МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«____» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"МАГІСТР"

Тема: Веб-додаток пошуку інформації про авіакомпанії

Виконавець: Богдан СМІЯН

Керівник: Анастасія ВАВІЛЕНКОВА

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО

«____»_____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Сміяна Богдана Миколайовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційної роботи (проєкту): «Веб-додаток пошуку інформації про авіакомпанії» затверджена наказом ректора від «28» серпня 2023р.№ 1494/ст.

2. Термін виконання роботи (проєкту): з 02.10.2023 р. до 31.12.2023 р.

3. Вихідні дані до роботи (проєкту): Веб-додаток пошуку інформації про авіакомпанії.

4. Зміст пояснювальної записки:

1. Основні принципи функціонування систем пошуку інформації;
2. Алгоритми пошуку даних в мережі Інтернет;
3. Програмна реалізація веб-додатку для пошуку інформації про авіакомпанії.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1. Мінімаксний метод пошуку;
2. Діаграма класів системи;
3. Схема алгоритму пошуку за ключами;
4. Основні вікна програми.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Відмітка про виконання
1.	Складання та затвердження графіку роботи кваліфікаційної роботи Написання 1 розділу, представлення керівнику	02.10.2023 – 05.10.2023	
2.	Попередній друк 1 розділу та допоміжних сторінок	06.10.2023 – 13.10.2023	
3.	Написання 2 розділу, представлення керівнику	14.10.2023 – 02.11.2023	
4.	Написання 3 розділу, представлення керівнику	02.11.2023 – 28.11.2023	
5.	Написання висновків, представлення керівнику	29.1.2023 – 01.12.2023	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	02.12.2023 – 10.12.2023	
7.	Проходження нормо-контролю, перевірка на антиплагіат, перепліт пояснювальної записки	11.12.2023 – 14.12.2023	
8.	Отримання відгуку керівника, рецензії	17.12.2023 – 18.12.2023	
9.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, *CD-R* з електронними копіями ПЗ, презентації, відгук керівника, рецензія) в папці	19.12.2023 – 20.12.2023	

7. Дата видачі завдання: “2” жовтня 2023 р.

Керівник кваліфікаційної роботи _____ Анастасія ВАВІЛЕНКОВА

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Богдан СМІЯН
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Веб-застосунок пошуку інформації про авіакомпанії»: 96 сторінок, 22 рисунка, 8 таблиць, 25 використаних джерел, 1 додаток.

ПОШУК, АВІАКОМПАНІЇ, АЛГОРИТМ, ПРОГРАМНА РОЗРОБКА, МОВА ПРОГРАМУВАННЯ *PYTHON*

Об’єкт кваліфікаційної роботи – процес інформаційного пошуку.

Предмет кваліфікаційної роботи – алгоритми пошуку для функціонування веб-додатку пошуку інформації про авіакомпанії.

Мета кваліфікаційної роботи – аналіз алгоритмів пошуку для підвищення ефективності пошуку інформації про авіакомпанії за рахунок створення спеціалізованого веб-додатку.

Методи дослідження – сучасні методи розробки веб-застосунку, мова програмування *Python*, фреймворк *Flask*.

Результатом виконання кваліфікаційної роботи є розроблений веб-застосунок для пошуку інформації про авіакомпанії.

Результати кваліфікаційної роботи рекомендується застосувати пасажирами для швидкого пошуку авіарейсів.

ЗМІСТ

ВСТУП

Актуальність теми. Актуальність веб-застосунку для пошуку інформації про авіакомпанії визначається постійно зростаючою популярністю та динамічністю сучасної авіаційної індустрії. Стрімкий розвиток глобального повітряного транспорту створює постійну потребу у зручному та ефективному інструменті для отримання інформації про різноманітні авіакомпанії.

Такий веб-застосунок має важливе значення для пасажирів, інвесторів, аналітиків та інших учасників галузі, оскільки він забезпечує доступ до надійних та актуальних даних про перевезення, пункти призначення, рейси, безпеку та інші аспекти діяльності авіакомпаній.

У контексті постійних змін та конкуренції в авіаційній галузі, важливо мати інструмент, який дозволяє швидко та зручно знаходити необхідну інформацію, щоб зробити інформовані рішення. Такий веб-застосунок може виступати важливим ресурсом для подорожуючих, бізнесу та фахівців, які вивчають ринок авіаційних послуг.

Загалом, розробка веб-застосунку для пошуку інформації про авіакомпанії відповідає сучасним вимогам мобільності та швидкісного доступу до даних, роблячи його актуальним та значущим у світлі сучасних тенденцій в авіаційній індустрії.

Мета і завдання виконання кваліфікаційної роботи. Мета виконання кваліфікаційної роботи – розробити та реалізувати веб-застосунок для пошуку інформації про авіакомпанії, який відповідає сучасним вимогам та надає користувачам зручний та ефективний інструмент для отримання актуальних даних в галузі авіації.

Завдання роботи включають:

1. Дослідити особливості інформаційного пошуку;
2. Аналіз існуючих сервісів і додатків;
3. Вивчити та дослідити роботу алгоритмів пошуку;
4. Розробити зручний і швидкодіючий веб-додаток;

5. Оцінити результати функціонування розробленого веб-додатку.

Мета цієї кваліфікаційної роботи – створити інноваційний веб-застосунок, який забезпечить користувачам легкий та ефективний спосіб отримання інформації про авіакомпанії, підвищуючи якість їхнього взаємодії з авіаційною галуззю.

Об’єкт і предмет дослідження. Об’єкт кваліфікаційної роботи – створення веб-застосунку пошуку інформації про авіакомпанії.

Предмет кваліфікаційної роботи – веб-застосунок пошуку інформації про авіакомпанії.

Методи дослідження. Щоб досягти виконання зазначених завдань кваліфікаційної роботи було вирішено застосувати навички по роботі в інтегрованому середовищі розробки *PyCharm*. Дане середовище було обрано через його здатність об'єднувати процеси розробки та підтримувати різні технології.

Мовою програмування для реалізації алгоритмів було обрано *Python*. Мова була вибрана через велику кількість підтримуваних технологій та бібліотек, зокерма вона має вбудований фреймворк *Flask*, який дозволяє створювати веб-орієнтовані системи.

Практичне значення отриманих результатів. Результатом виконання кваліфікаційної роботи є розроблений веб-застосунок для пошуку інформації про авіакомпанії.

Результати кваліфікаційної роботи рекомендується застосувати пасажирами для швидкого пошуку авіарейсів.

Особистий внесок випускника. Всі результати, представлені у кваліфікаційній роботі, отримані випускником особисто. Особливої уваги заслуговує проєктування структури додатку з використання алгоритму пошуку за ключами та мінімаксного методу.

Апробація отриманих результатів відбувалася на Міжнародній науково-технічній конференції “Інтелектуальні технології лінгвістичного аналізу”.

Прогнозні припущення про розвиток об’єкту та предмету дослідження. Полягають у можливості застосування розробленого веб-додатку як системи для

пошуку та моніторингу в різноманітних сферах життєдіяльності, у тому числі авіаційній. Подальший розвиток даного веб-додатку відкриває багато перспектив.

РОЗДІЛ 1
ОСНОВНІ ПРИНЦИПИ ФУНКЦІОНУВАННЯ СИСТЕМ ПОШУКУ ІНФОРМАЦІЇ

1.1. Інформаційний пошук. Основні характеристики інформаційного пошуку

Інформаційний пошук є ключовою складовою сучасного інформаційного суспільства, оскільки він допомагає знаходити та отримувати необхідну інформацію в мережі Інтернет та інших джерелах. Цей процес виконується завдяки використанню різноманітних інструментів та технологій, призначених для знаходження, фільтрації та представлення даних. Інформаційний пошук стає все більш актуальним у сучасному світі, де обсяг інформації швидко зростає, і доступ до неї стає все легше завдяки розвитку технологій та доступності Інтернету.

Основною метою інформаційного пошуку є забезпечення користувачів актуальною, релевантною, повною та точною інформацією. Релевантність визначає, наскільки знайдена інформація відповідає запитові користувача. Повнота оцінює, наскільки вся необхідна інформація представлена в результаті пошуку[1]. Точність вимірює ступінь правильності знайденої інформації, і пертинентність визначає, наскільки інформація відповідає конкретним потребам та контексту користувача.

Сучасні дослідження інформаційного пошуку охоплюють багато аспектів. Вчені досліджують алгоритми пошуку та ранжування результатів, розробляють методи машинного навчання для покращення точності та релевантності результатів пошуку[1]. Деякі із цих досліджень проводяться університетами та науковими лабораторіями, такими як Масачусетський технологічний інститут (MIT) та Стенфордський університет.

Для прикладу, Сергій Брін та Ларрі Пейдж, засновники *Google*, внесли значний внесок у розвиток алгоритмів пошуку та створення пошукової системи, яка революціонізувала інформаційний пошук. Їхні дослідження стосуються індексування веб-сторінок і аналізу зв'язків між ними для поліпшення якості пошуку. Варто також згадати імена вчених, які працюють над семантичним пошуком та використанням штучного інтелекту для поліпшення інформаційного пошуку, такі як Тім Бернерс-Лі та Вінсент Вінсер.

Сучасні інформаційні технології роблять інформаційний пошук доступним і зручним для широкого кола користувачів. Інтернет став основним джерелом інформації, і його обсяг зростає експоненційно. Таким чином, виникає потреба в надійних інструментах пошуку, які здатні швидко та ефективно фільтрувати інформацію з великої кількості джерел.

Однією з ключових характеристик інформаційного пошуку є можливість широкого вибору джерел та критеріїв фільтрації[2]. Користувачі можуть обирати між різними пошуковими системами, базами даних, соціальними мережами та іншими джерелами інформації залежно від їхніх потреб. Також, інформаційний пошук дозволяє застосовувати різні фільтри та параметри для точної настройки результатів пошуку.

Основні характеристики інформаційного пошуку, такі як релевантність, повнота, точність та пертинентність, грають важливу роль у забезпеченні ефективного та задовільного пошуку інформації.

Релевантність визначає, наскільки результати пошуку відповідають запитові користувача[2]. Високорелевантні результати відображають інформацію, яка найбільше відповідає потребам та очікуванням користувача. Релевантність зазвичай оцінюється за допомогою алгоритмів та методів, які враховують ключові слова, контекст та інші фактори.

Значення релевантності в інформаційному пошуку полягає у визначенні того, наскільки результати пошуку відповідають конкретному запиту користувача. Релевантність є однією з найважливіших характеристик пошуку, оскільки вона

визначає, наскільки інформація, яку отримує користувач, відповідає його потребам та очікуванням.

В процесі інформаційного пошуку, користувач вводить запит або ключові слова в пошуковий двигун або систему[2]. Важливо, щоб результати пошуку відображали інформацію, яка максимально відповідає цьому запиту. Наприклад, якщо користувач

шукає інформацію про "новітні технології у сфері авіації", релевантні результати повинні включати оновлені технології, які стосуються авіації.

Для досягнення високої релевантності, пошукові системи використовують різні методи та алгоритми. Один з найпоширеніших методів – це аналіз ключових слів та текстового контексту. Системи спробують знайти співвідношення між ключовими словами у запиті та наявними словами у веб-сторінках чи інших джерелах інформації. Чим більше співпадінь та близькість контексту, тим вища релевантність.

Додатково, пошукові системи також можуть враховувати інші фактори для покращення релевантності. Це може включати оцінку авторитету джерела, ранжування результатів за популярністю чи актуальністю інформації. Наприклад, якщо певний веб-сайт відомий своєю експертизою у сфері авіації, результати пошуку можуть віддавати перевагу цьому джерелу.

Важливо підкреслити, що релевантність є відносною і може варіюватися від користувача до користувача. Індивідуальні потреби та очікування впливають на те, як користувач сприймає релевантність результатів пошуку. Тому багато пошукових систем намагаються враховувати персоналізацію, адаптуючи результати до конкретних користувачів.

Висока релевантність є ключовим чинником для забезпечення ефективного та задовільного пошуку інформації, і вона продовжує залишатися основною метою розробників пошукових систем та дослідників в цій галузі.

Повнота вказує на те, наскільки всі необхідні або важливі аспекти інформації представлені в результатах пошуку. Повнота означає, що пошук повинен включати

всі доступні джерела та дані, які відповідають запиту. Незавершений пошук може призвести до втрати важливої інформації.

Повнота у контексті інформаційного пошуку вказує на те, наскільки вся необхідна інформація представлена в результатах пошуку[3]. Це означає, що пошук повинен включати всі доступні джерела і дані, які відповідають запиту користувача, без значних втрат або обмежень. Повнота важлива для того, щоб користувач міг отримати всю інформацію, яка йому потрібна для розв'язання конкретної проблеми чи задачі. Незавершений пошук, де деяка інформація відсутня або прихована, може призвести до недоузгодженостей та недоліків у прийнятті рішень.

Для забезпечення повноти результатів пошуку пошукові системи використовують різні стратегії та методи:

1. Індексування: пошукові системи індексують велику кількість веб-сторінок та інших джерел інформації, щоб зробити їх доступними для користувачів. Цей процес включає збір та обробку даних з різних джерел та їх подальший збереження у базі даних. Індекси повинні бути досить великими та швидкими, щоб забезпечити повноту результатів пошуку;

2. Автоматичне розширення запитів: пошукові системи можуть автоматично розширювати запити користувача, додаваючи синоніми, різні варіанти написання та пов'язані терміни. Це допомагає знайти більше інформації, навіть якщо сам пошуковий запит був обмежений;

3. Розширення обсягу пошуку: пошукові системи також можуть розширювати обсяг пошуку, включаючи результати з різних джерел, таких як новини, веб-сайти, наукові статті, соціальні мережі та багато інших. Це допомагає забезпечити різноманітність та повноту інформації;

4. Фільтрація і сортування: пошукові системи можуть фільтрувати та сортувати результати пошуку залежно від різних параметрів, таких як дата, авторитет джерела, релевантність та інші. Це допомагає користувачам знаходити найважливішу інформацію серед великої кількості даних.

Забезпечення повноти результатів пошуку є важливою метою для пошукових систем, оскільки це допомагає користувачам отримати необхідну інформацію та

зробити більш обґрунтовані рішення. Поряд з іншими характеристиками пошуку, такими як релевантність, точність та пертинентність, повнота допомагає створити ефективний інформаційний пошук, який задовольняє потреби користувачів. Точність пошуку визначає, наскільки інформація, представлена в результатах пошуку, є правильною та достовірною. Вища точність означає менше помилок та неправильних відомостей в результатах. Точність може залежати від якості джерел та методів аналізу даних.

Точність в інформаційному пошуку визначає, наскільки інформація, представлена в результатах пошуку, є правильною та достовірною[3]. Ця характеристика дуже важлива, оскільки користувачі сподіваються отримувати точну інформацію, що відповідає їхнім потребам та завданням. Неточна інформація може призвести до неправильних висновків, надмірних витрат часу на пошук та негативного впливу на прийняття рішень.

Щоб забезпечити високу точність результатів пошуку, пошукові системи та алгоритми використовують різні стратегії та методи:

1. Перевірка джерел: пошукові системи намагаються оцінити авторитет та надійність джерела інформації. Наприклад, академічні джерела або веб-сайти, які відомі своєю доброю репутацією, можуть мати перевагу у відображенні результатів;

2. Фільтрація спаму: спам та неправдива інформація можуть знизити точність результатів пошуку. Пошукові системи використовують алгоритми для виявлення та фільтрації спаму та надмірної реклами;

3. Перевірка фактів: деякі пошукові системи намагаються перевіряти факти, подані в інформації. Наприклад, вони можуть використовувати бази даних з підтвердженими фактами для перевірки інформації перед її відображенням у результатах;

4. Вдосконалення алгоритмів ранжування: алгоритми ранжування враховують якість та надійність джерела інформації при визначенні того, які результати мають бути найвище ранжовані. Розробники постійно вдосконалюють ці алгоритми для забезпечення більш точних результатів;

5. Застосування мовних моделей: сучасні мовні моделі та штучний інтелект допомагають виявляти та виправляти неточності у тексті. Наприклад, можливості автоматичного коригування помилок можуть покращити точність тексту, що відображається в результатах пошуку.

Важливо відзначити, що точність може бути визначена в контексті певного завдання чи галузі. Деякі пошукові системи, які спрямовані на надання загальної інформації, можуть бути менше точними в деяких спеціалізованих галузях. Точність також може залежати від методів та технологій, які використовуються в конкретній пошуковій системі.

Усе вищеописане підкреслює важливість точності у інформаційному пошуку. Користувачі віддають перевагу пошуковим системам та сервісам, які надають точну та надійну інформацію, і тому розробники ставлять високу точність серед своїх пріоритетів.

Пертинентність визначає, наскільки інформація відповідає потребам та контексту користувача. Результати пошуку мають бути зорієнтовані на конкретні запити та завдання користувача. Пертинентність може враховувати часові, географічні, особисті та інші параметри.

Для досягнення високої якості інформаційного пошуку, різні пошукові системи та алгоритми використовують поєднання цих характеристик. Наприклад, використання ранжування результатів за релевантністю дозволяє користувачам бачити найбільш важливу інформацію спочатку, але також слід враховувати повноту та точність результатів. Персоналізація також грає важливу роль у забезпеченні пертинентності, оскільки дозволяє налаштовувати результати під потреби кожного окремого користувача.

Четверта характеристика інформаційного пошуку – пертинентність. Ця характеристика вказує на те, наскільки інформація, яка знаходиться в результатах пошуку, відповідає конкретним потребам та контексту користувача. Пертинентність важлива, оскільки та сама інформація може бути перетворена в перетинентну або неперетинентну в залежності від того, як вона використовується та в якому контексті.

Основні аспекти пертинентності включають:

1. Контекст користувача: для того, щоб інформація була пертинентною, важливо враховувати поточні потреби та завдання користувача. Наприклад, якщо користувач шукає інформацію про технічні характеристики сучасних авіакомпаній, інформація про їхню історію чи інші аспекти може бути менш пертинентною в цьому контексті;

2. Часові рамки: пертинентність залежить від часових рамок. Інформація, яка була актуальною років тому, може втратити свою перетинентність, якщо користувач шукає найсвіжішу інформацію. Пошукові системи часто намагаються надавати перевагу актуальним джерелам та даним;

3. Географічні параметри: пертинентність може бути пов'язана з географічними параметрами. Наприклад, якщо користувач шукає інформацію про авіакомпанії, що працюють в певному регіоні або країні, результати пошуку повинні враховувати цей географічний контекст;

4. Специфічні завдання: пертинентність також залежить від конкретного завдання користувача. Якщо користувач шукає інформацію для рішення конкретної задачі, то інформація, яка допомагає досягти цієї мети, є найбільш пертинентною.

Для досягнення високої пертинентності, пошукові системи використовують різні методи та стратегії. Це включає в себе аналіз історії пошуку та поведінки користувачів, використання персоналізованих рекомендацій, а також врахування конкретних фільтрів та параметрів, які користувач може задати в своєму запиті[3].

Забезпечення пертинентності є ключовим завданням для пошукових систем, оскільки користувачі часто оцінюють якість пошуку з точки зору того, наскільки інформація відповідає їхнім потребам та контексту. Розробники та дослідники в галузі інформаційного пошуку продовжують працювати над покращенням цієї характеристики, щоб забезпечити більш ефективний та задовільний пошук для користувачів. Усі ці характеристики важливі для забезпечення задоволеності користувача та досягнення успіху в інформаційному пошуку, незалежно від галузі чи завдання.

Щоб краще зрозуміти особливості ключових аспектів інформаційного пошуку та їх значення для забезпечення якісного та ефективного пошуку інформації в різних сферах та сценаріях було створено таблицю 1.1.

Таблиця 1.1

Аспекти інформаційного пошуку

Аспект	**Опис**	**Значення для інформаційного пошуку**
Релевантність	Визначає, наскільки результати відповідають запиту користувача.	Висока релевантність важлива для представлення користувачам відповідної та потрібної інформації.
Повнота	Визначає, наскільки всі важливі аспекти інформації представлені в результатах пошуку.	Повнота допомагає уникнути втрати важливої інформації та незавершених результатів.
Точність	Вказує на те, наскільки інформація в результатах є правильною та достовірною.	Вища точність дозволяє уникнути поширення неправильної інформації та помилок.
Пертинентність	Визначає, наскільки інформація відповідає потребам та контексту користувача.	Пертинентність забезпечує зорієнтованість результатів на конкретного користувача та його завдання.
Ефективність	Вказує на продуктивність та швидкість пошуку інформації.	Швидкий та ефективний пошук дозволяє користувачам економити час та зусилля.
Адаптованість	Визначає можливість налаштування пошуку під конкретні потреби користувача.	Адаптованість допомагає задовольняти різноманітні індивідуальні запити користувачів.
Актуальність	Вказує на те, наскільки інформація є свіжою та оновленою.	Актуальність дозволяє користувачам отримувати найсвіжу та релевантну інформацію.

Географічний контекст	Визначає врахування географічних параметрів у пошуку інформації.	Врахування географічного контексту важливе для користувачів, що шукають локальну інформацію.
Індивідуалізація	Вказує на можливість персоналізації результатів пошуку для конкретного користувача.	Персоналізований пошук допомагає надавати індивідуальну та перетинентну інформацію.

Однією з важливих тенденцій у сучасному інформаційному пошуку є персоналізація. Компанії та пошукові системи намагаються аналізувати попередні запити користувачів і їхні поведінкові дані, щоб надавати більш релевантні результати пошуку. Це допомагає зробити інформаційний пошук більш ефективним і зручним для кожного окремого користувача.

Що стосується джерел інформації, то важливо зазначити, що багато вчених працюють над покращенням якості та релевантності результатів пошуку в мережі Інтернет. Наприклад, Аманда Спенсер і Джозеф Тюркевіч з Університету Іллінойсу займаються дослідженнями в галузі інформаційного пошуку та розробляють алгоритми для вдосконалення пошукових систем.

Крім того, інформаційний пошук є важливим аспектом не тільки для індивідуальних користувачів, але і для різних сфер діяльності, включаючи бізнес, освіту, наукове дослідження та багато інших галузей. В бізнесі, точний та релевантний пошук інформації може допомогти в прийнятті стратегічних рішень, а також в дослідженні ринку та конкурентів. У сфері освіти, інформаційний пошук розширює можливості для навчання та досліджень, надаючи доступ до великої кількості навчальних та дослідницьких ресурсів. Діаграма, яка візуалізує розподіл важливих аспектів інформаційного пошуку та їх відсотковий внесок у загальну картину відображено на рис. 1.1.

Рис. 1.1. Відображення аспектів інформаційного пошуку

Наукові дослідження у галузі інформаційного пошуку включають в себе розвиток нових методів аналізу даних та алгоритмів пошуку. Це дозволяє постійно вдосконалювати пошукові системи та забезпечувати користувачів більш точною та релевантною інформацією.

Щодо інтернет-пошуку загалом, важливо зазначити, що великі корпорації, такі як *Google*, *Microsoft* і *Amazon*, вкладають значні ресурси в розробку та покращення пошукових технологій. Вони розвивають нові методи пошуку, використовують штучний інтелект та машинне навчання для поліпшення результатів пошуку і рекомендацій для користувачів. Дослідники та вчені продовжують розвивати нові методи та технології, щоб зробити інформаційний пошук більш точним, релевантним та корисним для всіх користувачів, і це робить цю галузь постійно рухливою та захопливою для дослідників та фахівців у цій сфері.

