження в області методів обробки та аналізу текстів. Тобто було викладено методи та алгоритми зіставлення пошукових шаблонів на основі текстових неповних описів для вирішення пошукових задач. Проаналізовано дослідження та розробки в цій галузі та згруповано результати аналізу. Крім того, для кожного розглянутого алгоритму показані недоліки та переваги порівняно з іншими. На основі аналізу запропоновано ряд завдань для досягнення поставлених цілей.

На основі аналітичних досліджень розроблено власний алгоритм для вирішення задачі зіставлення шаблонів пошуку на основі неповних описів. У процесі розробки були розглянуті сильні та слабкі сторони досліджуваних алгоритмів. У результаті розроблений алгоритм отримує найвідоміші оцінки робочого часу знизу.

У ході дослідження представлені ідеї, які дозволяють вдосконалити розроблену інформаційну систему та визначають подальший розвиток дослідження та системи в цілому.

В ході розробки алгоритму, було висунуто ідею використання алгоритму Левенштейна у парі для обробки вхідних запитів. Ці структури даних були модифіковані та з’єднані між собою для досягнення поставленої мети, завдяки чому алгоритм працює швидко та використовує менше ресурсів.

Щоб поширити використання інформаційної системи на інші компанії світу, система має стати проектом з відкритим кодом, який буде опублікований на таких платформах, як GitHub, а самі артефакти мають бути доступними в популярних сховищах, таких як pip.

ДЖЕРЕЛА

1. Goyal D.P. Management Information Systems: Managerial Perspectives, 4th Edition.2014. 525pages.
2. О. В. Грицунов. Інформаційні системи та технології. 2010. 223 с.
3. Information system. [Електронний ресурс]. URL: http://surl.li/dyjma.
4. Approximate string matching. [Електронний ресурс]. URL: http://surl.li/dyjmw.
5. Donald H. Kraft, Erin Colvin. Fuzzy Information Retrieval. 2017. 63 pages.
6. Alberto Apostolico, Zvi Galil. Pattern Matching Algorithms. 1997. 400 pages.
7. D. Russell. The Principles of Computer Networking. 1989. 513 pages.
8. Roger Lee. Applied Computing & Information Technology. 2015. 194 pages.
9. N-gram. [Електронний ресурс]. URL: http://surl.li/dyjov.
10. Exist.ua. [Електронний ресурс]. URL: https://exist.ua/.
11. Doc.ua. [Електронний ресурс]. URL: https://doc.ua/.
12. Avto.pro. [Електронний ресурс]. URL: https://avto.pro/.
13. Subhāsha Candra Yādava. Introduction To Client Sever Computing. 2009. 212 pages.
14. Alex Berson. Client/server Architecture. 1996. 569 pages.
15. Changjie Tang, Xue Li, Nick Cercone, Xiaofang Zhou. Advanced Data Mining and Applications. 2008. 759 pages.
16. Tarek Sobh. Advances in Computer and Information Sciences and Engineering. 2008. 590 pages.
17. Dinesh P. Mehta, Sartaj Sahni. Handbook of Data Structures and Applications. 2018. 1120 pages.
18. Sebastian Maneth. Implementation and Application of Automata. 2009. 263 pages.
19. Rita Casadio. Algorithms in Bioinformatics. 2005. 436 pages.
20. Juha Kärkkäinen, Jens Stoye. Combinatorial Pattern Matching. 2012. 454 pages.

21. Обзоры web-фреймворков. [Електронний ресурс]. URL:https://praktikatech.wordpress.com/category/%D0%BE%D0%B1%D0%B7%D0%BE%D1%80%D1%8B-eb-%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA%D0%BE%D0%B2/page/2/.
22. Software framework. [Електронний ресурс]. URL: http://surl.li/dykyz .
23. Library (computing). [Електронний ресурс]. URL: http://surl.li/dykzs
24. Alessandro Pasetti. Software Frameworks and Embedded Control Systems. 2002. 263 pages.
25. Felipe Gutierrez. Introducing Spring Framework. 2014. 352 pages.
26. Web MVC framework. URL: http://docs.spring.io/spring/docs/current/spring-framework-
27. Hibernate. [Електронний ресурс]. URL: https://hibernate.org/.
28. Java.[Електронний ресурс]. URL: https://www.oracle.com/ru/java/
29. GWT.[Електронний ресурс]. URL: https://www.gwtproject.org/
30. Hans Bergsten. JavaServer Faces. 2004. 608 pages.
31. Graeme Rocher. The Definitive Guide to Grails. 2007. 384 pages.
32. Aristeidis Bampakos, Pablo Deeleman. Learning Angular - Third Edition. 2020. 430 pages.
33. Artemij Fedosejev. React.js Essentials. 2015. 208 pages.
34. Callum Macrae. Vue.js: Up and Running. 2018. 174 pages.
35. Bottle: Python Web Framework. [Електронний ресурс]. URL: https://bottlepy.org/docs/dev/.
36. CherryPy — A Minimalist Python Web Framework. [Електронний ресурс]. URL: https://docs.cherrypy.dev/en/latest/
37. Marty Alchin. Pro Django. 2013. 300 pages.
38. Flask. [Електронний ресурс]. URL: https://flask.palletsprojects.com/en/2.2.x/.
39. Twisted. [Електронний ресурс]. URL: https://twisted.org/

40. Tornado. [Електронний ресурс]. URL: https://www.tornadoweb.org/en/stable/
41. Pyramid [Електронний ресурс]. URL: https://trypyramid.com/
42. Stef Maruch, Aahz Maruch. Python For Dummies. 2006. 432 pages.
43. Adrian Mouat. Using Docker. 2015. 354 pages.
44. Jacob Lett. Bootstrap 4 Quick Start. 2018. 200 pages.
45. Visual Code Studio. [Електронний ресурс]. URL: https://code.visualstudio.com/
46. PyTest. [Електронний ресурс]. URL: https://docs.pytest.org/en/7.2.x/
47. Opencsv . [Електронний ресурс]. URL: https://opencsv.sourceforge.net/

Корніленко Олександр Сергійович

Прізвище, ім'я по батькові

Факультет інформаційних та прикладних технологій
Факультет

122 «Комп'ютерні науки»
Шифр і назва спеціальності

Комп'ютерні технології обробки даних (Data Science)
Освітня програма

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему «Інформаційна система пошуку автомобільних запчастин за неповним описом» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувалась у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

_____ _____

(дата) (підпис здобувача освіти)