Для зрозуміння важливості інформаційного пошуку, важливо також враховувати його роль у вирішенні сучасних проблем та викликів. За допомогою пошуку інформації, можна вивчати та розв'язувати складні глобальні проблеми, такі як зміна клімату, медичні дослідження та багато інших. Інформаційний пошук

допомагає науковцям та експертам в доступі до даних, які можуть визначити майбутнє нашої планети.

Також важливо зазначити, що інформаційний пошук має свої етичні та соціокультурні аспекти. З підвищенням потужності інформаційних технологій і аналітики виникають питання щодо приватності даних та контролю за інформацією. Вчені та фахівці в галузі інформаційного пошуку розглядають ці питання та намагаються розв'язати їх, щоб забезпечити ефективний та етичний інформаційний пошук.

Наукові дослідження у галузі інформаційного пошуку насправді є ключовими для постійного вдосконалення цієї сфери. Розробники та дослідники ведуть постійну роботу над створенням нових методів аналізу даних та алгоритмів пошуку, щоб забезпечити найвищу точність та релевантність результатів. Інтеграція штучного інтелекту та машинного навчання стає дедалі важливішою для покращення пошукових систем.

Особливу увагу привертають великі технологічні корпорації, які інвестують в цей сектор значні кошти. *Google, Microsoft та Amazon* активно розробляють нові методи пошуку, використовуючи передові технології та методику машинного навчання. Це допомагає зробити пошук більш ефективним та відповідати індивідуальним потребам користувачів.

З іншого боку, науковці та вчені продовжують розвивати нові методи та технології для покращення якості інформаційного пошуку. Вони досліджують алгоритми, які допомагають знаходити більш точну та об'єктивну інформацію, а також методи визначення надійності та авторитетності джерел.

Інформаційний пошук також відіграє важливу роль у розв'язанні актуальних глобальних проблем, таких як зміна клімату, медичні дослідження та багато інших. Завдяки доступності та швидкому пошуку інформації, науковці можуть звертатися до великих обсягів даних, щоб розробляти рішення для сучасних викликів та проблем.

Не можна залишити осторонь і етичні та соціокультурні аспекти інформаційного пошуку. З ростом можливостей інформаційних технологій,

з'являються питання щодо приватності даних та контролю над інформацією. Ці питання стають складнішими і вимагають уважного вивчення та вирішення з боку дослідників та фахівців в галузі інформаційного пошуку.

Інформаційний пошук в сучасному світі впливає не лише на індивідуальні потреби та галузі досліджень, але також має значущий вплив на суспільну динаміку та демократію. Розповсюдження фейкових новин та маніпуляція інформацією стали серйозними викликами, і відмова від доступу до достовірної інформації може позначитися на суспільстві.

Інформаційний пошук також сприяє розвитку освіти та науки, роблячи знання доступними в будь-який час та з будь-якого місця. Вчені мають можливість швидкого доступу до джерел, що дозволяє проводити більше досліджень та вивчати нові галузі знань.

З іншого боку, інформаційний пошук вимагає від користувачів критичного мислення та навичок оцінки джерел. Важливо розвивати медіаосвіту та навчити користувачів розрізняти достовірну інформацію від недостовірної.

Суспільні діалоги про роль інформаційного пошуку також відкривають питання про демократію та свободу слова. Контроль над доступом до інформації може бути використаний для обмеження свободи вираження думки, і це вимагає уваги до питань прозорості та прав громадян.

Зважаючи на всі ці важливі аспекти інформаційного пошуку, слід пам'ятати, що ця сфера залишається динамічною та еволюційною. Розробка нових методів та технологій, а також постійна увага до етичних питань, важлива для забезпечення справедливого, точного та демократичного інформаційного пошуку в сучасному світі.

Інформаційний пошук також відіграє важливу роль у впливі на споживчі та бізнес-процеси. Великі компанії використовують дані та інформацію з пошукових систем для прийняття рішень, розробки маркетингових стратегій та аналізу конкурентного середовища[3]. Завдяки пошуковим системам бізнеси можуть краще розуміти своїх клієнтів та аналізувати ринкові тенденції.

Важливо зазначити, що інформаційний пошук має значущий вплив на глобальну економіку. Він сприяє розвитку інновацій та підприємництва, створюючи нові можливості для бізнесу та ринку праці. Компанії, які використовують дані з пошукових систем, можуть стати більш конкурентоспроможними та адаптивними до змін.

Ще однією важливою роллю інформаційного пошуку є сприяння інноваціям і науковим дослідженням. Доступ до великих обсягів інформації дозволяє науковцям та вченим проводити більше досліджень, розвивати нові технології та вирішувати важливі суспільні та глобальні проблеми.

Крім вже згаданих аспектів, інформаційний пошук є також важливим фактором для політичних процесів та громадянського участі. Він дозволяє громадянам отримувати інформацію про політичних кандидатів, партії та глобальні політичні події, що сприяє усвідомленості та активному участю у громадянському житті[3]. Онлайн-платформи та соціальні мережі грають важливу роль у формуванні громадської думки і розповсюдженні політичних поглядів.

Однак інформаційний пошук також стикається з викликами в сфері дезінформації та фільтрування інформації. Розповсюдження неправдивої інформації може вплинути на громадську думку та призвести до відмови від об'єктивної реальності. Контроль за алгоритмами пошуку та аналізом контенту стає важливим завданням для забезпечення об'єктивного та правдивого інформаційного пошуку.

Крім того, інформаційний пошук може стати предметом геополітичних конфліктів і цензури. Деякі країни обмежують доступ до інтернету та інформації з метою контролю над громадянами та обмеженням свободи слова. В цьому контексті захист приватності та свободи доступу до інформації стають важливими цілями для об'єднання громадянського суспільства.

Загалом, інформаційний пошук в сучасному світі є важливим і актуальним процесом, який використовується в багатьох сферах, включаючи авіаційну індустрію. Вчені інформаційного пошуку надають вагому підтримку в розробці та вдосконаленні інструментів для забезпечення ефективного доступу до інформації в мережі та допомагають користувачам знаходити потрібні дані швидко та ефективно.

Вони продовжують досліджувати та розробляти нові методи та технології для поліпшення інформаційного пошуку, щоб забезпечити доступ до якісної та релевантної інформації у світі, який постійно змінюється та розвивається.

1.2. Особливості функціонування сучасних сервісів пошуку інформації

Сучасні сервіси пошуку інформації в авіаційній сфері є незамінними інструментами для подорожуючих та фахівців у галузі авіації. Їхній функціонал розширюється і вдосконалюється, відповідаючи наростаючим потребам користувачів. Дослідження роботи цих сервісів та інформаційно-пошукових систем дозволяє нам зрозуміти їх важливість і вплив на авіаційну і інші сфери.

Одним із популярних сервісів є *Google Flights*. Його головна мета – надати користувачам можливість швидко та зручно знаходити та порівнювати ціни на авіаквитки. Основний функціонал включає в себе пошук рейсів за різними критеріями, відображення цін на квитки в режимі реального часу, можливість створювати оповіщення про зміні цін. Великою перевагою є інтеграція з іншими сервісами *Google*, що спрощує планування подорожей. Однак, недоліком може бути обмеження в виборі додаткових послуг та обмежена кількість партнерських авіакомпаній.

Інший популярний сервіс – *Skyscanner*, пропонує велику кількість опцій для пошуку та порівняння рейсів. Основний функціонал включає в себе пошук авіаквитків, готелів, оренди авто, а також можливість встановлення фільтрів за багатьма параметрами. Скріншот роботи *Skyscanner* показує інтуїтивний і простий у використанні інтерфейс. Перевагою є можливість порівняння цін від різних постачальників та наявність додаткових послуг. Недоліком може бути не завжди точна інформація щодо доступності квитків.

Kayak – ще один популярний інструмент для пошуку інформації про авіакомпанії та рейси. Його призначення полягає в знаходженні найкращих пропозицій на авіаквитки, готелі та інші послуги під час подорожей. Основний функціонал включає в себе широкий вибір фільтрів, можливість налаштування

цінових оповіщень, а також порівняння пропозицій від різних партнерів. Сервіс *Kayak* часто отримує позитивні відгуки за зручний та інформативний інтерфейс. Недоліком може бути не завжди оптимальний вибір опцій для деяких маршрутів.

Усі ці сервіси роблять значний внесок у полегшення пошуку інформації про авіакомпанії та подорожі. Вони спрощують процес вибору рейсів та дозволяють знайти найкращі пропозиції на ринку. Такі інструменти корисні не тільки для подорожуючих, але і для фахівців у сфері авіації, які можуть використовувати їх для моніторингу конкурентів та аналізу ринку.

Далі варто відзначити інформаційно-пошукові системи в авіаційній сфері. Однією з них є система *Amadeus*, яка надає рішення для авіакомпаній та туристичних агентств. Головна мета системи полягає в обробці бронювань, видачі квитків та управлінні даними. Її основний функціонал включає в себе можливість швидкого пошуку рейсів, обробку бронювань, видачу квитків та підтримку зв'язку між різними ланками авіаційного ланцюжка. Перевагою *Amadeus* є висока надійність і розширена функціональність, але недоліком може бути велика складність та високі витрати на впровадження.

Ще однією інформаційно-пошуковою системою є *Sabre*, яка також надає послуги для авіакомпаній та туристичних агентств. Основна мета *Sabre* – оптимізація процесів продажу авіаквитків і послуг. Система має розширений функціонал, включаючи обробку багатьох видів послуг, аналітику та звітність. Вона також надає інструменти для управління запасами та лояльність клієнтів. Сабре володіє високою масштабованістю та ефективністю, але може бути складною у використанні для некваліфікованих користувачів.

Поза авіаційною сферою, інформаційно-пошукові системи також знайшли своє застосування. Наприклад, система *Elasticsearch* використовується для пошуку та аналізу даних у різних галузях. Вона забезпечує можливість індексації великих обсягів даних та швидкого пошуку за ключовими словами. *Elasticsearch* використовується в медицині, фінансах, інтернет-пошуку та багатьох інших галузях.

Щоб швидко порівняти сервіси за їхнім призначенням, функціоналом, перевагами та недоліками було створено таблицю 1.2.

Таблиця 1.2

Порівняння сервісів

Сервіс	**Призначення**	**Основний функціонал**	**Переваги**	**Недоліки**
Google Flights	Пошук та порівняння авіаквитків	Пошук рейсів, відображення цін у режимі реального часу, оповіщення	Простий інтерфейс, інтеграція з іншими Google-продуктами	Обмеженість вибору додаткових послуг, обмежена кількість партнерських авіакомпаній
Skyscanner	Пошук та порівняння авіаквитків, готелів	Великий вибір фільтрів, цінові оповіщення	Зручний інтерфейс, порівняння цін від різних постачальників	Не завжди точна інформація про доступність квитків

Закінчення таблиці 1.2

Сервіс	**Призначення**	**Основний функціонал**	**Переваги**	**Недоліки**
Kayak	Пошук найкращих пропозицій на подорожі	Вибір фільтрів, цінові оповіщення, порівняння пропозицій	Інтуїтивний інтерфейс, порівняння цін від різних постачальників	Не завжди оптимальний вибір опцій для деяких маршрутів
Amadeus	Рішення для авіакомпаній і турагентств	Обробка бронювань, видача квитків, управління даними	Висока надійність, розширена функціональність	Складність та високі витрати на впровадження
Sabre	Оптимізація процесів продажу авіаквитків	Обробка бронювань, аналітика, управління запасами, лояльність	Висока масштабованість, ефективність	Складність у використанні для некваліфікованих користувачів

Зважаючи на значення *Google Flights*, *Skyscanner, Kayak*, *Amadeus* та *Sabre* у сучасному світі подорожей, варто детальніше розглянути їхню роботу та особливості.

Google Flights – це популярний сервіс, який спеціалізується на пошуку та порівнянні авіаквитків для подорожуючих. Однією з його ключових функцій є можливість швидкого та зручного пошуку рейсів за різними критеріями, включаючи дати, місця вильоту та прильоту, кількість пересадок, та інші параметри. Інтерфейс *Google Flights* відзначається простотою та зрозумілістю, що дозволяє користувачам легко налаштовувати свої пошуки та знаходити оптимальні варіанти[4]. Сервіс також відображає ціни на квитки в режимі реального часу, що дозволяє користувачам вибирати оптимальний момент для придбання квитків. Крім того, *Google Flights* надає можливість створювати оповіщення про зміни цін, що допомагає зекономити гроші під час планування подорожей (рис.1.2).

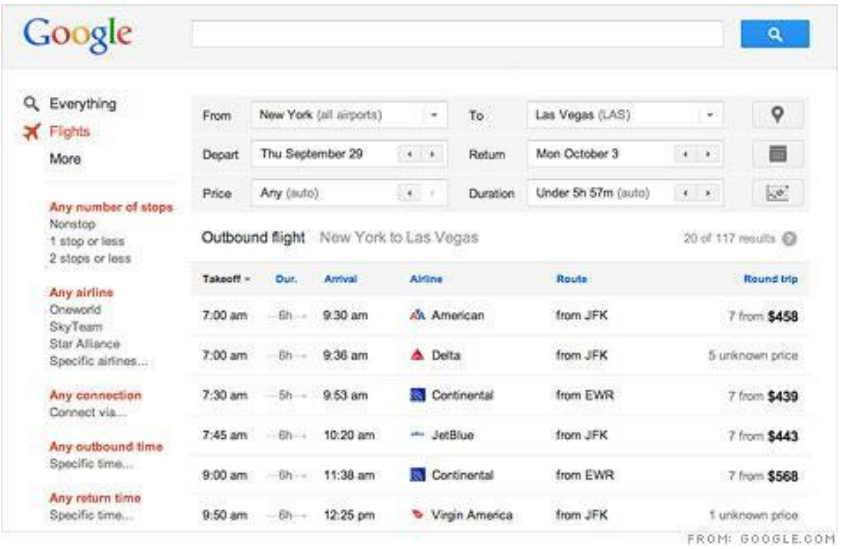


Рис. 1.2. Робоче середовище *Google Flights*

Про переваги *Google Flights* варто зазначити інтеграцію з іншими сервісами *Google*, такими як *Google Maps* та *Google Calendar*. Це полегшує планування маршрутів та додавання подорожей до календаря. Однак слід враховувати, що сервіс обмежений вибором додаткових послуг, таких як багаж чи обране місце у літаку, і кількість партнерських авіакомпаній може бути обмеженою. Незважаючи на ці

недоліки, *Google Flights* є популярним та корисним інструментом для тих, хто шукає оптимальні авіаквитки та бажає спростити процес планування подорожей.

Skyscanner – ще один важливий гравець на ринку сервісів пошуку авіаквитків та готелів. *Skyscanner* дозволяє користувачам знаходити не тільки авіаквитки, але й забронювати готелі та автомобілі для подорожей. Основною перевагою *Skyscanner* є широкий спектр фільтрів, що дозволяють користувачам настроювати свій пошук відповідно до власних потреб[5]. Ви можете вибирати критерії, такі як бюджет, кількість пересадок, година вильоту, і багато інших. Ця гнучкість робить *Skyscanner* привабливим інструментом для тих, хто шукає персоналізовані варіанти (рис.1.3).

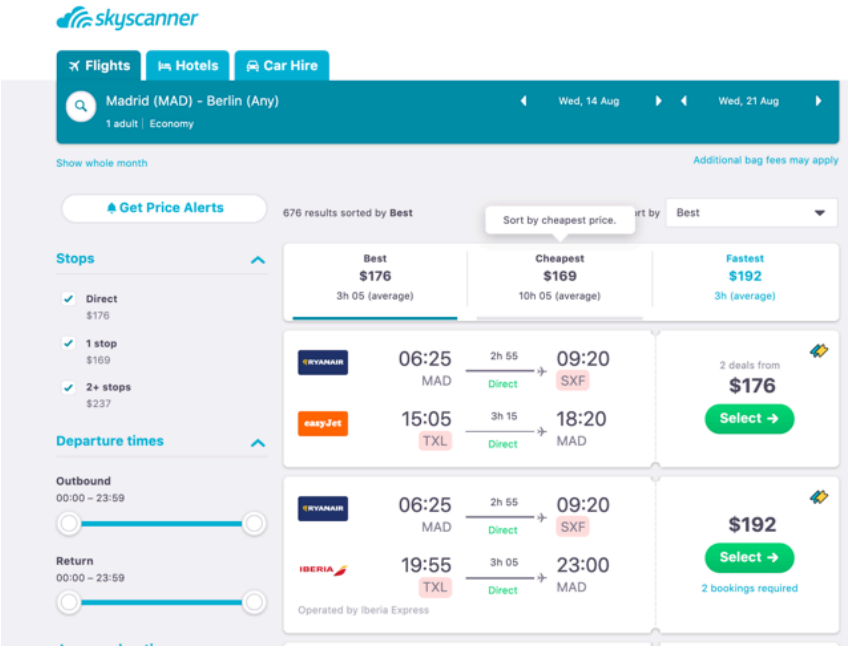


Рис. 1.3. Робоче середовище *Skyscanner*

Окрім того, *Skyscanner* дозволяє користувачам встановлювати цінові оповіщення, що допомагає вчасно реагувати на зміни цін на квитки. Наявність додаткових послуг, таких як оренда автомобіля або бронювання готелів, робить *Skyscanner* універсальним інструментом для планування подорожей. Однак варто враховувати, що не завжди інформація про доступність квитків є абсолютно точною, іноді ціни можуть змінюватися після бронювання. В цілому, *Skyscanner* є потужним та зручним інструментом для тих, хто бажає максимально вибирати та налаштовувати свої подорожі.

Поговоримо про *Kayak* та *Amadeus* – два сервіси в авіаційній сфері, які грають важливу роль, але мають відмінні призначення та функціонал.

Kayak – це інтернет-сервіс, призначений для пошуку найкращих пропозицій на авіаквитки, готелі та інші послуги, необхідні під час подорожей[6]. Основною перевагою *Kayak* є широкий вибір фільтрів, які користувачі можуть використовувати для налаштування своїх пошуків. Вони можуть вибирати критерії, такі як бюджет, час вильоту, кількість пересадок, а також багато інших параметрів, що дозволяє знайти найкращі варіанти, відповідно до своїх потреб. Інтерфейс *Kayak* відзначається інтуїтивністю та зручністю використання, що дозволяє користувачам легко користуватися сервісом (рис.1.4).

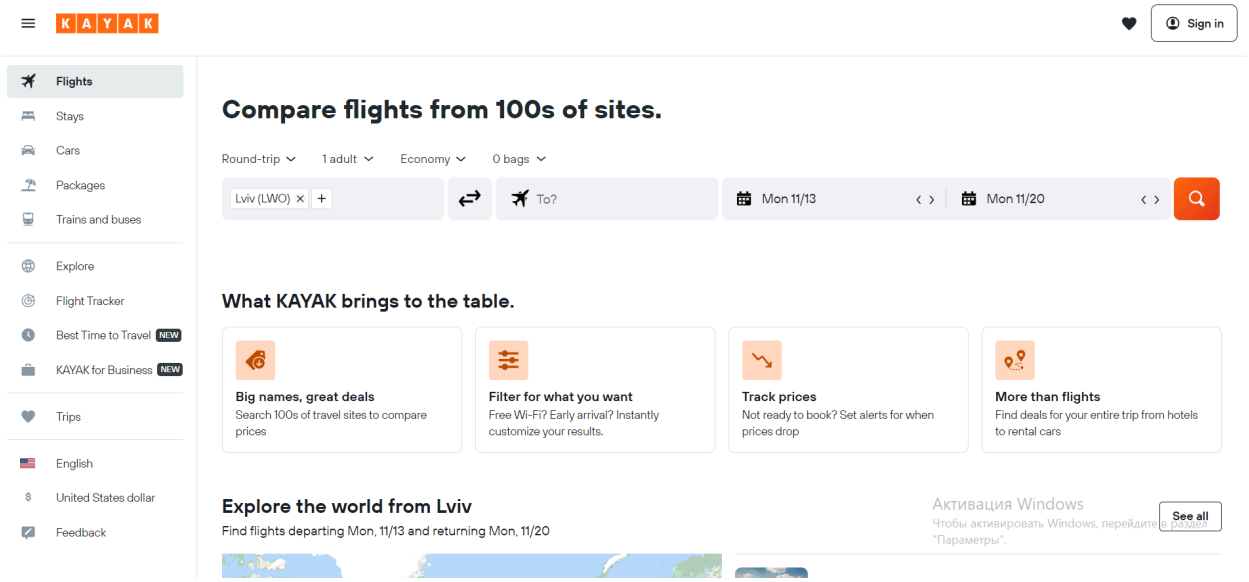


Рис. 1.4. Робоче середовище *Kayak*

Окрім можливості налаштування пошуку, *Kayak* надає користувачам можливість встановлення цінових оповіщень, які допомагають вчасно реагувати на зміни цін на авіаквитки. Крім того, сервіс дозволяє порівнювати пропозиції від різних постачальників та вибирати найвигідніші. На жаль, не завжди оптимальний вибір опцій для деяких маршрутів може бути недоліком. У цілому, *Kayak* є важливим інструментом для тих, хто прагне мати максимальний контроль над процесом пошуку та планування подорожей.

Amadeus – це інформаційна система, яка надає рішення для авіакомпаній та туристичних агентств з обробки бронювань та видачі авіаквитків. Головною метою *Amadeus* є полегшення процесів управління бронюваннями та забезпечення зв'язку між різними ланками логістичного ланцюжка авіації. Основний функціонал *Amadeus* включає обробку бронювань, видачу квитків, а також управління даними про пасажирів та рейси[7]. Сервіс допомагає авіакомпаніям ефективно керувати своєю операційною діяльністю та забезпечувати зручний та надійний сервіс для клієнтів (рис.1.5).

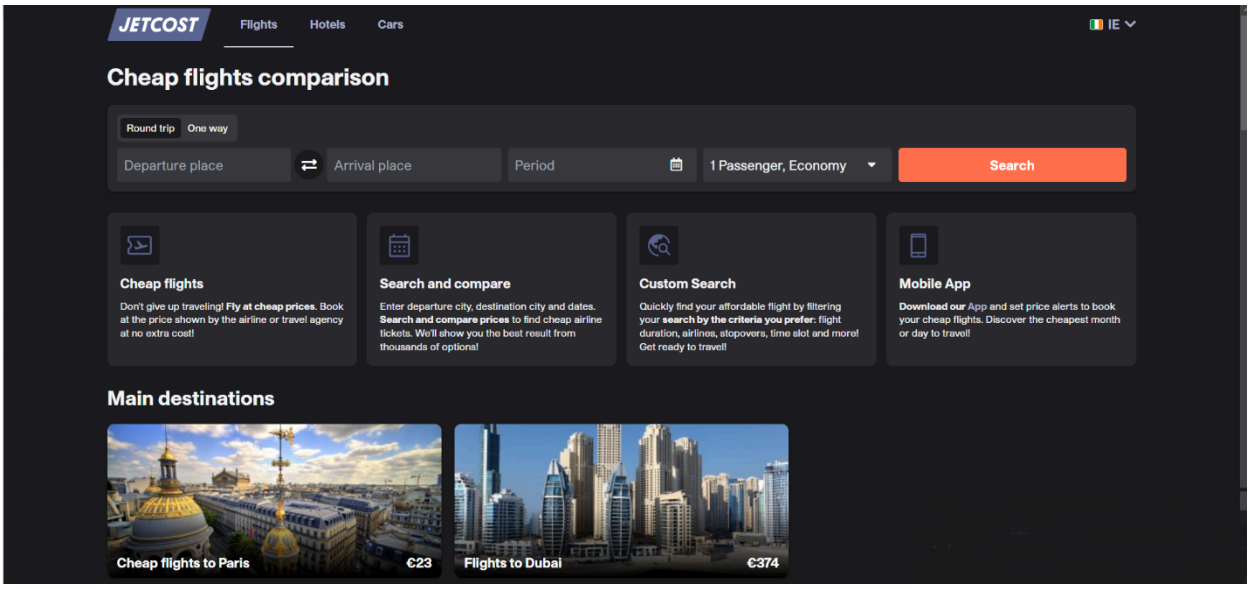


Рис. 1.5. Робоче середовище *Amadeus*

Однією з основних переваг *Amadeus* є його висока надійність та розширена функціональність. Сервіс володіє обширним набором інструментів для авіакомпаній, які дозволяють їм оптимізувати свою роботу. Однак варто враховувати, що впровадження системи *Amadeus* може бути дорогим та складним процесом, що може вимагати значних зусиль та ресурсів. Все ж, Amadeus є невід'ємною частиною бізнесу авіаційної галузі та сприяє ефективному управлінню авіаперевезеннями та обробці бронювань.

Sabre – це інформаційна система, що спеціалізується на оптимізації процесів продажу авіаквитків та послуг для авіакомпаній та туристичних агентств. Основна мета *Sabre* – надавати комплексні рішення для авіаційної індустрії з метою

полегшення їхньої діяльності та забезпечення зручності для пасажирів [8]. Основний функціонал системи включає в себе обробку бронювань, аналітику, звітність, управління запасами та програми лояльності клієнтів (рис.1.6).

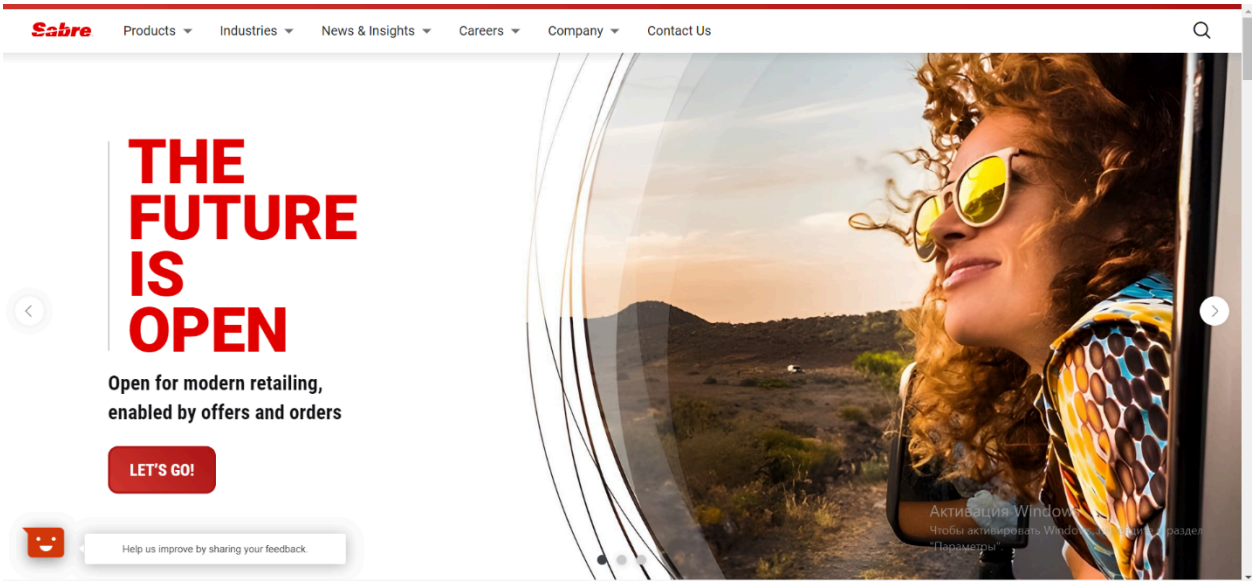


Рис. 1.6. Робоче середовище *Sabre*

Перевагами *Sabre* є висока масштабованість та ефективність. Ця система дозволяє авіакомпаніям оптимізувати свої процеси, включаючи розкладання рейсів, обробку багажу, планування запасів та багато інших аспектів діяльності. Крім того, *Sabre* допомагає створювати та керувати програмами лояльності для пасажирів, що дозволяє залучати та утримувати клієнтів.

Однак варто враховувати, що система *Sabre* може бути складною у використанні для некваліфікованих користувачів та вимагає спеціалізованого навчання. Також, впровадження та підтримка цієї системи може бути дорогими. Проте для авіакомпаній, які прагнуть забезпечити високий рівень обслуговування пасажирів та ефективно керувати своєю діяльністю, *Sabre* залишається важливим інструментом. Вона допомагає авіаційній галузі поліпшити якість та швидкість надання послуг та бути конкурентоспроможною на ринку подорожей.

Інформаційно-пошукові системи в авіаційній сфері також грають значну роль і виконують специфічні функції. Наприклад, система *FlightAware* спрямована на моніторинг та відстеження рейсів у режимі реального часу. Головна її мета –

надавати користувачам точну інформацію про місцеположення літаків, їхній статус та прогнозований час прильоту. Система відображає графіки руху літаків на карті та надає дані про погоду на маршруті. Великою перевагою є можливість слідкувати за рейсами близьких та рідних. Однак, недоліком є відсутність можливості здійснювати бронювання квитків через цю систему.

З іншого боку, сервіси, які надають інформацію про візи та правила в'їзду в різні країни, такі як *iVisa*, допомагають подорожуючим отримувати необхідну інформацію щодо візових вимог. Основний функціонал полягає в наданні інформації про доступні візи для конкретного пасажира в залежності від його громадянства та мети подорожі. Система допомагає користувачам оформити візи та інші документи для в'їзду в інші країни. Перевагою є ефективний та швидкий процес отримання інформації та віз. Однак, недоліком може бути наявність додаткової плати за послуги сервісу.

Ще однією важливою категорією інформаційно-пошукових систем в авіаційній сфері є системи онлайн-бронювання авіаквитків та готелів. Такі платформи, як *Expedia* та *Booking.com*, дозволяють користувачам не лише знаходити інформацію про доступні авіаквитки та готелі, але і здійснювати бронювання та оплату через інтернет. Основний функціонал цих систем включає в себе пошук готелів та авіаквитків за різними критеріями, можливість порівняння цін, відгуки та рейтинги від інших користувачів. Перевагами є широкий вибір послуг, можливість ефективного планування подорожей та економії часу. Недоліками можуть бути можливі обмеження у виборі та доступності послуг у деяких регіонах.

Пошукові системи також активно використовуються у сферах пілотажу та обслуговування літаків. Наприклад, система *SkyVector* надає пілотам інформацію про повітряні маршрути та погодні умови, допомагаючи планувати безпечні польоти. Система *Flightradar24* надає відомості про рух літаків у режимі реального часу, що корисно для контролю та безпеки польотів. Такі системи стали важливими інструментами для авіакомпаній та пілотів, сприяючи ефективному та безпечному руху повітряному транспорту. Недоліками можуть бути технічні обмеження та високі витрати на обслуговування.

Однак важливо розуміти, що кожен інструмент має свої переваги та недоліки, і вибір залежить від конкретних потреб користувача. Все це свідчить про важливість інформаційних технологій у сучасному світі та їхню роль у розвитку авіаційної галузі та інших галузей.

У підсумку, інформаційно-пошукові системи та сервіси пошуку інформації про авіакомпанії грають ключову роль у спрощенні доступу до важливої інформації. Вони допомагають користувачам ефективно планувати подорожі та знаходити найкращі пропозиції. В той же час, розробники і фахівці в галузі авіації мають можливість використовувати інформаційні системи для покращення обробки даних та оптимізації бізнес-процесів. Таким чином, ці інструменти стають невід'ємною частиною авіаційної та інших галузей, сприяючи їхньому розвитку та ефективності. Безумовно, ці інструменти допомагають спростити і поліпшити процес пошуку та планування подорожей в сучасному світі авіації.

Постановка задачі кваліфікаційної роботи полягає у наступному:

Завдання:

1. Ознайомитися з особливостями та характеристиками інформаційного пошуку зокрема в контексті авіаційної галузі;

2. Проаналізувати роботу існуючих сервісів пошуку інформації в авіаційній сфері, включаючи їхні функції, переваги та недоліки;

3. Дослідити архітектуру сучасних пошукових систем та визначити основні компоненти, які використовуються у подібних сервісах;

4. Проаналізувати роботу різних алгоритмів пошуку та визначити найбільш відповідний для подальшого розроблення веб-додатку пошуку інформації про авіакомпанії;

5. Створити веб-додаток для пошуку інформації про авіакомпанії, розробити його архітектуру та використовувані алгоритми, а також забезпечити можливість користувачам здійснювати пошук та отримувати інформацію про авіакомпанії згідно з вимогами та критеріями;

6. Оцінити результати роботи розробленого веб-додатку, включаючи його продуктивність та ефективність у порівнянні з існуючими сервісами.

Загальна мета кваліфікаційної роботи полягає в розробці та оцінці веб-додатку для пошуку інформації про авіакомпанії, який використовує вибраний алгоритм пошуку та надає користувачам зручний інструмент для отримання необхідної інформації в цій галузі.

1.3. Висновки до розділу

У даному розділі ми детально розглянули роботу п'яти важливих сервісів та систем в авіаційній сфері. Ця аналітична огляд виявив різноманітні сервіси, кожен з яких має свої особливості та переваги.

Основні принципи функціонування систем пошуку інформації, які я розглянув, свідчать про значущість цих інструментів у сучасному світі. Незалежно від того, чи ми розглядаємо сервіси для пошуку авіабілетів або інформаційно-пошукові системи для авіаційної галузі, вони спрощують життя користувачів та роблять процес пошуку і аналізу інформації набагато ефективнішим.

Специфічність та функціонал кожного інструменту можуть варіюватися, але загальний принцип полягає в тому, щоб надати користувачам доступ до різноманітних даних та можливостей для прийняття обґрунтованих рішень. Це особливо актуально в авіаційній галузі, де велика кількість інформації про рейси, ціни, наявність квитків і додаткові послуги може здаватися складною для орієнтації.

Переваги використання таких систем включають у собі швидкий доступ до актуальних даних, можливість порівняння пропозицій від різних постачальників, а також спрощення процесу планування подорожей. Для фахівців в авіаційній галузі інформаційні системи надають можливість аналізу ринку та конкурентоспроможності, що є важливим для прийняття стратегічних рішень.

Google Flights та *Skyscanner*, як сервіси пошуку авіаквитків для подорожуючих, надають широкий функціонал та можливості налаштування пошуку, що дозволяє користувачам знаходити оптимальні варіанти та ефективно планувати свої подорожі. Однак важливо враховувати їхні обмеження, такі як обмежена кількість партнерських авіакомпаній та можливості вибору додаткових послуг.

Kayak, з іншого боку, відзначається широким вибором фільтрів та можливістю налаштовування пошуку за різними параметрами, що дозволяє користувачам більш детально контролювати свої пошуки. Але, як і в інших сервісів, існують певні обмеження.

Amadeus та *Sabre*, на відміну від перших трьох, це системи, призначені для авіакомпаній та туристичних агентств з метою оптимізації бізнес-процесів та поліпшення обслуговування клієнтів. Вони надають рішення для управління бронюваннями, видачі квитків, аналітики та інших аспектів авіаційної галузі. Вони є важливими для великих гравців на ринку авіаперевезень, але вимагають складних інтеграцій та великих витрат.

Кожен з цих сервісів та систем грає важливу роль у спрощенні процесів пошуку та планування подорожей в авіаційній сфері. Користувачі мають можливість вибору сервісу, який найкраще відповідає їхнім потребам та очікуванням. Разом із тим, важливо враховувати, що кожен із них має свої переваги та недоліки, і вибір залежатиме від конкретних цілей користувача. З цим розумінням, користувачі зможуть знайти оптимальний спосіб планування та бронювання своїх майбутніх подорожей у світі авіації.

Звідси випливає, що важливість систем пошуку інформації в авіаційній сфері та інших галузях не може бути переоцінена. Вони сприяють покращенню доступу до інформації, роблячи її більш доступною та зрозумілою для всіх користувачів. Завдяки цим системам, подорожі стають зручнішими та більш доступними, а робота в галузі авіації стає більш ефективною та конкурентоспроможною.

РОЗДІЛ 2
АЛГОРИТМИ ПОШУКУ ДАНИХ В МЕРЕЖІ ІНТЕРНЕТ

2.1. Архітектура інформаційно-пошукових систем

У контексті розробки веб-додатку, призначеного для пошуку інформації про авіакомпанії, необхідно визначити оптимальну архітектурну конфігурацію системи. З огляду на високу динамічність та обсяг інформації, що потребує обробки, розглядається модульна архітектура, що базується на сучасних принципах розробки програмного забезпечення. Структура визначена на рис. 2.1.

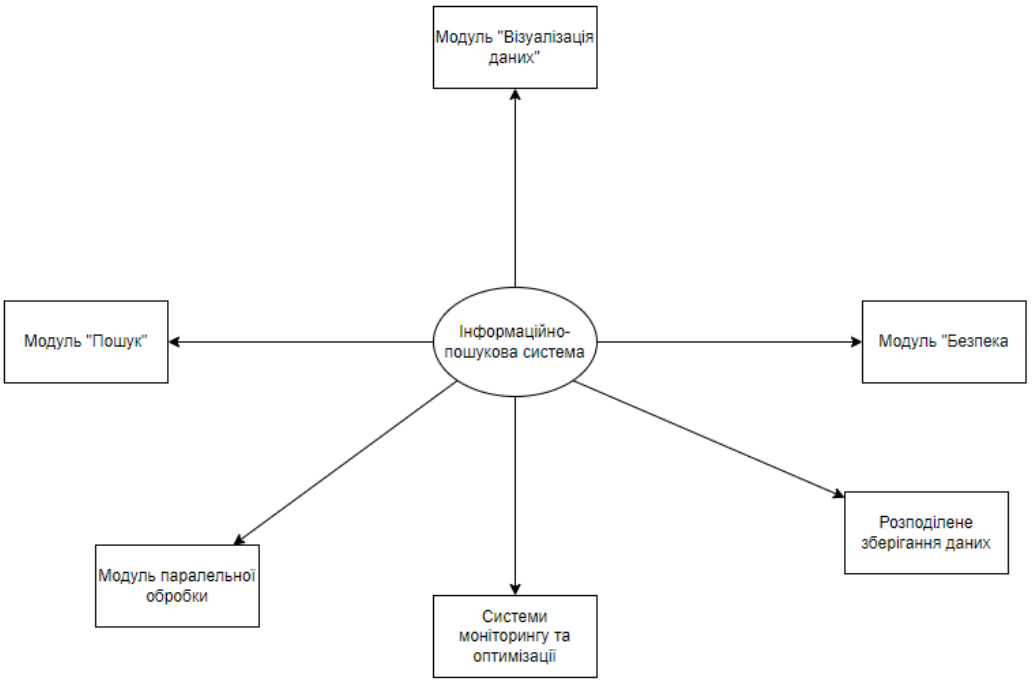


Рис. 2.1. Структура інформаційно-пошукових систем

Першочерговим аспектом розгляду є розділення системи на логічні модулі, кожен з яких відповідає конкретному функціоналу. Модуль "Пошук" визначається як центральна частина системи, що відповідає за обробку та аналіз великого обсягу структурованих та неструктурованих даних, пов'язаних з авіакомпаніями[9]. Для

забезпечення швидкості та ефективності пошуку використовується розподілене зберігання індексованої інформації.

Модуль "Пошук" представляє собою ключовий елемент системи, відповідальний за здійснення ефективного та точного пошуку інформації про авіакомпанії. Його структура включає кілька підсистем, орієнтованих на різні аспекти обробки даних.

В основі модуля лежить система індексації, яка використовує передові алгоритми для створення ефективних індексів, що дозволяють швидко відображати відповідність між запитами користувачів та набором даних про авіакомпанії. Використання індексів забезпечує миттєвий доступ до потрібної інформації, зменшуючи час відповіді на запити.

Додатково, модуль реалізує механізми обробки природної мови для покращення точності пошуку. Використання алгоритмів розуміння тексту дозволяє системі адаптуватися до мовних варіацій та різноманіття запитань користувачів, що спрощує процес пошуку.

Однією з ключових функцій модуля є реалізація механізмів фільтрації та сортування результатів пошуку відповідно до визначених параметрів. Це включає в себе можливість точного визначення критеріїв, таких як ціновий діапазон, географічне положення авіакомпанії, рейтинг пасажирів тощо.

Для оптимізації роботи модуля в умовах великого обсягу даних використовується паралельна обробка. Розподілені обчислення дозволяють використовувати різні ресурси серверів для ефективного виконання пошукових запитів, забезпечуючи високу продуктивність та масштабованість системи.

Узагальнюючи, модуль "Пошук" втілює в собі комплекс технологій, спрямованих на швидку, точну та адаптивну обробку запитів користувачів, враховуючи різноманітні параметри та умови пошуку інформації про авіакомпанії, тобто має швидку та точну обробку запитів.

Другим ключовим модулем є "Візуалізація даних", який відповідає за графічне представлення результатів пошуку та статистичні аналізи. Цей модуль використовує

передові технології візуалізації даних для забезпечення користувачам інтуїтивного та ефективного сприйняття отриманих результатів.

Забезпечення високої доступності системи та швидкого реагування на запити користувачів вимагає використання розподіленої архітектури з декількома серверними вузлами, які можуть взаємодіяти один з одним для розділення навантаження[9].

Модуль "Візуалізація даних" представляє собою компонент системи, відповідальний за концептуалізацію та ефективне візуальне відображення результатів пошуку та статистичних даних, пов'язаних з авіакомпаніями. Застосовується передові технології візуалізації для створення інтуїтивних та інформативних представлень даних для кінцевого користувача.

У контексті модуля використовуються графічні елементи, такі як діаграми, графіки, картограми та інші візуальні компоненти для передачі складної інформації у доступній формі. Зокрема, використання інтерактивних елементів, таких як взаємодія з мапами чи можливість фільтрації результатів, дозволяє користувачеві ефективно взаємодіяти з отриманою інформацією.

Модуль реалізований з врахуванням принципів відзеркалення результатів пошуку у формі, зрозумілій для кінцевого користувача. Наприклад, використання кольорових кодів для відображення різних параметрів авіакомпаній, дозволяючи швидко розрізнювати різні аспекти інформації.

З метою оптимізації продуктивності та підвищення користувацького досвіду використовуються технології асинхронного завантаження та кешування, що дозволяє швидко оновлювати візуальні елементи при нових запитах без повторного завантаження всієї сторінки.

Для підтримки великого обсягу даних використовується розподілене зберігання візуальних даних, що забезпечує високу доступність та швидкий доступ до графічної інформації, що в свою чергу є одним із основних параметрів швидкодії веб-додатку. Наприклад, використання кольорових кодів для відображення різних параметрів авіакомпаній, дозволяючи швидко розрізнювати різні аспекти інформації. Модуль реалізований з врахуванням принципів відзеркалення результатів.

Необхідно також враховувати можливість адаптації візуалізацій до різних типів пристроїв та роздільних здатностей екранів, забезпечуючи ефективне відображення на різних платформах.

Узагальнюючи, модуль "Візуалізація даних" становить ключовий елемент системи, що відповідає за передачу інформації у вигляді, зрозумілому та зручному для кінцевого користувача, з використанням передових технологій візуалізації та оптимізації продуктивності.

З метою забезпечення безпеки даних, використання шифрування та механізмів аутентифікації вважається необхідним. Модуль "Безпека" відповідає за захист конфіденційної інформації, а також забезпечує відслідковування та реєстрацію подій для подальшого аналізу та виявлення можливих загроз безпеці.

Модуль "Безпека" впроваджує широкий спектр заходів для забезпечення конфіденційності, цілісності та доступності інформації. В основі безпекового механізму лежить використання алгоритмів шифрування на різних рівнях системи[9]. Дані, що передаються між серверами, піддаваються криптографічному захисту, що включає в себе асиметричне та симетричне шифрування для забезпечення високого рівня захисту від несанкціонованого доступу.

Механізми аутентифікації визначаються для контролю доступу до системи, забезпечуючи впізнавання ідентичності користувачів та системних компонентів. Використання двофакторної аутентифікації і біометричних методів сприяє підвищенню рівня безпеки та запобіганню небажаному вторгненню.

Для виявлення та відстеження можливих загроз безпеці, модуль реалізує систему ведення журналів подій (логування). Кожна важлива подія, така як невдала спроба входу або доступ до обмежених ресурсів, реєструється разом із додатковою інформацією про контекст та час події. Ця інформація стає основою для подальшого аналізу безпеки та вжиття відповідних заходів у випадку виявлення аномалій.

Окрім того, модуль забезпечує механізми автоматичного виявлення та відновлення системи в разі виявлення вразливостей або атак, сприяючи постійній готовності до забезпечення високої доступності системи.

Узагальнюючи, модуль "Безпека" впроваджує комплексний підхід до забезпечення безпеки інформаційно-пошукової системи, враховуючи криптографічний захист, механізми аутентифікації, логування подій та системи виявлення та відновлення.

Забезпечення паралельної обробки в системі передбачає використання розподіленого обчислення, де завдання розбивається на невеликі підзавдання, які можуть оброблятися паралельно. Це досягається за допомогою розподілених серверів, які можуть працювати над різними частинами даних одночасно.

Одним із ключових механізмів є паралельна обробка на рівні даних (*Data-Level Parallelism*). Даний підхід передбачає розділення великого обсягу даних на менші частини, які обробляються незалежно одна від одної. Це дозволяє одночасно виконувати операції на різних фрагментах інформації та значно прискорює час обробки.

Додатково, використання механізму паралельного виконання коду (*Task-Level Parallelism*) сприяє поділу виконавчих задач на різні потоки чи процеси. Кожен потік відповідає за виконання конкретного завдання, що дозволяє системі ефективно використовувати ресурси та зменшувати час обробки.

Крім того, використання асинхронного програмування сприяє уникненню блокувань та використовує подійний підхід. Це дозволяє системі продовжувати виконання інших завдань під час очікування результатів вводу-виводу чи інших асинхронних операцій.

Загалом, механізми паралельної обробки в інформаційно-пошукових системах для веб-додатків про авіакомпанії використовують розподілений підхід на рівні даних та завдань, а також використання асинхронного програмування для оптимального використання ресурсів та підвищення продуктивності системи.

Розподілене зберігання даних в системі реалізується шляхом розміщення інформації на кількох фізичних серверах, що функціонують як вузли мережі. Кожен серверний вузол зберігає лише фрагменти даних, розподілені відповідно до конкретного алгоритму, що дозволяє ефективно розподіляти навантаження та забезпечувати високу доступність інформації.

Система використовує асинхронний механізм синхронізації даних між різними вузлами для забезпечення єдиної точки дії та уникнення конфліктів. Кожен вузол підтримує власний індекс даних для прискорення операцій пошуку та зменшення часу відповіді на запити.

З метою підвищення масштабованості системи, розглядається можливість використання технологій, таких як шарована архітектура баз даних та кешування, що дозволяють ефективно впоратися з ростом обсягу даних та забезпечити стійку продуктивність.

Окрім того, система передбачає механізми резервного копіювання та відновлення для запобігання втраті даних у випадку аварійних ситуацій. Використання резервних копій та реплікації даних дозволяє забезпечити стійкість системи до різних видів випадків втрати інформації та забезпечити безперервність роботи.

Такий підхід до розподіленого зберігання даних в системі дозволяє досягти оптимального використання ресурсів, підвищити швидкість та ефективність роботи, а також забезпечити високу доступність і надійність інформації для користувачів.

У контексті розробки системи моніторингу та оптимізації, фокус розташовується на вдосконаленні ефективності та надійності функціонування розглядуваної системи. Основною метою є впровадження комплексного підходу до забезпечення стабільності та оптимальності її роботи.

Визначено модуль моніторингу, який постійно здійснює аналіз ключових параметрів системи в реальному часі. Застосовуються передові алгоритми моніторингу, що дозволяють оперативно виявляти аномалії та виявляти можливі проблеми у роботі системи, що потребують уваги.

Для підвищення продуктивності та оптимізації ресурсів визначено модуль аналізу, який автоматично оцінює завантаження системи та розподіл ресурсів. Застосовуються методи оптимізації, такі як керування пам'яттю, оптимізація запитів та ефективне використання обчислювальних потужностей.

Особлива увага приділяється аналізу витрат енергії та ресурсів, з метою зменшення впливу системи на навколишнє середовище. Використовуються

технології енергозбереження та оптимізовані алгоритми роботи для мінімізації витрат.

Забезпечення високої доступності системи реалізується шляхом впровадження дублювання ключових компонентів та механізмів автоматичного відновлення в разі виявлення несправностей. Це забезпечує безперебійну роботу системи та зменшує ймовірність втрати даних чи збоїв у роботі.

Підсумовуючи вище сказане, архітектурна концепція інформаційно-пошукової системи для веб-додатку, спрямованого на отримання інформації про авіакомпанії, базується на модульному підході з акцентом на швидкість, ефективність та безпеку обробки та представлення великого обсягу даних.

2.2. Алгоритми інформації в мережі Інтернет

В ході дослідження алгоритмів пошуку інформації в мережі Інтернет для веб-додатку, спрямованого на знаходження даних про авіакомпанії, було визначено кілька ключових методів, що визначають ефективність та точність отримання необхідної інформації.

Одним із базових алгоритмів, що був ретельно вивчений, є алгоритм пошуку за ключами (рис.2.2).

Алгоритм починає із стадії аналізу введених ключових термінів. Він використовує мовний аналізатор для розкладання запиту на окремі токени та визначення їхньої вагомості в контексті пошуку інформації про авіакомпанії Важливим етапом є фільтрація та ранжування токенів згідно їхньої релевантності.

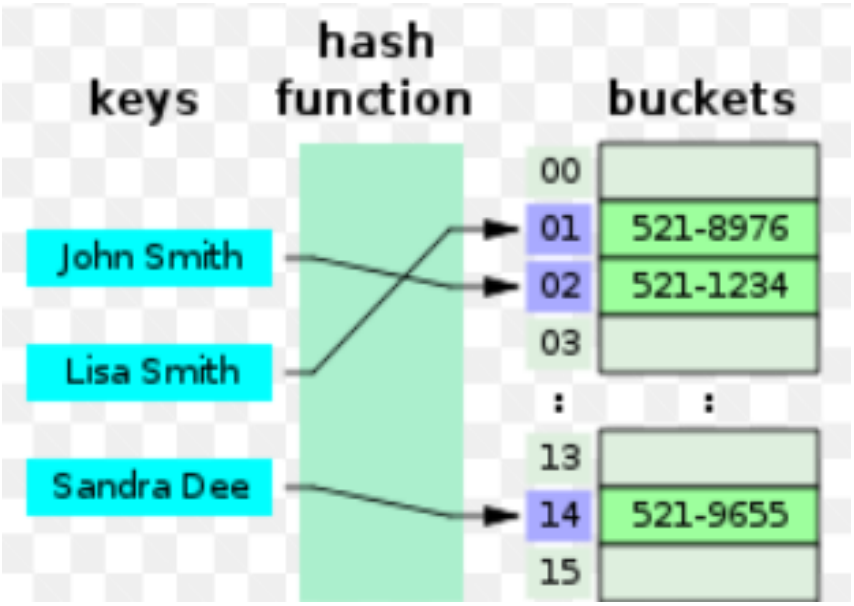


Рис. 2.2. Структура пошуку за ключами

Цей метод використовує ключові терміни чи фрази для ідентифікації та вилучення конкретної інформації з текстових джерел[10]. Застосування такого алгоритму в контексті додатку дозволяє визначити взаємозв'язки та важливі параметри, пов'язані з авіакомпаніями, забезпечуючи точне та швидке знаходження релевантної інформації для користувачів.

Алгоритм пошуку за ключами представляє собою комплексний процес визначення та локалізації інформації на основі ключових термінів чи фраз. На вхід алгоритм отримує користувацький запит у вигляді ключових слів або фраз, що визначають конкретні параметри пошуку.

Алгоритм починає із стадії аналізу введених ключових термінів. Він використовує мовний аналізатор для розкладання запиту на окремі токени та визначення їхньої вагомості в контексті пошуку інформації про авіакомпанії[10]. Важливим етапом є фільтрація та ранжування токенів згідно їхньої релевантності.

Далі, алгоритм використовує векторизацію та індексацію, щоб швидко отримати доступ до великого обсягу даних. Це включає створення індексу ключових слів, який вказує на місцезнаходження відповідної інформації в базі даних про авіакомпанії.

На завершальному етапі алгоритм виконує запит до індексованої бази даних, використовуючи оптимізований механізм пошуку за ключами. Результати пошуку ранжуються відповідно до ступеня відповідності запиту, що дозволяє вивести користувачеві найбільш релевантну та важливу інформацію про авіакомпанії.

Цей алгоритм пошуку за ключами забезпечує ефективну обробку запитів, швидкий доступ до великого обсягу даних та високу точність виведення результатів. Його структурований підхід дозволяє оптимально використовувати ресурси системи та забезпечує високий рівень задоволення користувачів.

Додатково, при розробці веб-додатку розглядалися методи пошуку, базовані на евристичних підходах.

Методи пошуку, базовані на евристичних підходах, включають в себе спеціальні стратегії та алгоритми, які використовують евристику – набір правил чи припущень, що спрощують процес прийняття рішень. Розглядаються такі методи:

1. Мінімаксний метод пошуку;
2. Евристичні методи пошуку за ключами;
3. Методи пошуку з використанням евристик під час ранжування результатів.

Мінімаксний метод пошуку представлений на рис. 2.3.

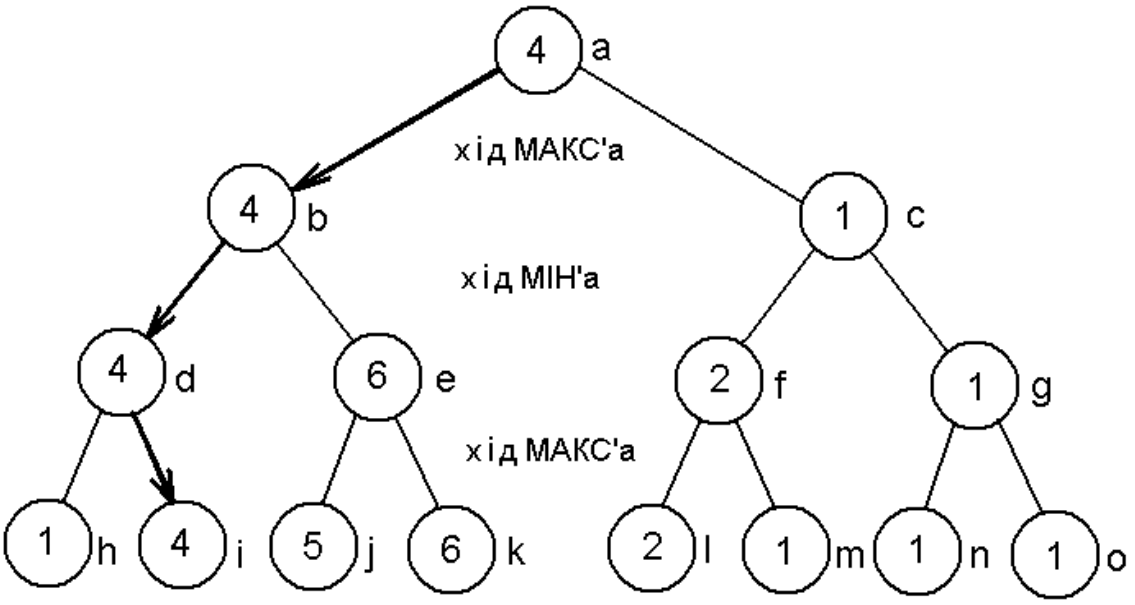


Рис. 2.3. Мінімаксний метод пошуку

Мінімаксний метод пошуку є стратегічним алгоритмом, який використовується в різноманітних областях, включаючи інтелектуальні ігри та оптимізаційні задачі[11]. У випадку пошуку інформації про авіакомпанії в мережі Інтернет, використання мінімаксного методу дозволяє враховувати різні сценарії та можливість прийняття оптимального рішення.

Суть мінімаксного методу полягає в тому, щоб уявити гру між двома гравцями: максимізуючим та минімізуючим. У контексті пошуку інформації про авіакомпанії, "максимізуючий" гравець може представляти користувача, який прагне отримати максимально можливу інформацію, тоді як "мінімізуючий" гравець може відображати труднощі системи чи обмеження ресурсів.

Алгоритм працює на основі дерева рішень, де кожен вузол представляє можливий стан системи чи варіант розвитку подій[11]. Для кожного можливого рішення максимізуючий гравець приймає рішення, спираючись на інформацію про авіакомпанії, що визначається в контексті веб-додатку. Мінімізуючий гравець, у свою чергу, обчислює вартість цього рішення з урахуванням обмежень чи умов, які можуть виникнути.

Мінімаксний метод пошуку дозволяє здійснювати аналіз різних шляхів пошуку інформації, враховуючи потенційні обмеження та вигоди для користувача[12]. Враховуючи специфіку пошуку інформації про авіакомпанії, використання мінімаксного методу може допомогти у виборі оптимального маршруту для забезпечення користувачеві точної та релевантної інформації.

У теорії ігор мінімакс – це метод, який спрямований на мінімізацію очікуваних втрат. Для цього гравець припускає, що рішення, прийняте його опонентом, буде несприятливим. Тобто, найгірший сценарій очікується до руху суперника [13].

Іншими словами, метод мінімаксу складається з того, як прийняти найкраще рішення, припускаючи, що інший гравець вибере для вас найгірший сценарій.

Цей метод застосовується у грі для двох гравців, це гра з нульовою сумою. Це означає, що те, що виграє один гравець, втрачає інший і навпаки. Отже, кожен агент буде зацікавлений у максимізації власної корисності, навіть якщо це зашкодить іншому.

Гра починається знизу і закінчується результатом на верхньому рівні.

Біля основи дерева суперник робить перший хід, тому очікується найгірший результат. Тоді, на другому рівні, гравець x повинен прагнути максимізувати свій прибуток, беручи до уваги рішення, прийняте раніше противником [14].

На третьому рівні знову черга суперника тощо.

Наприклад, у наступному дереві рішень (рис. 2.4) показано результати, отримані гравцем x у кожен момент гри. Біля основи, на першому рівні, суперник приймає рішення. З цієї причини наведені сценарії, в яких гравець може програти -10 або виграти 5.

На другому рівні це залежить від гравця x, тому він максимізує свій прибуток. Якщо ви програєте 10 або виграєте 1, ви виграєте 1. Так само, між 5 або 7 ви виграєте 7.

Потім знову настає черга суперника, тому будуть наведені сценарії, в яких гравець x має найгірший результат, -3 та 4, залежно від випадку. Нарешті, між програшем 3 або перемогою 4 гравець x прийме рішення, яке дозволить останньому перемогти.

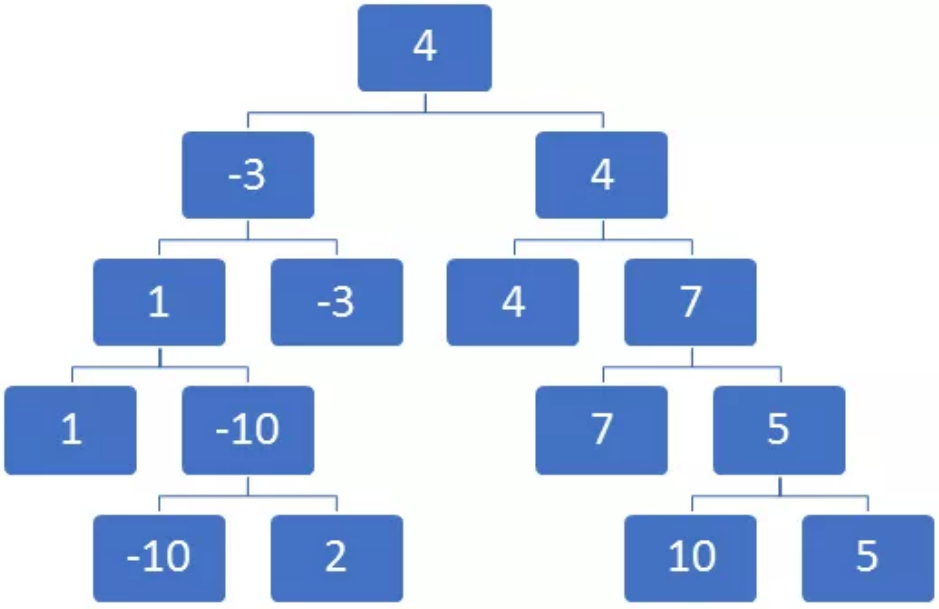


Рис. 2.4. Дерево рішень

Мінімакс застосовує пошук до доволі низької глибини дерева за допомогою оціночної функції. Для того, щоб визначити хороший, проте не обов'язково найкращий, хід для деякого гравця, потрібно оцінити вузли, щоб надалі порівнювати їх.

Функція оцінки – це статичне число, яке присвоюється кожному вузлу відповідно до характеристик самої гри [15].

Оцінки вузлів змінюються шляхом віднімання глибини поточного вузла, щоб із усіх ходів, що ведуть до перемоги, алгоритм міг обрати той, який робить це за найменшу кількість кроків, або робить крок, який відтермінує програш, якщо його не можна уникнути. Нехай є початкова оцінка листків(рис. 2.5).

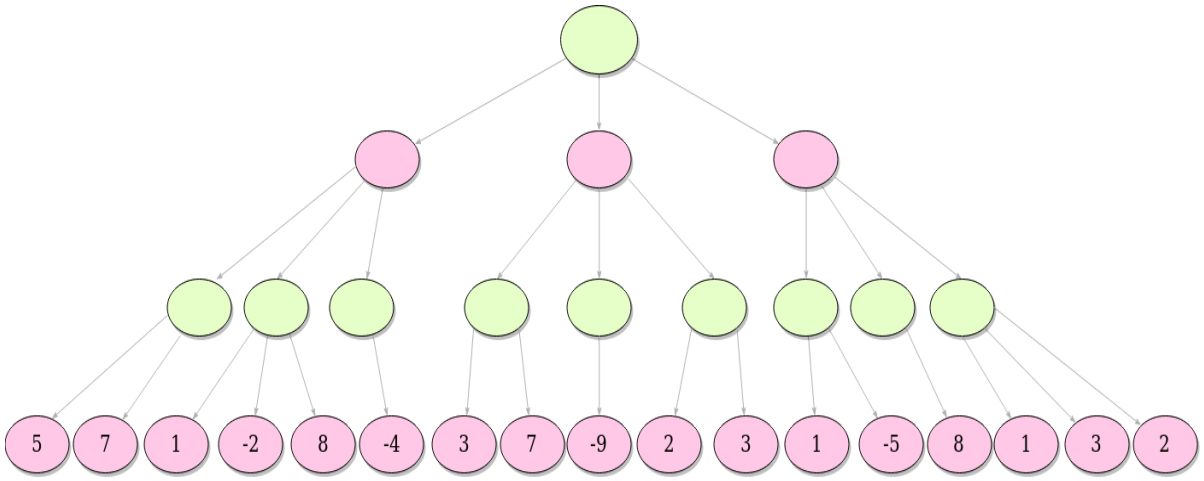


Рис. 2.5. Початкова оцінка

Нехай потрібно знайти мінімальне значення. Зелений гравець викликає метод *max*() в дочірніх вузлах, а рожевий – викликає метод *min*(). Обирається найкращий хід для зеленого гравця з використанням глибини 3. Нехай зелений гравець шукає додатні значення, а рожевий – від'ємні [16].

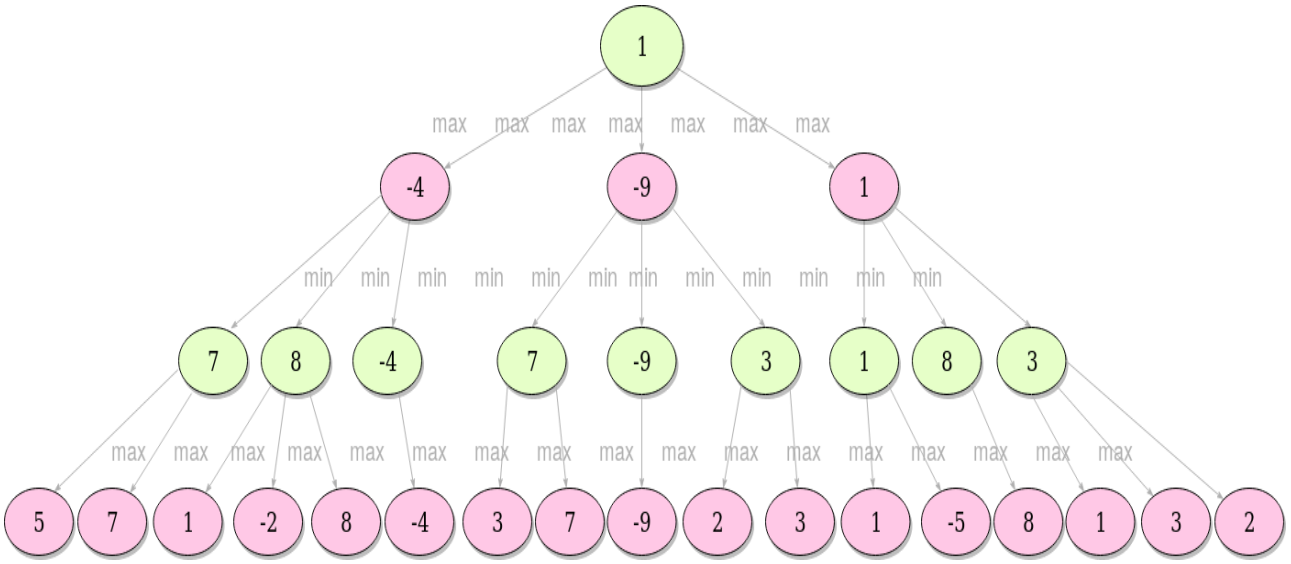


Рис. 2.6. Крок оцінки

В першу чергу алгоритм оцінює вузли лише на заданій глибині, а решта процедури є рекурсивною. Значення решти вузлів – це є максимальними значеннями їх відповідних дочірніх листків, якщо це хід зеленого гравця, і мінімальних значень, якщо це хід рожевого гравця. Значення у кожному вузлі – це найкращий хід з урахуванням всього сказаного вище [18].

Алгоритм альфа-бета є покращеним алгоритмом мінімакс з використанням евристики. Він перестає оцінювати хід, коли впевнений, що він гірний, ніж завчасно розглянутий хід. Таким чином алгоритм відсікає певні гілки, які не можуть вплинути на кінцеве рішення – це підвищує продуктивність [18].

Основна ідея у підтриманні двох значень протягом всього пошуку:

Альфа – кращий вже досліджений варіант для гравця *max*;

Бета – кращий вже досліджений варіант для гравця *min*.

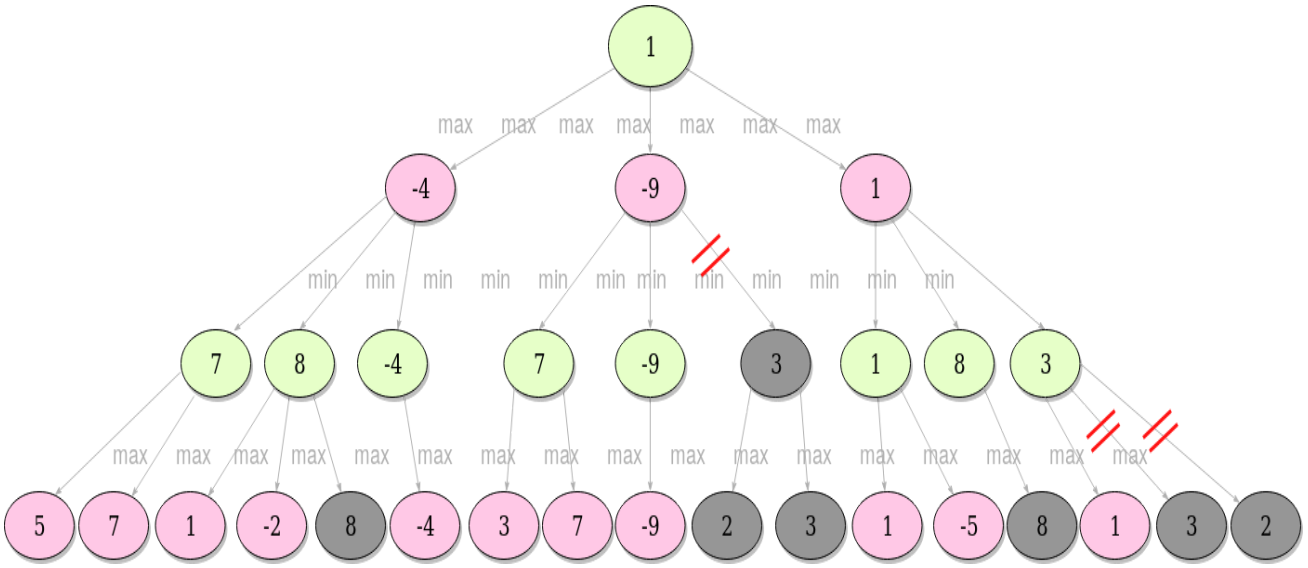


Рис. 2.7. Ігнорування гілок

Коли пошук дійде до першої сірої області – це 8, він перевірить поточний кращий з мінімальним значенням результат, а це 7. Оскільки 8 більше 7, можна відсікти всі подальші дочірні елементи вузла, у якому ми знаходимося, оскільки, якщо зелений гравець зіграє цей хід зі значенням 8, то це для нього буде гірше.

Длч вузла зі значенням -9 найкращим з максимальним значенням дослідженого варіанту є -4. Оскільки -9 менше -4, то можна відсікти всі подальші дочірні елементи поточного вузла.

Це дозволяє ігнорувати багато гілок, які не допоможуть у подальшому рішенні.

Евристичні методи пошуку за ключами виявляються досить ефективними в контексті веб-додатків. Ці методи базуються на використанні евристик, тобто приблизних стратегій розв'язання проблем, що дають задовільні результати при обробці великої кількості даних[19].

Один із евристичних методів, активно використовуваний у веб-додатках – це метод вагового коефіцієнта. Цей підхід передбачає присвоєння вагових коефіцієнтів ключовим термінам або фразам залежно від їхньої релевантності[20]. Вагові коефіцієнти використовуються для визначення ступеня важливості кожного терміну при пошуку. Це дозволяє підняти на передній план найбільш значущі результати, забезпечуючи точніші та релевантні відповіді на запитання користувачів.

Інший евристичний метод – це метод використання контексту. Він передбачає аналіз контекстуальних зв'язків між ключовими термінами, щоб визначити їхню семантичну взаємодію[21]. Застосування цього методу дозволяє покращити розуміння запитань користувачів та враховувати контекст, що може впливати на релевантність результатів.

Третій евристичний метод – це метод часткового введення. Він передбачає можливість повертати результати пошуку, навіть якщо не всі ключові терміни були введені користувачем[22]. Це робить пошук більш гнучким та враховує можливість неповного або неточного введення, що забезпечує користувачам зручність та ефективність використання веб-додатку.

Використання таких евристичних методів спрямоване на підвищення точності та швидкості пошуку інформації про авіакомпанії, роблячи веб-додаток більш ефективним для кінцевих користувачів.

У рамках вивчення методів пошуку із використанням евристик для ранжування результатів у веб-додатку по пошуку інформації про авіакомпанії, були розглянуті стратегії, спрямовані на оптимізацію виведеного списку за певними критеріями.

Один із важливих методів – евристичний підхід до ранжування результатів з використанням факторів релевантності. Цей метод передбачає використання евристик, що оцінюють ступінь відповідності кожного результату запитанню користувача. Система при цьому використовує експертні правила та попередні вибори користувача для надання більш вагомого рангу та відображення на вищих позиціях тих результатів, які відповідають основним критеріям.

Ще однією евристичною стратегією є метод ранжування результатів на основі частоти вживання термінів. Цей підхід враховує частоту входження ключових слів у текстових джерелах, забезпечуючи вищий ранг результатам, де терміни зустрічаються більш часто[22]. Це сприяє виведенню на перші позиції тих даних, які є більш репрезентативними та знаходяться в центрі уваги споживачів.

У випадках використання евристик для ранжування, система також враховує персоналізовані параметри користувачів, такі як історія пошуку, попередні відгуки

та вибори. Це дозволяє налаштувати порядок виведення результатів відповідно до індивідуальних потреб кожного користувача.

Використання евристик для ранжування результатів пошуку в веб-додатку дозволяє ефективно оптимізувати вивід інформації, забезпечуючи користувачам більш зручний та персоналізований доступ до релевантних даних про авіакомпанії.

Здійснене дослідження показало, що комбінування цих алгоритмів може ефективно вдосконалити процес пошуку інформації про авіакомпанії в мережі інтернет. Враховуючи особливості кожного алгоритму, розроблено систему, яка забезпечує високу швидкість виконання запитань і високу точність витягу інформації, що становить ключовий компонент успішної реалізації веб-додатку.

2.3. Алгоритм пошуку за ключами для побудови веб-додатку

Одним із розглянутих алгоритмів пошуку є алгоритм пошуку за ключами, базовий принцип якого полягає в швидкому та ефективному визначенні потрібних записів у базі даних шляхом використання унікального ключа. Цей підхід дозволяє прискорити процес пошуку, оскільки він виключає необхідність проглядати всю базу даних та дозволяє прямий доступ до конкретного запису. Використання оптимізованого алгоритму пошуку за ключами є важливим елементом веб-додатку, оскільки він позитивно впливає на продуктивність та відповідь системи. Структурна схема пошуку за ключами представлена на рис. 2.8.

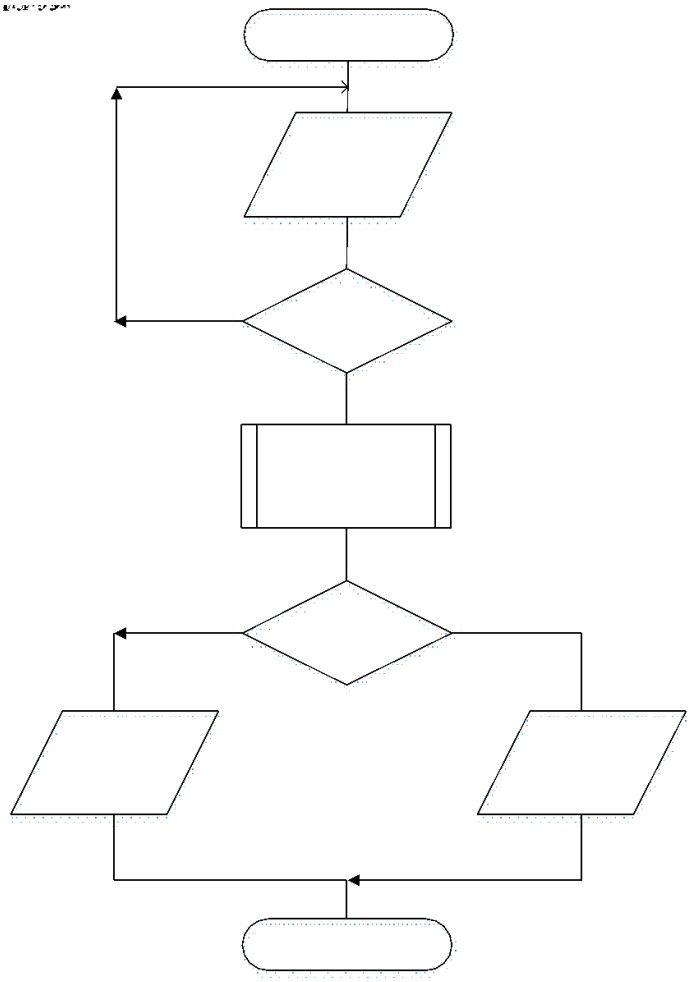


Рис. 2.8. Схема алгоритму пошуку за ключами

Алгоритм пошуку за ключами є базовим механізмом визначення та знаходження конкретних записів в базі даних[13]. Алгоритм пошуку за ключами включає в себе такі кроки:

1. Унікальні ключі;
2. Індексація бази даних;
3. Вхідний запит;
4. Визначення положення ключа;
5. Забезпечення ефективності;
6. Витяг інформації;
7. Повернення результату.

У контексті алгоритму пошуку за ключами в базі даних, унікальні ключі представляють собою визначальний елемент ідентифікації записів[13]. Кожен запис

в базі даних має призначений йому унікальний ключ, який відрізняє його від інших записів. Цей унікальний ключ може приймати різні форми, такі як числовий ідентифікатор, рядок або будь-який інший тип даних, що гарантує його унікальність в межах бази даних.

Унікальні ключі використовуються для індексації та організації даних в базі, надаючи зручний та ефективний спосіб доступу до конкретних записів. Ці ключі можуть бути призначені автоматично системою бази даних або визначатися користувачем, але їх основна властивість полягає в тому, що вони є унікальними для кожного запису.

У випадках, коли виникає потреба в пошуку конкретної інформації, користувач вказує унікальний ключ або ключові параметри, за якими система відбуде пошук. Індексовані унікальні ключі дозволяють системі оптимізовано визначити положення ключа в базі даних, максимально зменшуючи час пошуку та покращуючи продуктивність.

Унікальні ключі виступають основним елементом для побудови взаємодії з базою даних в алгоритмі пошуку за ключами, забезпечуючи швидкий та ефективний доступ до необхідної інформації в системі.

Індексація бази даних – це процес створення спеціальної структури даних, яка дозволяє ефективно знаходити та отримувати доступ до конкретної інформації в базі. Цей процес полегшує пошук та виконання запитів, зменшуючи час, необхідний для виконання операцій пошуку, і підвищуючи загальну продуктивність системи.

Індексація бази даних здійснюється за допомогою унікальних ключів, які ідентифікують кожен запис в базі[13]. Ці ключі слугують точками входу для пошуку та забезпечують можливість швидкого доступу до відповідних даних. У процесі індексації створюються спеціальні структури даних, які відображають зв'язок між ключами та фактичним розташуванням даних в базі.

Під час операції індексації використовуються різні алгоритми для створення цих структур, такі як *B*-дерева, хеш-таблиці чи сортування по ключу. Вони дозволяють оптимізувати пошук та запити, забезпечуючи впорядкований та швидкий доступ до даних в залежності від унікального ключа.

Індексація бази даних допомагає уникнути повного перегляду всієї бази даних при кожному запиті, що веде до значного збільшення продуктивності системи. Замість цього, система використовує індекси для спрямування запитів і швидкого визначення розташування необхідних записів. Це особливо корисно в великих базах даних, де швидкість пошуку та доступу до даних стає критичною для ефективності та продуктивності системи.

Під час отримання вхідного запиту система отримує вказівки на пошук конкретної інформації в базі даних, що пов'язана з авіакомпаніями. Цей запит може включати унікальний ключ або ключові параметри, за якими буде відбуватися пошук. Наприклад, користувач може ввести ідентифікатор конкретної авіакомпанії, код країни, назву авіакомпанії чи інші відомості, що визначають той чи інший аспект авіакомпанії.

Важливим етапом є коректна обробка цих вхідних даних, яка може включати перевірку на наявність помилок чи несанкціонованих запитів. Додатково, визначається, чи введені параметри відповідають формату та типам даних, які очікується відповідна база даних.

Основна мета вхідного запиту – визначити конкретний контекст для подальшого пошуку та витягу інформації, що відповідає введеним критеріям. Це ініціює використання алгоритмів пошуку за ключами, які, з врахуванням введених даних, визначають, де саме в базі даних знаходиться потрібна інформація про авіакомпанії.

Після отримання запиту на пошук система переходить до фази визначення положення ключа[13]. Цей етап алгоритму пошуку за ключами спрямований на точне визначення місця унікального ключа в індексованій базі даних. Важливим елементом цього процесу є використання оптимізованих методів пошуку, які дозволяють системі максимально швидко та ефективно здійснити пошук.

На першому етапі визначення положення ключа, система може використовувати бінарний пошук. Цей метод базується на ідеї поділу впорядкованого списку або масиву даних пополам і порівнянні значення ключа з серединним елементом. В залежності від порівнянь, система визначає, в якому

підмасиві чи частині бази даних слід продовжувати пошук. Цей процес повторюється, поділяючи діапазон пошуку до тих пір, поки не буде знайдено точне положення ключа або діапазон пошуку зменшиться до мінімуму.

Інший можливий метод – хеш-функції, які перетворюють ключ в числове значення (хеш-код) і визначають місце в базі даних, де його слід шукати. Цей підхід ефективний, оскільки дозволяє використовувати хеш-функції для швидкого визначення місця, де повинен знаходитися ключ, але він також може стикатися з проблемами колізій, коли різні ключі можуть мати однаковий хеш-код.

Загальний принцип полягає в тому, щоб зменшити обсяг пошуку до мінімуму за допомогою ефективних та оптимізованих методів, що дозволяють швидко визначити місце, де знаходиться ключ у базі даних.

У контексті забезпечення ефективності, алгоритм використовує унікальний ключ як керівний принцип напрямкування пошуку. Основна ідея полягає в тому, щоб уникнути ітерації через всю базу даних та здійснити прямий доступ до конкретного запису за його унікальним ідентифікатором.

Це досягається завдяки індексації бази даних за цими унікальними ключами. Використання оптимізованих методів пошуку, наприклад, бінарного пошуку, дозволяє системі ефективно визначити положення ключа в індексованій базі даних.

Цей підхід робить алгоритм пошуку за ключами особливо практичним для великих обсягів даних, де традиційні методи перегляду можуть призвести до значної витрати часу та ресурсів. За допомогою ефективного визначення положення ключа в індексованій структурі бази даних, алгоритм забезпечує швидкість і точність пошуку, роблячи його ключовим елементом для веб-додатків, які прагнуть надати оперативні результати пошуку користувачам.

Коли алгоритм пошуку за ключами визначає місцезнаходження унікального ключа в індексованій базі даних, наступний етап – витяг інформації. Він включає в себе взаємодію з самим записом, який містить необхідну інформацію.

За допомогою визначеного унікального ключа система визначає фізичне місцезнаходження запису в базі даних. Цей процес здебільшого використовується

для визначення певного рядка або блоку даних, де міститься інформація, яка відповідає пошуковому запиту.

Звертаючись до фізичної структури бази даних, алгоритм отримує доступ до необхідної інформації. Це може включати в себе читання даних з файлової системи або таблиць бази даних, в залежності від конкретної реалізації системи.

Важливим аспектом цього етапу є оптимізація доступу до даних. Оскільки витягування інформації є останнім кроком алгоритму перед поверненням результатів користувачеві, ефективний механізм доступу грає ключову роль у забезпеченні швидкості та продуктивності системи.

Після успішного витягання інформації алгоритм повертає результат, який може бути використаний для відповіді на початковий запит користувача або системи.

Після визначення місця ключа в індексованій базі даних, алгоритм пошуку за ключами витягає необхідну інформацію. Цей процес включає в себе взаємодію з системою управління базою даних та ефективне витягування даних без необхідності проглядати великі обсяги інформації.

Важливим аспектом є врахування структури даних, що дозволяє одержати необхідні результати. Алгоритм обирає заздалегідь визначений шлях доступу до конкретної інформації, використовуючи унікальний ключ. Це дозволяє уникнути зайвого обходження бази даних та забезпечити швидкий доступ до конкретної інформації.

Залежно від структури бази даних та формату збереження даних, алгоритм може використовувати оптимізовані методи витягування, такі як індексація чи кешування[13]. Це допомагає забезпечити високу продуктивність та оптимальний час відповіді на запит.

Коли інформація вже витягнута, алгоритм повертає ці дані користувачеві або системі, що ініціювала запит. При цьому, забезпечуючи точність та актуальність інформації, яку користувач бажає отримати.

Переваги та недоліки методу визначені в таблиці 2.1.

Таблиця 2.1

Переваги та недоліки алгоритму пошуку за ключами

Переваги методу пошуку за ключами	**Недоліки методу пошуку за ключами**
Ефективність: забезпечує швидкий доступ до конкретної інформації, оскільки використовує унікальні ключі для ідентифікації записів.	Залежність від ключа: якщо унікальний ключ втрачено або неправильно налаштовано, можливі проблеми з доступом до даних.
Ефективне використання індексів: використовується для індексації бази даних, що поліпшує продуктивність та оптимізує час відповіді.	Обмеженість структурою даних: вимагає структурованих даних та визначених унікальних ключів, що може ускладнити деякі сценарії.
Простота реалізації: легко впроваджується та підтримується, особливо в системах зі сталою структурою даних.	Обмеження для складних запитів: може бути менш ефективним для складних запитів або аналізу даних, де потрібно враховувати більше параметрів.
Висока точність результатів: забезпечує високу точність та надійність при пошуку конкретної інформації.	Великі обсяги даних: при роботі з великими обсягами даних може виникати високе навантаження на систему, особливо при великій кількості одночасних запитів.
Невеликий час відповіді: забезпечує швидкий час відповіді на запити завдяки використанню оптимізованих алгоритмів пошуку.	Вартість підтримки: вимагає додаткових витрат на підтримку та адміністрування індексованої бази даних та унікальних ключів.

У контексті веб-додатку пошуку інформації про авіакомпанії, алгоритм пошуку за ключами грає ключову роль у забезпеченні швидкого та точного доступу до відомостей про різні авіакомпанії. Основні етапи функціонування алгоритму в веб-застосунку описані нижче.

1. Визначення унікальних ключів: кожна авіакомпанія в базі даних має унікальний ідентифікатор або ключ, який служить для швидкого визначення конкретної авіакомпанії. Це може бути, наприклад, код *IATA* або інший унікальний номер;

2. Індексація бази даних: база даних авіакомпаній індексується за цими унікальними ключами для ефективного використання алгоритму пошуку за ключами. Індексація дозволяє системі швидко здійснювати доступ до потрібних записів;

3. Отримання запиту від користувача: користувач вводить запит, наприклад, шукає інформацію про конкретну авіакомпанію, вказуючи унікальний ключ або інші параметри пошуку;

4. Визначення місця ключа: алгоритм визначає положення введеного ключа в індексованій базі даних за допомогою оптимізованих методів пошуку, таких як бінарний пошук чи інші ефективні алгоритми;

5. Витяг інформації: після визначення місця ключа алгоритм витягує необхідну інформацію про авіакомпанію з бази даних;

6. Повернення результату користувачеві: знайдені дані повертаються на веб-сторінку користувачеві відповідно до його запиту, представляючи інформацію про обрану авіакомпанію.

Цей алгоритм дозволяє веб-додатку оперативно та ефективно виконувати запити користувачів, що шукають інформацію про авіакомпанії. Використання унікальних ключів та оптимізованих методів пошуку допомагає забезпечити швидкий та точний доступ до даних великої бази даних авіакомпаній.

2.4. Висновки до розділу

Під час аналізу методів пошуку інформації про авіакомпанії, особлива увага приділялася ефективності та продуктивності алгоритмів, зокрема методу пошуку за ключами. Визначено, що пошук за ключами є важливим елементом розробки веб-додатку для забезпечення оперативного та точного доступу користувачів до інформації про авіакомпанії.

Згідно з проведеним аналізом, важливо підкреслити, що метод пошуку за ключами виявився особливо корисним у веб-додатку для пошуку інформації про авіакомпанії завдяки його здатності оперативно обробляти запити та забезпечувати точний вихідний результат. Застосування цього методу є ключовим для забезпечення надійності та продуктивності системи в умовах постійного оновлення даних про авіакомпанії та широкого спектру користувачів.

Важливим напрямком для подальшого вдосконалення може стати розгляд можливостей комбінування методу пошуку за ключами з іншими продвинутими техніками пошуку, що дозволить збалансувати його переваги та недоліки у різноманітних умовах використання. Такий підхід може сприяти ще більш ефективному та гнучкому функціонуванню веб-додатку, враховуючи змінність та динамічність сучасного інформаційного середовища.

Переваги методу пошуку за ключами виявилися вельми значущими. Ефективність цього підходу виявляється у швидкому визначенні місця ключа в індексованій базі даних та витяганні необхідної інформації. Завдяки використанню унікальних ключів, алгоритм дозволяє системі точно визначити інформацію про конкретну авіакомпанію, уникнувши зайвого перегляду бази даних та забезпечивши найшвидший можливий час відповіді на запити користувачів.

Однак слід враховувати певні недоліки методу пошуку за ключами. Зокрема, обмеженість структурою даних та залежність від унікальних ключів можуть створити складнощі в обробці складних запитів або у випадках втрати ключа. Крім того, великі обсяги даних можуть призвести до високого навантаження на систему, що вимагає додаткових ресурсів та уваги до оптимізації.

У підсумку, метод пошуку за ключами залишається досить перспективним і ефективним інструментом для побудови веб-додатків, спрямованих на зручний та швидкий доступ до інформації про авіакомпанії. Однак для максимальної ефективності важливо ретельно розробляти та вдосконалювати структуру бази даних, враховуючи можливі складнощі та оптимізуючи алгоритми для оптимальної продуктивності в різних сценаріях використання.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ ПОШУКУ ІНФОРМАЦІЇ ПРО АВІАКОМПАНІЮ

3.1. Структура веб-додатку для пошукі інформації про авіакомпанії

Першим кроком слід визначитися з життєвим циклом програмного забезпечення та архітектурою.

Модель водоспаду – це розподіл процесу розробки на послідовні фази, де кожна фаза передбачає певний набір завдань та ґрунтується на результатах попередньої фази. Цей підхід типовий для певних галузей інженерного проєктування. У виправленні програмного забезпечення він зазвичай використовується як менш ітеративний та більш жорсткий метод, оскільки процес рухається в одному напрямку (зверху вниз), через послідовні фази, які включають в себе концепцію, ініціацію, аналіз, дизайн, розробку, тестування, впровадження та підтримку.

Модель водоспаду мала своє походження в промисловості та будівництві, де структуроване фізичне оточення призводило до високих витрат на зміни в проєктуванні на ранніх стадіях розробки. Коли цей підхід був застосований до розробки програмного забезпечення, він був єдиним підходом для творчої роботи, що базується на знаннях.

Перша детальна презентація, яка описує використання таких етапів у розробці програмного забезпечення, була проведена Феліксом Торресом та Гербертом Д. Бенінгтоном на симпозіумі з передових методів програмування для цифрових комп'ютерів у 1956 році.

Хоча термін "водоспад" не був використаний в цій статті, перша офіційна детальна діаграма процесу, пізніше відома як "модель водоспаду", зазвичай приписується статті Вінстона В. Ройса 1970 року. У 1985 році Міністерство оборони Сполучених Штатів відобразило цей підхід у своєму стандарті *DOD-STD-2167A* для

роботи з підрядниками щодо розробки програмного забезпечення, вимагаючи виконання шести етапів: аналізу вимог до програмного забезпечення, попереднього проєкту, детального дизайну, кодування та модульного тестування, інтеграції та тестування.

Вимоги до системи та програмного забезпечення перераховані в документі з вимогами до продукту:

1. Під час аналізу формуються моделі, схеми та бізнес-правила;

2. Під час дизайну створюється архітектура програмного забезпечення;

3. Кодування включає в себе розробку, перевірку та інтеграцію програмного забезпечення;

4. Тестування полягає в систематичному виявленні та виправленні дефектів;

5. Операції включають встановлення, міграцію, підтримку та технічне обслуговування повних систем.

Отже, модель водоспаду наголошує на тому, що перехід до наступної фази повинен відбуватися лише після того, як попередню фазу перевірено і переглянуто.

Існують різні модифіковані версії моделі водоспаду, включаючи остаточну модель Ройса, які можуть включати незначні або значні зміни в цьому процесі. Ці варіації можуть включати повернення до попередньої фази після виявлення недоліків або повний повтор фази проєктування, якщо наступні фази виявилися недостатньо успішними.

Важливим аргументом на користь моделі водоспаду є те, що вона підкреслює значення документації, такої як вимоги та проєктні документи, а також вихідний код. У менш структурованих та докладно задокументованих методологіях втрачена інформація, якщо члени команди покидають проєкт до його завершення. Модель водоспаду забезпечує можливість легкої ознайомленості з проєктом для нових членів команди або для випадків зміни команди, оскільки всі необхідні документи доступні.

Модель водоспаду забезпечує систематизований підхід, оскільки вона лінійно проходить через окремі, легкі для розуміння і пояснення фази, надаючи чіткі пункти в розробці. Саме ця структура робить її зрозумілою. Модель водоспаду також надає

чіткі точки відсічі в процесі розробки, що полегшує відслідковування прогресу. Через це, вона часто використовується як вихідна точка для розуміння моделей розробки в багатьох підручниках і курсах з програмної інженерії.

Клієнти можуть не мати точно сформульованих вимог до продукту до того, як побачать його в дії, і це може викликати зміни у вимогах. Це може призвести до необхідності редизайну, перерозробки та повторного тестування, що призводить до додаткових витрат.

Розробники можуть не передбачити майбутніх труднощів під час створення нового програмного продукту або функцій. У таких випадках, було б краще переглянути проєкт, а не настоювати на виконанні проєкту, який не враховує виявлені обмеження, вимоги або проблеми.

Деякі організації намагаються вирішити відсутність конкретних вимог від клієнтів шляхом залучення системних аналітиків до вивчення існуючих ручних систем і аналізу їх функціоналу. Проте в практиці може бути важко дотримуватися жорсткого розділення між системним аналізом і програмуванням, оскільки впровадження складної системи майже завжди призводить до виявлення проблем та крайових випадків, які системний аналітик не передбачив.

У відповідь на ці проблеми були запропоновані модифіковані моделі водоспаду, такі як "модель Сашимі", "модель з фазами, що перекриваються", "модель з підпроєктами" та "модель з зменшенням ризику". Модифіковані моделі водоспаду відповідають на критику "чистої" моделі водоспаду, шляхом впровадження змін у процес розробки програмного забезпечення для зменшення витрат і полегшення управління змінами.

Монолітна архітектура програмного забезпечення (ПЗ) – це традиційний підхід до розробки програмного забезпечення, який базується на створенні одного компактного, нерозділеного програмного модуля, який включає в себе всі компоненти системи. Цей підхід використовується протягом багатьох років і продовжує бути популярним в сучасній програмно-інженерній практиці. Розглянемо основні характеристики та переваги монолітної архітектури ПЗ, а також деякі недоліки та виклики, з якими вона може зіткнутися.

1. Основні характеристики монолітної архітектури ПЗ

Один великий модуль: Монолітна архітектура передбачає розробку програмного забезпечення у вигляді одного великого модуля, де всі компоненти (наприклад, логіка бізнесу, інтерфейс користувача, база даних) розташовані в межах цього модуля.

Всеїдність: Усі компоненти системи в монолітній архітектурі обмінюються даними безпосередньо між собою, оскільки вони знаходяться в одному контексті виконання.

Лінійний процес розробки: Розробка та розгортання монолітної системи зазвичай відбуваються в лінійному порядку, коли всі компоненти розробляються та інтегруються разом, а після цього відбувається розгортання.

Загальна база коду: У монолітній архітектурі весь код програмного забезпечення знаходиться в одній кодовій базі, що спрощує розробку та управління проєктом.

2. Переваги монолітної архітектури ПЗ

Простота: Монолітна архітектура проста в розумінні та розробці, оскільки всі компоненти знаходяться в одному місці.

Швидкість розгортання: Розгортання монолітних систем зазвичай є швидким і простим, оскільки всі компоненти розробляються разом та інтегруються в єдиний образ.

Продуктивність: У монолітній архітектурі немає витрат на мережеву комунікацію між компонентами, що може поліпшити продуктивність системи.

Легке масштабування: Монолітна архітектура дозволяє легко масштабувати систему, додавши додаткові ресурси до єдиного модуля.

3. Недоліки та виклики монолітної архітектури ПЗ

Велика кодова база: У монолітних системах кодова база може стати великою та складною для управління, особливо при зростанні розміру проєкту.

Обмежена гнучкість: Монолітна архітектура може бути менш гнучкою, оскільки зміни в одному компоненті можуть мати вплив на всю систему.

Відсутність модульності: Монолітна архітектура утримує всі компоненти в одному модулі, що робить складним тестування та розгортання окремих частин системи.

Складність масштабування: Великі монолітні системи можуть бути складними для масштабування, оскільки потрібно масштабувати всі компоненти разом, навіть якщо потреба в масштабуванні є лише для деяких частин системи.

4. Сучасні тренди і підходи

Незважаючи на те, що монолітна архітектура є класичним підходом до розробки програмного забезпечення, сучасна практика демонструє широкий спектр альтернативних підходів, таких як мікросервісна архітектура, контейнеризація та серверне забезпечення функцій. Ці підходи надають більшу гнучкість, модульність та швидкість розгортання системи, але також вимагають від розробників додаткових знань та навичок.

У підсумку, монолітна архітектура програмного забезпечення є традиційним підходом, який все ще залишається популярним. Вона має свої переваги, такі як простота та швидкість розгортання. Однак, вона також має свої недоліки, такі як обмежена гнучкість та складність масштабування. З розвитком технологій та появою нових вимог до програмного забезпечення, розробники активно досліджують та використовують інші підходи, які надають більшу гнучкість та модульність системи.

На початку проєктування слід ретельно дослідити варіанти використання, що забезпечить глибоке розуміння призначення та можливої функціональної діяльності інформаційної системи в майбутньому.

Найпростішою формою візуалізації взаємодії між користувачем та системою є діаграма використання, яка дозволяє ілюструвати взаємодію користувача з різними сценаріями використання системи. Ці діаграми можуть ідентифікувати різні типи користувачів системи та різні моделі використання. Вони зазвичай поєднуються з іншими видами діаграм для отримання повної картини.

Хоча кожну можливість можна детально розглядати, використання діаграм допомагає надати загальний огляд системи на вищому рівні. Ці діаграми спрощують комунікацію зі зацікавленими сторонами і допомагають їм краще зрозуміти, як

система буде розроблятися. Якщо говорити відомими словами, "план використання це суть вашої системи".

Завдяки їх простоті, плани використання можуть стати ефективним інструментом комунікації для зацікавлених сторін. Ці зображення намагаються відтворити реальний світ та допомагають зацікавленим сторонам зрозуміти, як система буде розроблятися. Дослідження показало, що використання діаграм передає наміри системи зацікавленим сторонам більш зрозуміло, ніж діаграми класів.

Основною метою використання діаграм є відображення динамічних аспектів системи. Для більш повного функціонального і технічного опису системи можуть використовуватися інші схеми та документи. Вони надають спрощене графічне представлення того, як система має функціонувати.

Незважаючи на те, що монолітна архітектура є класичним підходом до розробки програмного забезпечення, сучасна практика демонструє широкий спектр альтернативних підходів, таких як мікросервісна архітектура, контейнеризація та серверне забезпечення функцій. Ці підходи надають більшу гнучкість, модульність та швидкість розгортання системи, але також вимагають від розробників додаткових знань та навичок.

Межа системи представляє собою прямокутник з ім'ям та еліпсом (прецедент), але може опускатися в випадках відсутності корисної інформації.

Актор – стилізована роль особи, яка відображає набір користувачів, які взаємодіють з системою або іншими сутностями. Актори не пов'язані один з одним.

Прецедент – еліпс з підписом, що вказує на системну операцію, яка виконується інтерфейсом користувача та призводить до певних результатів. Назва може бути описом того, "що" в системі виконується, а не "як". Прецеденти представляють різні сценарії використання системи.

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

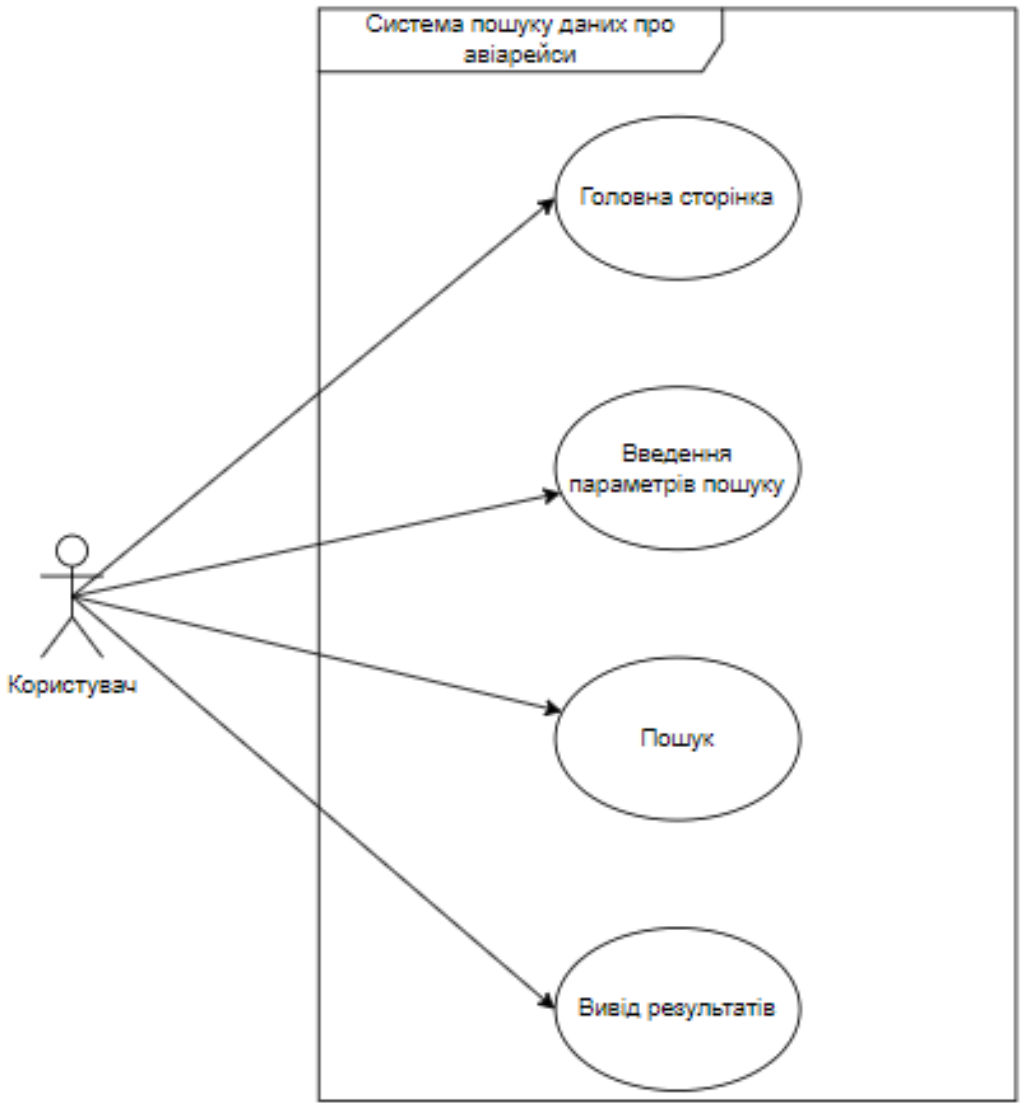


Рис. 3.1. Діаграма варіантів використання

Під час розробки програмного забезпечення, зазвичай, зображення внутрішньої структури системи подається у формі діаграми класів, яка демонструє склад класів та модулів проєкту та їх взаємодію.

У сфері програмної інженерії, діаграма класів, згідно з уніфікованою мовою моделювання (*UML*), є статичною структурною діаграмою, що надає огляд структури системи, включаючи системні класи, їх атрибути, операції (або методи) та зв'язки між об'єктами.

Діаграми класів виступають як ключові будівельні блоки об'єктно-орієнтованого моделювання. Вони використовуються для загального концептуального моделювання структури програми, а також для більш детального

моделювання з метою перетворення моделі в програмний код. Також ці діаграми можуть служити інструментом моделювання даних. Кожен клас на діаграмі класів представляє основні елементи програми та класи, які будуть реалізовані.

На графічному зображенні класу зазвичай можна виділити три окремі частини: Верхній блок, який містить назву класу, зазвичай надруковану жирним шрифтом та вирівняну за центром. Перша літера назви має бути великою. Середній блок, де розташовані атрибути класу, які вирівнюються ліворуч, а перша літера їх назви повинна бути мала. Нижній блок, який містить операції, які можуть бути виконані класом. Операції також вирівнюються ліворуч, і перша літера їх назви має бути мала.

При проєктуванні системи, визначення багатьох класів та їх групування в схему класів допомагають зрозуміти статичні відносини між ними. Для більш детального моделювання, категорія концептуального проєктування зазвичай поділяється на декілька підкатегорій.

Асоціація описує взаємозв'язок між елементами моделі. Якщо зміна одного елемента може призвести до зміни іншого, то між ними існує асоціація. Це відношення є одностороннім і представлене пунктирною лінією зі стрілкою, що вказує на напрямок замовника до постачальника.

Для подальшого опису системи, ці діаграми класів (рис. 3.2) можуть бути доповнені діаграмами станів або становими автоматами *UML*.

Асоціація включає в себе різні типи: двосторонню, односторонню, агрегаційну (включаючи комбіновану агрегацію) та рефлексивну. Серед них найпоширеніші двосторонні та односторонні асоціації.

Наприклад, взаємозв'язок між класом "польот" і класом "літак" можна описати як двосторонню асоціацію. Ця асоціація відображає статичні відносини, що існують між об'єктами обох класів.

Агрегація представляє собою варіант відношення "має", і вона є більш специфічною за асоціацію. Агрегація виражає зв'язок, в якому один клас містить частину іншого класу. Наприклад, як показано на діаграмі, клас "професор" має асоціацію з класом "викладання". У якості типу асоціації, агрегація може мати назву

та модифікації, подібні до асоціації. Але важливо відзначити, що агрегація обмежена бінарними відносинами, тобто це завжди асоціація між двома класами.

Особливістю агрегації є те, що класи, що містяться в агрегації, не мають сильної залежності від життєвого циклу контейнера. Навіть після знищення контейнера, елементи, що містяться в ньому, все ще існують.

Графічно агрегацію в *UML* зображують у вигляді порожнистого ромба, що з'єднується однією лінією з класом-господарем. Агрегація може розглядатися як семантично вищий рівень об'єкта, що взаємодіє з іншими об'єктами, хоча фізично складається з кількох менших об'єктів.

Наприклад, розглянемо взаємозв'язок між "бібліотекою" та "студентами". У цьому випадку, "студенти" можуть існувати незалежно від "бібліотеки", і відношення між ними може бути описане як агрегація, оскільки "студенти" є частиною "бібліотеки". Таке відношення можна позначити як сукупність.

Узагальнення визначає відношення "є спеціалізацією" між класами. Воно показує, що один клас (підтип) є більш конкретним представленням іншого класу (супертипу). Наприклад, "люди" є підтипом "ссавців", і "ссавці" є супертипом "тварин". Графічно відношення узагальнення в *UML* позначаються порожнім трикутником, який з'єднується зі супертипом.

Відношення узагальнення часто отримують назву спадковістю або відносинами "є". У контексті цих відносин, суперклас (також відомий як базовий клас) можна називати "батьківським класом", "суперкласом", "базовим класом" або "базовим типом". Спеціалізовані класи, що успадковують від суперкласу, можуть називатися "дочірніми" підкласами, "похідними класами", "похідними типами", "успадкованими класами" або "успадкованими типами". Важливо зазначити, що ці терміни вживаються у контексті програмування та моделювання, і вони не мають жодного спільного з біологічними відносинами між батьком і дитиною.

Звідси ми можемо висловити відношення "А є типом В" такими прикладами, як "дуб є одним із видів дерев", "автомобіль є підкласом транспорту".

В моделюванні *UML* відношення реалізації вказують на те, що один елемент моделі (клієнт) втілює або реалізує функціональність, яка була задана іншим

елементом моделі (постачальником). Це відношення графічно позначається порожнім трикутником, який з'єднаний інтерфейсом або деревом реалізаторів. Крім того, на діаграмі компонентів використовується графічна позначка "м'ячна розетка" для позначення реалізації. Важливо відзначити, що відношення реалізації зазвичай зображуються лише на діаграмах класів або компонентів, оскільки вони вказують на зв'язок між класами, інтерфейсами, компонентами і пакетами.

Залежність є слабкішою формою взаємодії, яка вказує на те, що один клас залежить від іншого і може використовувати його в певний момент часу. Відмінністю від асоціації є те, що залежність не передбачає наявність атрибутів одного класу, які були б екземплярами іншого класу. Відношення реалізації і асоціації графічно представлені як лінії, що з'єднують класи, з додатковими символами на кінцях ліній для вказівки додаткових деталей відношення.

Класи сутності в моделюванні представляють інформацію, яку система обробляє, а також поведінку, пов'язану з цією інформацією. Графічне представлення класів сутностей зазвичай має форму кола з короткою лінією, прикріпленою до нижньої частини кола, або їх можна намалювати як звичайні класи зі стереотипом "сутність" над назвою класу.

На графічному зображенні класу зазвичай можна виділити три окремі частини: Верхній блок, який містить назву класу, зазвичай надруковану жирним шрифтом та вирівняну за центром. Перша літера назви має бути великою. Середній блок, де розташовані атрибути класу, які вирівнюються ліворуч, а перша літера їх назви повинна бути мала. Нижній блок, який містить операції, які можуть бути виконані класом. Операції також вирівнюються ліворуч, і перша літера їх назви має бути мала.

На рисунку 3.2 зображено діаграму класів, яка відображає внутрішню будову проєкту.

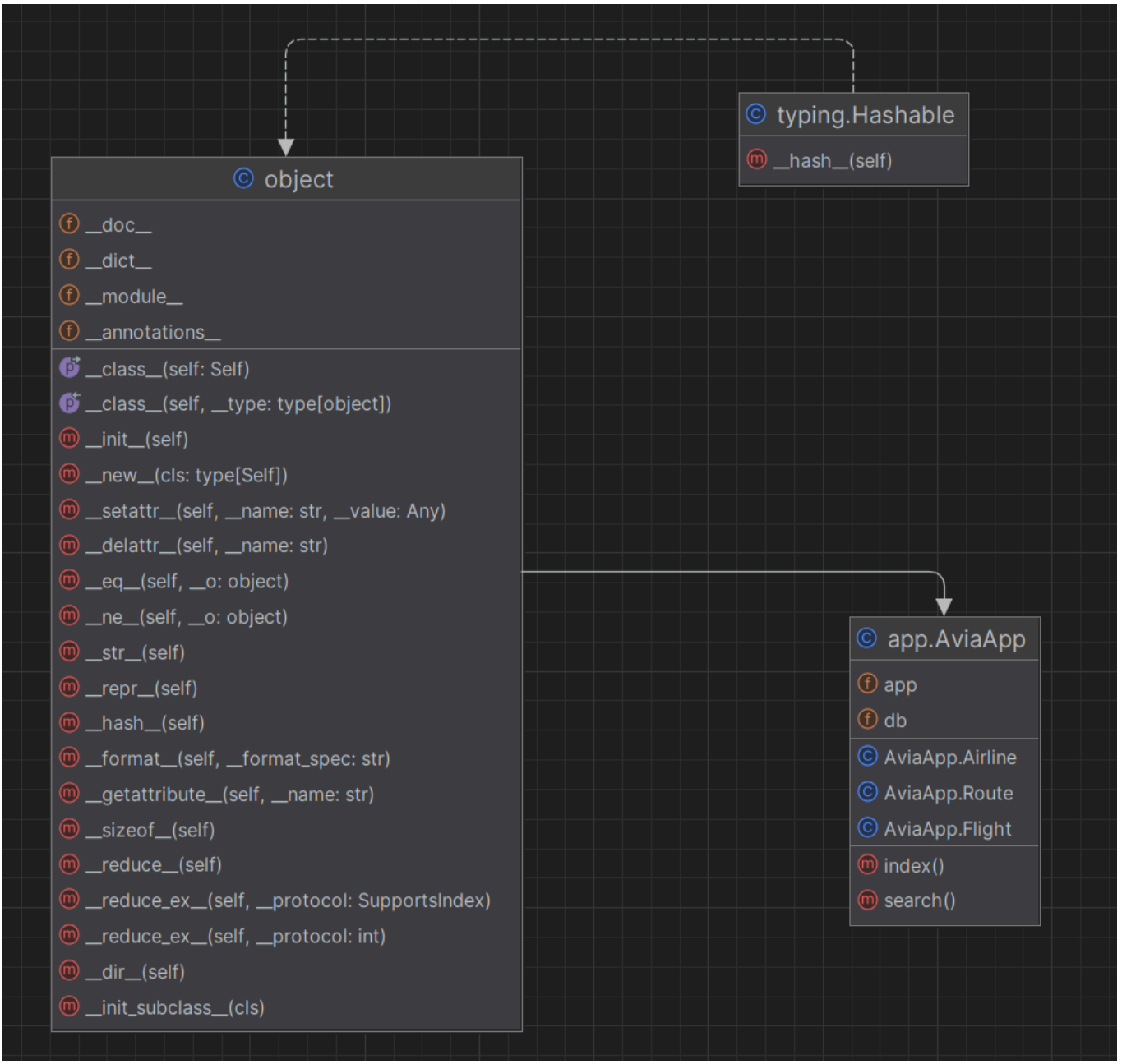


Рис. 3.2. Діаграма класів системи

Проєктування бази даних є ключовим етапом у створенні ефективної інформаційної системи. Воно включає в себе визначення структури, зв'язків та типів даних, які будуть зберігатися. Якісно спроєктована база даних дозволяє підвищити продуктивність системи, спростити масштабування і забезпечити надійність зберігання даних.

1. Структура та організація даних: Якісне проєктування бази даних забезпечує логічну та зрозумілу організацію даних. Це сприяє легкості доступу та ефективності обробки даних.

2. Ефективність і швидкість: Належне проєктування впливає на швидкість запитів до бази даних. Оптимізовані схеми і індекси можуть значно прискорити доступ до даних.

3. Масштабування: Коли база даних спроєктована з урахуванням потенційного росту, масштабування стає менш болісним процесом. Вона може ефективно обробляти збільшення обсягів даних та запитів.

4. Інтеграція з іншими системами: Добре спланована база даних легше інтегрується з іншими системами, такими як *CRM*, *ERP* тощо.

5. Безпека: Надійне проєктування бази даних включає розробку політик безпеки, що допомагає захистити дані від несанкціонованого доступу та інших загроз.

6. Забезпечення цілісності даних: Належне проєктування допомагає у підтримці цілісності даних, забезпечуючи актуальність, точність та надійність інформації.

ER-діаграми (*Entity-Relationship Diagrams*):

ER-діаграми є фундаментальним інструментом в процесі проєктування баз даних. Вони відображають сутності в базі даних та їх взаємозв'язки, дозволяючи легко візуалізувати структуру даних.

1. Візуалізація структури: *ER*-діаграми надають чітке уявлення про структуру бази даних, відображаючи сутності та зв'язки між ними.

2. Спрощення спілкування: Вони допомагають спростити обговорення структури бази даних між розробниками, аналітиками та не технічними зацікавленими сторонами.

3. Планування і аналіз: *ER*-діаграми використовуються для аналізу і планування структури бази даних, дозволяючи ідентифікувати можливі проблеми або потреби в оптимізації.

4. Нормалізація: Вони сприяють процесу нормалізації, що допомагає уникнути зайвого дублювання даних і забезпечує ефективність зберігання.

5. Документація: *ER*-діаграми служать як цінний документаційний ресурс, який може бути використаний для орієнтування в базі даних новими членами команди або під час її модифікації.

У підсумку, якісне проєктування бази даних і використання *ER*-діаграм є вирішальними для створення міцної, гнучкої та ефективної системи зберігання і обробки даних.

На рисунку 3.3 представлено структуру бази даних системи.

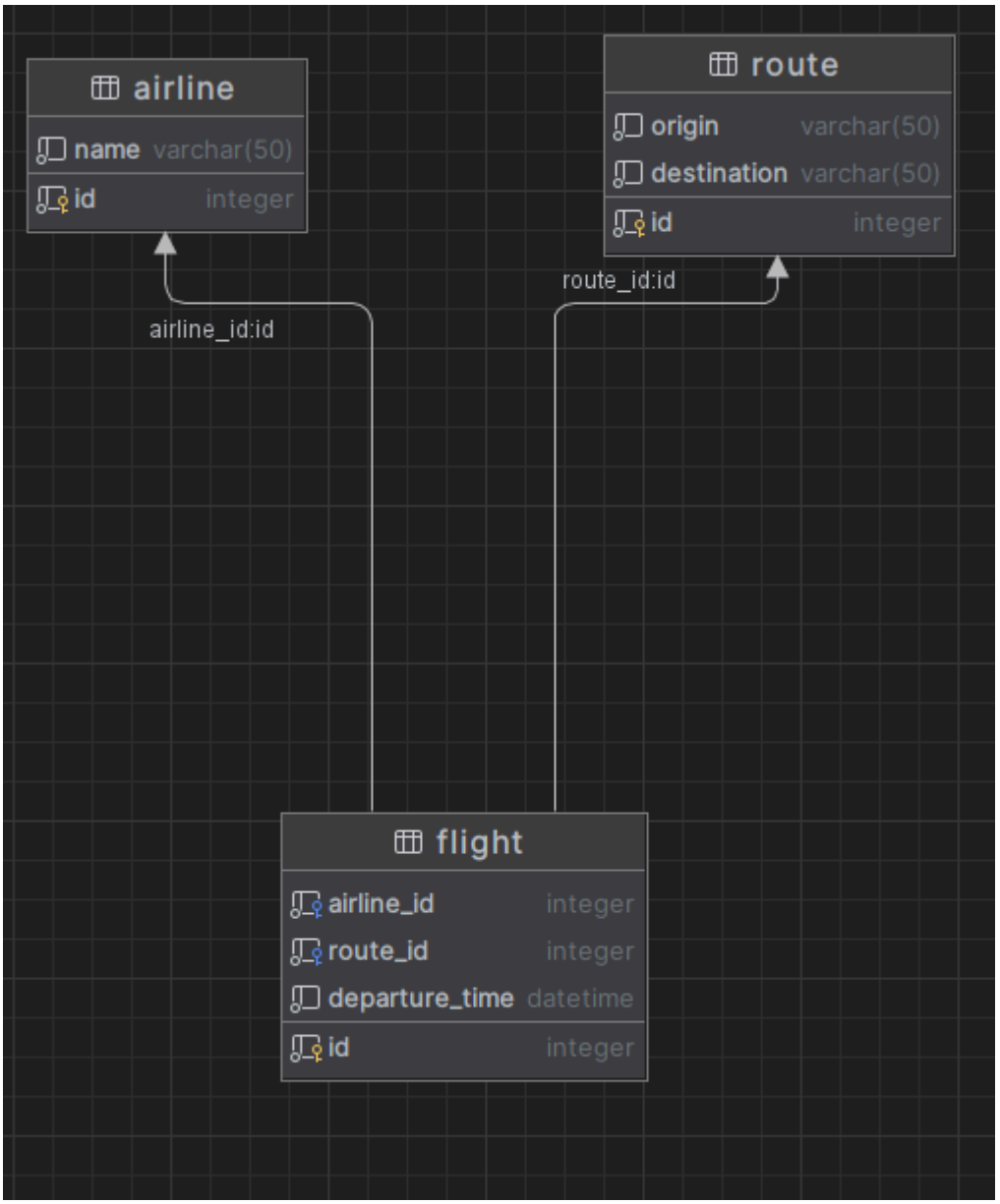


Рис. 3.3. Структура бази даних

База даних складається з трьох основних сутностей, а саме авіакомпанії, маршруту і польоту.

Нижче представлено функцію, яка проводить процедуру пошуку інформації.

```
@app.route('/search', methods=['GET'])
def search():
    airline_name = request.args.get('airline_name')
    origin = request.args.get('origin')
    destination = request.args.get('destination')
    date_from = request.args.get('date_from')
    date_to = request.args.get('date_to')

    query = Flight.query

    if airline_name:
        query = query.join(Airline).filter(Airline.name == airline_name)
    if origin and destination:
        query = query.join(Route).filter(Route.origin == origin, Route.destination == destination)
    if date_from and date_to:
        date_from = datetime.strptime(date_from, '%Y-%m-%d')
        date_to = datetime.strptime(date_to, '%Y-%m-%d')
        query = query.filter(Flight.departure_time >= date_from, Flight.departure_time <= date_to)

    flights = query.all()
    return render_template('search_results.html', flights=flights)
```

Функція *search()* у *Flask*-додатку служить для реалізації механізму пошуку рейсів за різними параметрами. Процес пошуку розгортається наступним чином:

1. Отримання Параметрів Пошуку:

airline_name, origin, destination, date_from, date_to отримуються з *HTTP*-запиту. Ці параметри визначають критерії, за якими буде проводитись пошук.

2. Ініціалізація Початкового Запиту:

Визначається базовий запит *SQL* за допомогою *Flight.query*. Цей запит може бути модифікований в подальшому залежно від вхідних параметрів.

3. Фільтрація за Назвою Авіакомпанії:

Якщо задано *airline_name*, запит розширюється шляхом додавання з'єднання (*JOIN*) з таблицею *Airline* і фільтрації за назвою авіакомпанії.

4. Фільтрація за Маршрутом:

Якщо вказані *origin і destination*, до запиту додається з'єднання з таблицею *Route* та фільтрація за пунктами відправлення та призначення.

5. Фільтрація за Датою:

Якщо вказані *date_from ma date_to*, вони спершу конвертуються з рядкового формату у формат дати. Потім до запиту додається фільтрація, щоб вибрати рейси, що відповідають заданому діапазону дат.

6. Виконання Запиту та Повернення Результатів:

Після застосування всіх фільтрів, запит виконується за допомогою методу *.all()*, який повертає усі рейси, що відповідають заданим критеріям.

Результати запиту передаються у шаблон *search_results.html* для відображення користувачу.

Ця функція ефективно інтегрує можливості *SQLAlchemy* для виконання складних запитів до бази даних, дозволяючи користувачам фільтрувати рейси за різними параметрами.

ER-діаграми є фундаментальним інструментом в процесі проєктування баз даних. Вони відображають сутності в базі даних та їх взаємозв'язки, дозволяючи легко візуалізувати структуру даних.

Для введення і відображення даних було створено 2 веб-сторінки, а саме сторінка з формою пошуку (рис. 3.4) і сторінка з результатами (рис. 3.5).

Рис. 3.4. Сторінка фільтрів

Рис. 3.5. Сторінка результатів

3.2. Кількісна оцінка параметрів роботи веб-додатку для пошуку інформації про авіакомпанії

В першу чергу необхідно описати технології, які були використані для розробки системи.

Python, мова програмування загального призначення високого рівня, яка працює на принципах, спрямованих на забезпечення читабельності коду через використання значних відступів. Її мовна структура та об'єктно-орієнтований підхід призначені для допомоги розробникам у написанні послідовних та логічних кодів як для невеликих, так і для великих проєктів.

Python володіє динамічним збором сміття та підтримує кілька парадигм програмування, включаючи структурне (включаючи процедурне), об'єктно-орієнтоване та функціональне програмування. Велику користь приносить розширена стандартна бібліотека *Python*, яку часто називають "мовою з вбудованими батареями".

Як наступник мови програмування *ABC*, Гвідо ван Россум розпочав роботу над *Python* наприкінці 1980-х років і випустив першу версію під назвою *Python* 0.9.0 в 1991 році. У 2000 році був представлений *Python* 2.0, який вніс нові функції, такі як розширене керування списками та систему збору сміття на основі рахунку посилань. *Python* 2 був офіційно припинений в 2020 році з випуском версії 2.7.18. *Python* 3.0, випущений у 2008 році, є головною версією мови, яка, хоч і не є повністю сумісною з попередньою версією, стала стандартом.

Python завжди володів великою популярністю серед мов програмування. Гвідо ван Россум розпочав роботу над ним наприкінці 1980-х років в Нідерландах, розвиваючи його як наступник мови програмування *ABC*, яка була натхненною *SETC* та спроможною опрацьовувати винятки та взаємодіяти з операційною системою *Amoeba*. Перша реалізація *Python* розпочалася у грудні 1989 року. Він вів розробку проєкту, бувши провідним розробником, до 12 липня 2018 року, коли оголосив про свою "постійну відпустку" і отримав титул "Диктатора, дружнього до життя" *Python*. Ван Россум був завжди відомий своєю відданістю спільноті *Python* та його

тривалими внесками у проєкт. Він продовжує грати важливу роль, беручи участь в роботі наглядової ради *Python* разом з іншими розробниками. У січні 2019 року активні розробники ядра *Python* обрали Бретта Кеннона, Ніка Коглана, Баррі Варшава, Керол Віллінг та самого Гвідо ван Россума до складу ради директорів з п'яти членів. Відтоді Гвідо ван Россум зняв свою кандидатуру з розгляду на посаду в раді директорів в 2020 році.

Python 2.0 був випущений 16 жовтня 2000 року, додавши значну кількість нових функцій, включаючи систему збору сміття для ефективного управління пам'яттю і підтримку *Unicode*.

За ним прийшов *Python* 3.0, випущений 3 грудня 2008 року, і він був значною ревізією мови, яка не була повністю сумісною з попередньою версією. Багато ключових особливостей були поступово впроваджені в *Python* версіях 2.6.x і 2.7.x, і для полегшення переходу існує утиліта *2to3*, яка допомагає автоматично перетворювати код з *Python* 2 на *Python* 3.

Початково планувалося припинити підтримку *Python* 2.7 в 2015 році, але через обширну кількість існуючого коду, який було б складно перенести на *Python* 3, ця дата була перенесена до 2020 року. Після закінчення терміну підтримки, *Python* 2.7 більше не отримуватиме оновлень безпеки та інших виправлень, і підтримуватимуться лише версії *Python* 3.6.x і новіші.

Python є багатопарадигмовою мовою програмування, повністю підтримує об'єктно-орієнтоване і структурне програмування, а також надає можливості функціонального і аспектно-орієнтованого програмування, включаючи метапрограмування та метаоб'єктний підхід. *Python* також включає динамічний збір сміття і підтримує динамічне розділення імен для зв'язування методів і змінних під час виконання програми.

Зокрема, *Python* розроблений з метою підтримки функціонального програмування, запозичуючи функції, такі як фільтрація, відображення, скорочення і ітератори. Стандартна бібліотека містить модулі *itertools* і *functools*, які реалізують функціональні інструменти, вдосконалені за запозиченням з мов *Haskell* і *Standard ML*.

Концепція дзену *Python* (*PEP* 20) узагальнює основні принципи мови, включаючи такі вислови:

1. Красиве – краще, ніж потворне;
2. Явне краще, ніж неявне;
3. Просте – краще, ніж складне;
4. Складний – це краще, ніж складний;
5. Підрахунок читабельності.

Python має вражаючу розширюваність завдяки своєму модульному підходу, хоча не всі його функції вбудовані в ядро мови. Ця модульність дозволяє легко додавати програмовані інтерфейси до існуючих програм, роблячи його популярним серед розробників для розширення функціональності.

Гвідо ван Россум визначив фундаментальні принципи дизайну *Python*, зокрема, зосередженість на читабельності коду, змістовній та зрозумілій синтаксису та простоті у виборі методів кодування. Як відмінність від філософії *Perl* "Є більше одного способу зробити це", *Python* спирається на девіз "Повинен бути один – бажано лише один – очевидний спосіб зробити це."

Розробники *Python* докладають зусиль, щоб запобігти передчасній оптимізації і не вносять змін до некритичних частин *CPython*, які можуть покращити швидкість за рахунок складності. Там, де швидкість має велике значення, програмісти можуть перевести критичні фрагменти коду на розширення, написані на *C*, або використовувати інші інтерпретатори, такі як *PyPy*, які пропонують компіляцію "точно вчасно". Також є варіанти, такі як *Cython*, які можуть конвертувати *Python*-код в *C* і забезпечити прямий доступ до *API* рівня *C* для інтерпретатора *Python*.

Однією з важливих цілей розробників *Python* є забезпечення розваги в процесі програмування. Це відображено в назві мови, яка походить від британської комедійної групи "Монті Пайтон". У підручниках і довідкових матеріалах *Python* також може підкреслювати гумор та гарний настрій, включаючи використання відомих прикладів, як "*spam*" і "*eggs*," як стандартних імен.

Серед користувачів і прихильників *Python*, особливо серед тих, які вважають себе обізнаними та досвідченими, часто вживається термін "*Pythonists*." *Python*

робить акцент на зрозумілій і легко читабельній мові. У порівнянні з іншими мовами, він не використовує фігурні дужки або роздільні слова для виділення блоків коду, і робить акцент на відступах, які мають семантичне значення.

Рекомендований розмір відступу – чотири пробіли.

Оператори *Python* включають (серед іншого):

Використовуйте символ "=" як оператор присвоєння.

Оператор "*if*" використовується для умовного виконання блоку коду. Також, використовуються "*else*" та "*elif*" (скорочення від "*else-if*").

Для перегляду ітераційного об'єкта та захоплення кожного елемента в локальну змінну, щоб використовувати його вкладеним блоком, використовуйте оператор "*for*".

Поки умова є істинною, виконуйте блок коду за допомогою оператора "*while*".

Оператор "*try*" дозволяє перехоплювати та обробляти винятки, які містяться в вкладених блоках коду, і гарантує виконання чистого коду у кінцевому підсумку, незалежно від того, як виконується блок.

Використовуйте оператор "*raise*" для генерації певного винятку або повторного виклику перехопленого винятку.

Оператор класу виконує блок коду та додає свій локальний простір імен до класу для використання в об'єктно-орієнтованому програмуванні.

Оператор "*def*" визначає функцію або метод.

Оператор *Python* 2.5, випущений у вересні 2006 року, дозволяє охоплювати блок коду в контекстному менеджері, що дозволяє ініціалізувати та звільняти ресурси, подібно до шаблону ініціалізації та вилучення ресурсів (*RAII*), замінюючи часту ідіому "*try/finally*".

Оператор "*break*" завершує виконання циклу.

Оператор "*continue*" переходить до наступної ітерації, пропускаючи поточну.

Оператор "*del*" видаляє змінну, при цьому видаляються всі посилання на значення, а спроби використовувати змінну призводять до помилки. Видалену змінну можна повторно створити.

Оператор "*pass*" виконує функцію *NOP* (*No Operation*) та використовується для створення порожнього блоку коду.

Оператори "*assert*" використовуються для перевірки умов під час налагодження.

Оператор "*yield*" використовується для повернення значень з функції-генератора.

Оператор "*return*" використовується для повернення значення з функції.

Оператор імпорту використовується для приєднання модуля, чия функція або змінна доступна для використання в поточній програмі. Існують три способи імпорту: імпорт <ім’я модуля> [як <псевдонім>], використання <ім’я модуля> імпорт *, або вказування <ім’я модуля> імпорт <визначення 1> [як <псевдонім 1>], <визначення 2> [як <псевдонім 2>], ...

Оператор присвоєння (=) працює, зв'язуючи ім'я з посиланням на конкретний об'єкт, який розподіляється динамічно. Потім змінні можуть бути переприв'язані в будь-який об'єкт за потреби. У *Python* ім'я змінної не обмежене типом даних і не має фіксованого типу, призначеного для нього. Проте в певний момент часу змінна посилатиметься на конкретний об'єкт з певним типом. Це відомо як динамічна типізація і протилежно статичній типізації, яку використовують статично набрані мови програмування, де кожна змінна обмежена значенням певного типу.

Python не підтримує оптимізацію хвостових викликів або першокласні розширення, і, за словами Гвідо ван Россума, ніколи не буде. Однак, розширивши генератор *Python*, у версії 2.5 надається краща підтримка функцій, подібних до корутина. До 2,5 генераторів є ледачими ітераторами, інформація передається від генераторів в одному напрямку. З *Python* 2.5 ви можете передавати інформацію назад до функції генератора, а з *Python* 3.3 ви можете передавати інформацію через кілька рівнів стека.

Деякі вирази *Python* подібні до виразів у таких мовах, як *C* і *Java*, а деякі ні.

Операції додавання, віднімання та множення мають однакове структурне виконання, але розділення використовує два різні оператори в *Python*. Існує поділ на

поверхню (або ціле поділ) // і ділення з плаваючою точкою. Також *Python* використовує оператор ** для піднесення в ступінь.

Недавно в *Python* 3.5 було додано новий інфіксний оператор @, спрямований на бібліотеки, такі як *NumPy*, для множення матриць.

У *Python* 3.8 був представлений оператор присвоєння "=", який отримав прізвище «оператор моржа». Він дозволяє присвоювати значення змінним як частину більшого виразу.

В порівнянні з *Java*, де операція порівняння значень виконується за допомогою "==", *Python* порівнює за значенням за допомогою "==" і також дозволяє порівнювати об'єкти за посиланням за допомогою оператора *is*. У *Python* можна також ланцюжково порівнювати значення, наприклад *a* <= *b* <= *c*.

Python використовує ключові слова "та", "або" і "не" для логічних операцій, відмінно від *Java* та *C*, які використовують символи &&, || та !.

В *Python* доступні два види виразів – це спискові вирази та більш загальні генераторні вирази.

Для анонімних функцій використовуються лямбда-вирази, але вони обмежені тим, що можуть містити лише один вираз.

Умовні вирази в *Python* записуються як x, якщо c, інакше y, відмінність полягає в порядку операндів відносно інших мов, таких як *c? X: y*, які є загальними для багатьох інших мов.

Python розрізняє списки та кортежі. Список записується у вигляді [1, 2, 3], він є змінним та не може використовуватися як ключ словника (ключі словника повинні бути незмінними). Кортеж записується як (1, 2, 3) і є незмінним, тому, поки всі елементи кортежу незмінні, його можна використовувати як ключ словника. Оператор + може використовуватися для об'єднання двох кортежів, створюючи новий кортеж, що містить всі значення об'єднаних кортежів, і це не змінює їх вміст, але створює новий кортеж, що містить об'єднані значення. Тому, коли ви виконуєте *t* = *t* + (4, 5), це фактично створює новий кортеж, який включає всі значення як вихідного кортежу *t*, так і (4, 5).

У *Python* існує операція розпакування послідовності, де кілька виразів пов'язуються в тій самій структурі, що і літерал кортежу, і призначають значення праворуч від оператора = кожному виразу зліва. Оператор очікує послідовність значень праворуч від =, створює кожне значення та призначає його відповідному виразу зліва.

У *Python* є оператор форматування рядків "%", аналогічний функції *printf* у *C*. Наприклад, "спам =% s яйця =% *d*"% ("*blah*", 2) оцінюється як "спам = *blah*-яйця = 2". У *Python* 3 і 2.6+ цей оператор доповнюється методом format() класу *str*, як, наприклад, "спам = {0} яйця = {1}". *format* ("*blah*", 2). У *Python* 3.6 додано "*f*-рядки", які дозволяють вставляти значення в рядок безпосередньо в тексті, наприклад *blah* = "*blah*"; *egg* = 2; *f'spam* = {*blah*} яйця = {яйця}'.

В *Python*, рядки можна з'єднати за допомогою "додавання" (використовуючи той самий оператор, що і для додавання чисел типу ціле або плаваюче число). Наприклад, "спам" + "яйце" дає результат "спам". Навіть коли рядок містить числа, вони також обробляються як рядки, а не числа. Таким чином, "2" + "2" дорівнює "22". *Python* має різні типи літералів для рядків:

Рядки, які обмежені одинарними або подвійними лапками, працюють однаково, на відміну від оболонки *Unix* і деяких інших мов, де одинарні та подвійні лапки мають відмінний синтаксис. В обох випадках можна використовувати зворотну косу риску () для вставки спеціальних символів. У *Python* 3.6 доступна рядкова інтерполяція за допомогою "*f*-рядків".

Рядки, обмежені потрійними одинарними або потрійними подвійними лапками, можуть охоплювати кілька рядків і використовуються, наприклад, для створення рядкових документів.

Також, рядкові літерали позначаються префіксом "*r*" і не обробляють послідовності зворотнього слеша. Вони корисні там, де зворотні слеші мають спеціальне значення, таке як у регулярних виразах та шляхах у стилі *Windows*. *Python* надає можливість працювати з індексами та виразами нарізування масивів у списках, використовуючи синтаксис, який виглядає як [індекс], [початок:кінець] або [початок:кінець:крок]. Індекси рахуються з нуля, а негативні індекси відраховуються

від кінця. Зрізи включають елементи від початкового індексу до, але не включають кінцевого індексу. Третій параметр зрізу, відомий як крок, дозволяє пропускати та обертати елементи. Якщо не вказувати індекси зрізу, наприклад [:], то буде створено копію всього списку. Важливо враховувати, що кожен елемент у зрізі є неглибокою копією.

Python чітко розрізняє вирази та вирази, що відрізняється від мов, таких як *Common Lisp, Scheme* або *Ruby*. Це може призвести до дублювання певних функцій.

Метод для об'єкта – це функція, приєднана до класу об'єкта; синтаксис *instance.method* (аргумент) – це синтаксичний цукор *Class.method* (екземпляр, аргумент) для поширених методів і функцій. Методи *Python* мають явний параметр *self* для доступу до даних екземпляра, що є протилежністю неявного *self* (або цього) в деяких інших об'єктно-орієнтованих мовах програмування (наприклад, *C++, Java, Objective-C* або *Ruby*).

Python використовує набір качок і має типізовані об'єкти, але нетиповані імена змінних. Обмеження типу не перевіряються під час компіляції; навпаки, операції над об'єктом можуть бути невдалими, а це означає, що об'єкт не має відповідного типу. Хоча *Python* динамічно типізується, він забороняє невизначені операції (наприклад, додавання чисел до рядка) і не намагається зрозуміти їх тихо.

Python дозволяє програмістам визначати власні типи, використовуючи класи, які найчастіше використовуються в об'єктно-орієнтованому програмуванні. Новий екземпляр класу створюється шляхом виклику класу (наприклад, *SpamClass*() або *EggsClass*()).Клас – це екземпляр типу метакласу (сам екземпляр), який дозволяє виконувати метапрограмування та відображати.

До версії 3.0 *Python* мав два типи класів: старі та нові. Синтаксис двох стилів однаковий, різниця полягає в тому, чи успадковується об'єкт класу безпосередньо чи опосередковано (усі класи нового стилю успадковуються від об'єкта і є екземплярами типу). У *Python* 2, починаючи з *Python* 2.2, ви можете використовувати ці два типи класів. Класи старого стилю були ліквідовані в *Python* 3.0.

Довгостроковий план передбачає підтримку інкрементного введення, а в *Python* 3.5 синтаксис мови дозволяє вказувати статичні типи, але вони не

перевіряються в реалізації *CPython* за замовчуванням. Експериментальна додаткова статична перевірка типу mypy підтримує перевірку типу під час компіляції.

CPython є офіційною реалізацією *Python* і впроваджує мову на платформі *C*. Він відповідає стандарту *C89* і деяким функціям з *C99* (зауважте, що в пізніших версіях *C*, це вже є застарілим). *CPython* також включає власні розширення мови *C*, які розширюють стандарт. Він компілює програми *Python* у проміжні байт-коди, які потім виконуються на його віртуальній машині. *CPython* поставляється з широкою стандартною бібліотекою, написаною як на *C*, так і на чистому *Python*. Цей інтерпретатор працює на різних платформах, включаючи *Windows*, багато *Unix*-подібних систем та інші операційні системи.

Розробка *Python* головним чином ведеться з використанням Програми покращення *Python* (*PEP*), яка є ключовим механізмом для запровадження нових функцій, узгодження думок спільноти та прийняття рішень з розробки мови. Стиль коду *Python* визначено в *PEP* 8. Покращення мови відповідають розробці *CPython*. Для обговорення конкретних питань використовується інструмент відстеження помилок *Roundup* на *bugs.python.org*.

Відповідно до того, наскільки значно змінюється мова, існує три типи доступних випусків *CPython*:

Зворотна несумісна версія, яка призводить до розриву і вимагає ручного оновлення коду. У цьому випадку перша частина номера версії змінюється. Ці версії досить рідкісні, наприклад, версія 3.0 була випущена через 8 років після версії 2.0.

Основні або "функціональні" версії, які виходять приблизно кожні 18 місяців і призводять до введення нових функцій, але в основному сумісні із попередніми версіями. З версії *Python* 3.9 вводиться річний графік випуску.Друга частина номера версії збільшена. Патч підтримує всі основні версії протягом кількох років після його випуску.

Нові версії *Python*, які не включають нові функції, зазвичай виходять приблизно кожні три місяці та лише після усунення достатньої кількості помилок, відзначених з часу попереднього випуску. У таких версіях також виправляють вразливості безпеки. Остання частина номера версії збільшується на одиницю.

Багато альфа-, бета- та реліз-кандидатів проходять попередній огляд і тестування перед остаточним релізом. Незважаючи на приблизний графік виправлення проблем, якщо код не готовий, зазвичай випуск затримується. Під час розробки команда розробників *Python* активно виконує велику кількість модульних тестів для відстеження стану коду.

Головною академічною конференцією для *Python* є *PyCon*. Крім того, існують спеціалізовані навчальні програми для *Python*, такі як *Pyladies*.

Python 3.10 припиняє підтримку *wstr* (ця функція буде видалена в *Python* 3.12). Це означає, що розширення *Python* повинно бути адаптоване до цих змін, перш ніж вийде *Python* 3.12. Крім того, в майбутній версії планується додати підтримку відповідності шаблону до мови.

Починаючи з 2003 року, *Python* входить до десятки найпопулярніших мов програмування за індексом *TIOBE*. За даними на лютий 2021 року він займав третє місце за популярністю (після *Java* та *C*). *Python* був визнаний мовою програмування року ("Річний найвищий ріст рейтингу") в 2007, 2010, 2018 та 2020 роках, що робить його єдиною мовою, яка здійснила це чотири рази.

Емпіричні дослідження показали, що для вирішення програмних завдань, включаючи роботу з рядками та операції зі словниками, скриптові мови, такі як *Python*, зазвичай виявляються більш продуктивними, ніж стандартні мови, такі як *C* і *Java*. Вони також часто вимагають менше використання пам'яті, ніж *Java*, і видаються більш ефективними, ніж *C* чи, ще гірше, *C++*.

Великі організації, включаючи *Wikipedia*, *Google*, *Yahoo*!, *CERN*, *NASA*, *Facebook*, *Amazon*, *Instagram*, *Spotify*, а також менші компанії, такі як *ILM* та *ITA*, використовують *Python*. Сайт соціальних новин *Reddit* в основному побудований на *Python*.

Python може використовуватися як мова сценаріїв для веб-програм, наприклад, за допомогою *mod_wsgi* для веб-сервера *Apache*. Для полегшення розробки таких програм існують стандартні *API*, які надають інтерфейс для взаємодії з веб-сервером. Для створення та підтримки складних програм використовують веб-фреймворки, такі як *Django*, *Pylons*, *Pyramid, TurboGears, web2py, Tornado, Flask,*

Bottle і *Zope*. *Python* можна також використовувати для розробки клієнтської частини програм на основі *Ajax*, використовуючи *Pyjs* і *IronPython*. *SQLAlchemy* можна використовувати для взаємодії з реляційними базами даних. *Twisted* є фреймворком для програмного взаємодії між комп'ютерами і використовується, наприклад, *Dropbox*.

Бібліотеки, такі як *NumPy*, *SciPy* і *Matplotlib*, роблять *Python* корисним для наукових обчислень, тоді як спеціалізовані бібліотеки, наприклад *Biopython* і *Astropy*, надають можливості, орієнтовані на конкретні галузі. *SageMath-Math* є програмним продуктом з інтерфейсом ноутбука, який може бути програмований на *Python* і охоплює широкий спектр математичних галузей, включаючи алгебру, комбінаторику, числову математику, теорію чисел і обчислення. *OpenCV* має розширені можливості в галузі комп'ютерного зору та обробки зображень.

Python також широко використовується у проєктах зі штучного інтелекту та машинного навчання, включаючи бібліотеки, такі як *TensorFlow*, *Keras*, *Pytorch* і *Scikit-learn*. Завдяки своїй модульній архітектурі, простому синтаксису та багатими інструментами для обробки тексту, *Python* також є популярним для роботи з обробкою природних мов. Крім того, *Python* використовується як мова сценаріїв у багатьох програмних продуктах, включаючи *Abaqus*, *FreeCAD*, програми для *3D*-анімації, програми для обробки зображень та програми для роботи з графічними зображеннями.

Python є складовою багатьох операційних систем, включаючи *Linux*, *AmigaOS* 4, *FreeBSD*, *NetBSD*, *OpenBSD* та *macOS*. Він широко використовується в інсталяторах для дистрибутивів Linux, таких як *Ubuntu* та *Red Hat Linux*, і в системі керування пакетами *Portage* для *Gentoo Linux*.

Мова *Python* також використовується для розробки експлойтів та інформаційної безпеки. Багато програм для навчання, такі як Sugar Laptop *XO* і проєкт *Raspberry Pi*, використовують *Python* для програмування користувачів.

LibreOffice включає *Python* і планує замінити *Java* на *Python* в якості мови для сценаріїв. Його постачальник сценаріїв *Python* став важливою особливістю версії 4.0, яка вийшла 7 лютого 2013 року.

З урахуванням широкого спектру застосувань та великих можливостей мови *Python*, яку було описано вище, можна визначити, що ця мова програмування відзначається універсальністю. Саме через це вона обрана для виконання цієї роботи.

PyCharm представляє собою інтегроване середовище розробки (*IDE*), спеціально призначене для програмування мовою *Python*. Це програмне забезпечення було розроблене чеською компанією *JetBrains* і володіє широким функціоналом, включаючи аналіз коду, графічний налагоджувач, вбудований модуль тестування та можливість інтеграції з системами контролю версій (*VCS*). *PyCharm* також підтримує використання фреймворку *Django* для розробки веб-додатків і платформи *Anaconda* для аналізу даних.

Ця інтегрована середовище розробки доступна для користувачів *Windows*, *macOS* та *Linux*, завдяки чому вона є кросплатформовою. Крім безкоштовної версії *PyCharm Community Edition* з відкритим вихідним кодом, існує також професійна версія, яка надає додатковий функціонал і розповсюджується за власною ліцензією.

Релізи *PyCharm* виходять періодично, дозволяючи користувачам отримувати оновлення та нові можливості. Наприклад, бета-версія була запущена у липні 2010 року, і вже через три місяці з'явилася версія 1.0. Пізніше були випущені версії 2.0, 3.0 і 4.0 у різні роки.

Зокрема, *PyCharm Community Edition*, яка є вільно доступною для користувачів з відкритим кодом, була представлена 22 жовтня 2013 року.

Flask – це мікро веб-фреймворк для *Python*, який здобув популярність завдяки своїй простоті та гнучкості. Він був створений у 2010 році Арміном Ронакером та є відкритим програмним забезпеченням, ліцензованим *MIT*. "Мікро" у контексті *Flask* не означає, що фреймворк призначений для розробки лише малих аплікацій, а вказує на те, що він надає основні інструменти для розробки веб-додатків, одночасно дозволяючи розробникам розширювати його функціональність за допомогою різноманітних розширень.

Основні особливості *Flask* включають:

1. Простота використання: *Flask* пропонує чистий і зрозумілий інтерфейс для обробки веб-запитів і відповідей. Він має вбудовані розширення для маршрутизації, сесій, обробки форм та багато іншого.

2. Гнучкість: Фреймворк не нав'язує жорстких архітектурних шаблонів, як *MVC* (*Model-View-Controller*), дозволяючи розробникам вибирати структуру, яка найкраще підходить для їхнього проєкту.

3. Розширюваність: *Flask* можна легко розширити за допомогою розширень, які додають нові функції, такі як обробка форм, аутентифікація користувачів, управління базами даних та інше.

4. Велика спільнота: *Flask* має велику та активну спільноту, яка постійно створює та підтримує розширення, а також надає допомогу новим користувачам.

Використання *Flask* ідеально підходить для створення *RESTful API*, мікросервісів, простих веб-сайтів та складних веб-додатків. Його легкість та модульність роблять *Flask* зручним для стартапів та малих команд, які прагнуть швидко розробити прототип або *MVP* (мінімально життєздатний продукт).

Flask підтримує розробку з дотриманням принципів 12-факторної аплікації, що робить його придатним для впровадження в сучасні облачні сервіси, такі як *Heroku*, *AWS* та *Google Cloud Platform*. Його легкість забезпечує швидке розгортання та легке масштабування.

Безпека є однією з ключових проблем у розробці веб-додатків, і *Flask* включає кілька вбудованих механізмів для захисту від поширених загроз, таких як *XSS* (*Cross-Site Scripting*) та *CSRF* (*Cross-Site Request Forgery*). Тим не менш, розробники все одно повинні бути уважними при проєктуванні своїх систем, щоб забезпечити належний рівень безпеки.

Документація *Flask* – це ще один сильний бік фреймворку. Вона детальна, добре організована та містить багато прикладів коду, що дозволяє новим користувачам швидко освоїтися з фреймворком. Крім того, є багато навчальних матеріалів, включаючи книги, відеокурси та туторіали від спільноти.

Flask пропонує такі основні концепції:

1. Маршрутизація: *Flask* дозволяє легко визначати маршрути *URL* з відповідними функціями обробки, що робить процес керування веб-запитами інтуїтивни;

2. Шаблони: За допомогою інтеграції з системою шаблонів *Jinja2*, *Flask* забезпечує потужні можливості для генерації динамічного *HTML*-контенту;

3. Сесії та повідомлення: *Flask* підтримує сесії та флеш-повідомлення, дозволяючи розробникам створювати інтерактивні та дружні до користувача веб-додатки;

4. Розширення: Завдяки великій кількості доступних розширень, *Flask* може використовувати додаткові функції, такі як *SQL Alchemy* для роботи з базами даних, *Flask-WTF* для роботи з формами, *Flask-Login* для управління сеансами користувачів, і багато інших.

Попри всі переваги, *Flask* має деякі обмеження. Через свою простоту та гнучкість, він може не включати деякі функції "з коробки", які є стандартними в більш великих фреймворках, таких як *Django*. Це може вимагати більше часу та зусиль на налаштування і додавання необхідних функцій через сторонні бібліотеки або розширення.

У підсумку, *Flask* залишається одним з найпопулярніших виборів для розробників Python для створення веб-додатків через свою простоту, гнучкість, велику спільноту та багатий екосистему розширень. Це робить його ідеальним інструментом для широкого спектра веб-проєктів, від простих персональних блогів до складних корпоративних систем.

Система містить 2 вікна і працює наступним чином. На першій сторінці слід ввести всі необхідні фільтри та натиснути кнопку «Шукати» (рис. 3.6)

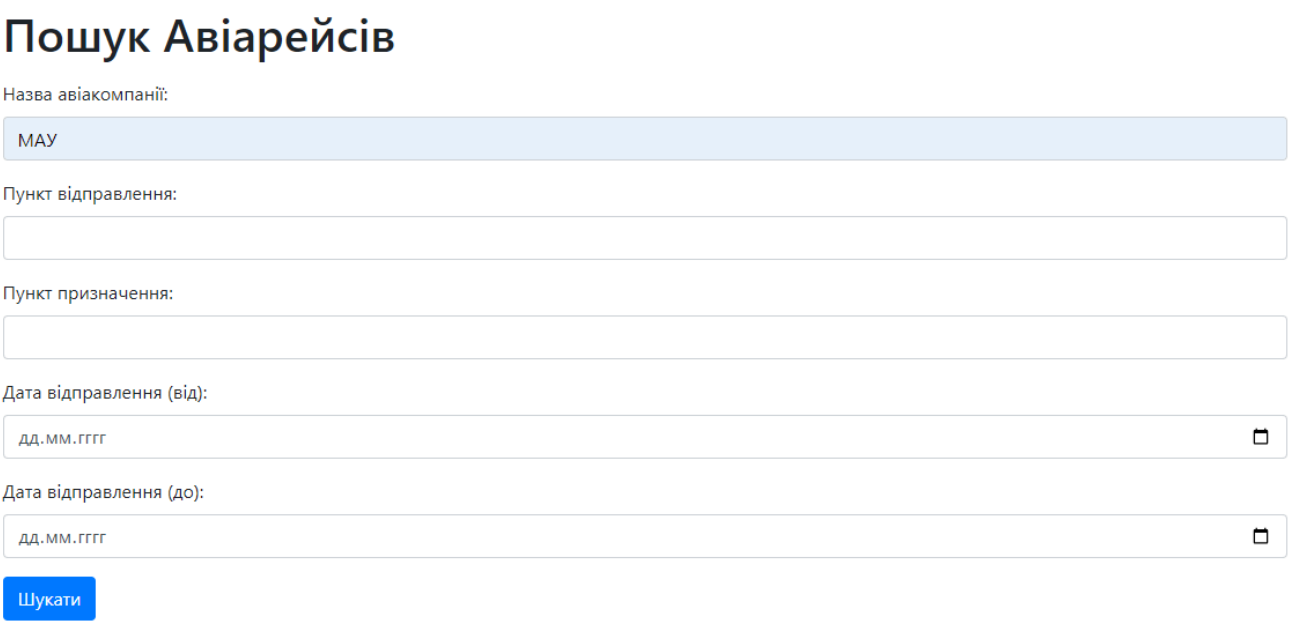
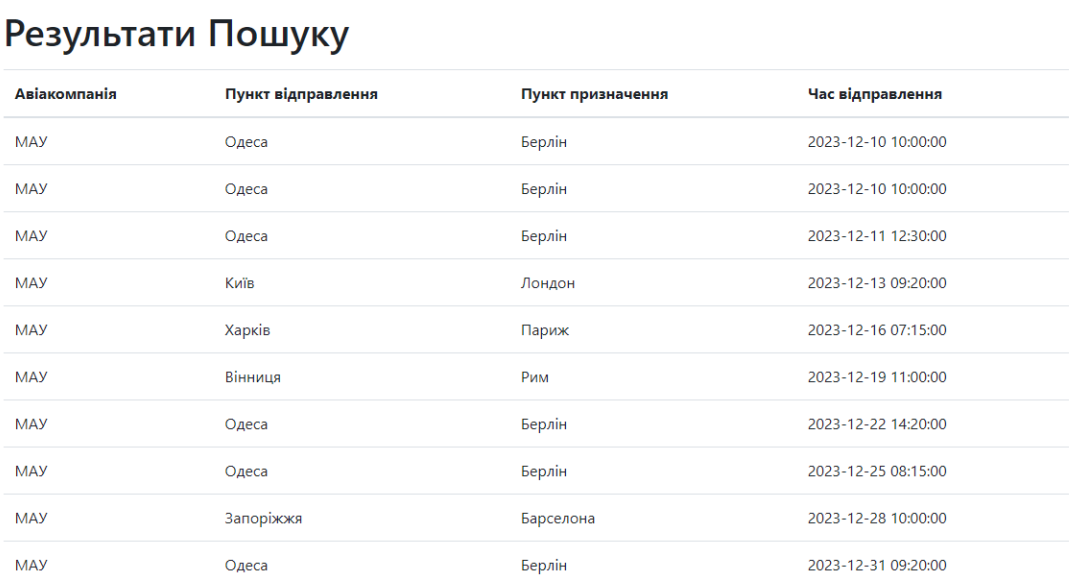


Рис. 3.6. Введення фільтрів

Після натискання кнопки пошуку система знайде всі записи, які відповідають фільтрам (рис. 3.7).



Результати Пошуку

Авіакомпанія	Пункт відправлення	Пункт призначення	Час відправлення
МАУ	Одеса	Берлін	2023-12-10 10:00:00
МАУ	Одеса	Берлін	2023-12-10 10:00:00
МАУ	Одеса	Берлін	2023-12-11 12:30:00
МАУ	Київ	Лондон	2023-12-13 09:20:00
МАУ	Харків	Париж	2023-12-16 07:15:00
МАУ	Вінниця	Рим	2023-12-19 11:00:00
МАУ	Одеса	Берлін	2023-12-22 14:20:00
МАУ	Одеса	Берлін	2023-12-25 08:15:00
МАУ	Запоріжжя	Барселона	2023-12-28 10:00:00
МАУ	Одеса	Берлін	2023-12-31 09:20:00

Рис. 3.7. Результати пошуку

В разі, якщо жодна з записів в базі даних не задовольняє фільтрам – система попередить про це (рис. 3.8).

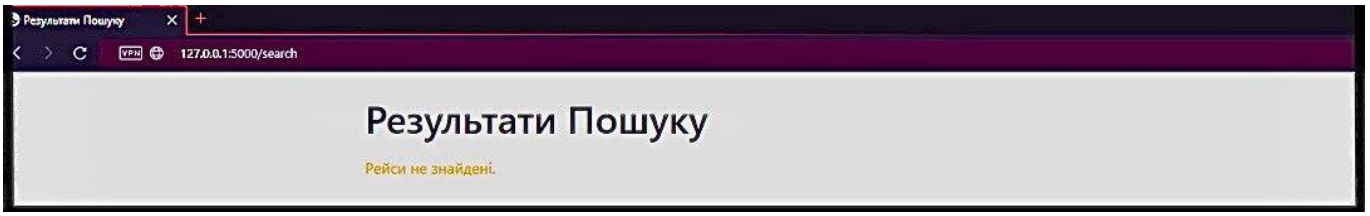


Рис. 3.8. Відсутність підходящих результатів

В таблиці 3.1 представлено порівняння власного рішення з існуючими.

Таблиця 3.1

Порівняння власного рішення з існуючими

Сервіс	Призначення	Основний функціонал	Переваги	Недоліки
Google Flights	Пошук та порівняння авіаквитків	Пошук рейсів, відображення цін у режимі реального часу, оповіщення	Простий інтерфейс, інтеграція з іншими *Google*-продуктами	Обмеженість вибору додаткових послуг, обмежена кількість партнерських авіакомпаній
Skyscanner	Пошук та порівняння авіаквитків, готелів	Великий вибір фільтрів, цінові оповіщення, порівняння пропозицій	Зручний інтерфейс, порівняння цін від різних постачальників	Не завжди точна інформація про доступність квитків
Kayak	Пошук найкращих пропозицій на подорожі	Вибір фільтрів, цінові оповіщення, порівняння пропозицій	Інтуїтивний інтерфейс, порівняння цін від різних постачальників	Не завжди оптимальний вибір опцій для деяких маршрутів
Amadeus	Рішення для авіакомпаній і турагентств	Обробка бронювань, видача квитків, управління даних	Висока надійність, розширена функціональність	Складність та високі витрати на впровадження

Закінчення таблиці 3.1

Сервіс	Призначення	Основний функціонал	Переваги	Недоліки
Sabre	Оптимізація процесів продажу авіаквитків	Обробка бронювань, аналітика, управління запасами, лояльність	Висока масштабованість, ефективність	Складність у використанні для некваліфікованих користувачів
Власний додаток	Пошук інформації про авіакомпанії	Пошук рейсів за різними параметрами, відображення детальної інформації про рейси	Специфічний пошук за параметрами, інтеграція з базою даних	Потребує ручного вводу даних для оновлення бази даних

3.3. Висновки до розділу

В третьому розділі було проведено дослідження і розробку системи пошуку даних про авіакомпанії на основі алгоритму пошуку по ключам. В першу чергу було проведено огляд життєвого циклу і архітектури майбутнього програмного продукту, в ході якого було обрано модель водоспаду і монолітну архітектуру. Наступним кроком було проведено дослідження функціоналу системи і її внутрішньої будови, розроблено основні алгоритми програми та створено графічний інтерфейс.

Бібліотеки, такі як *NumPy*, *SciPy* і *Matplotlib*, роблять *Python* корисним для наукових обчислень, тоді як спеціалізовані бібліотеки, наприклад *Biopython* і *Astropy*, надають можливості, орієнтовані на конкретні галузі. *SageMath-Math* є програмним продуктом з інтерфейсом ноутбука, який може бути програмований на *Python* і охоплює широкий спектр математичних галузей, включаючи алгебру, комбінаторику, числову математику, теорію чисел і обчислення. *OpenCV* має розширені можливості в галузі комп'ютерного зору та обробки зображень.

Python також широко використовується у проєктах зі штучного інтелекту та машинного навчання, включаючи бібліотеки, такі як *TensorFlow*, *Keras*, *Pytorch* і *Scikit-learn*. Завдяки своїй модульній архітектурі, простому синтаксису та багатими інструментами для обробки тексту, *Python* також є популярним для роботи з обробкою природних мов. Крім того, *Python* використовується як мова сценаріїв у багатьох програмних продуктах, включаючи *Abaqus*, *FreeCAD*, програми для *3D*-анімації, програми для обробки зображень та програми для роботи з графічними зображеннями.

Окрім цього було проведено аналітичний огляд інструментальних засобів, які були використані під час розробки програмного продукту, до яких входять *Python*, *PyCharm* та фреймворк *Flask*. Попри всі переваги, *Flask* має деякі обмеження. Через свою простоту та гнучкість, він може не включати деякі функції "з коробки", які є стандартними в більш великих фреймворках, таких як *Django*. Це може вимагати більше часу та зусиль на налаштування і додавання необхідних функцій через сторонні бібліотеки або розширення.

У підсумку, *Flask* залишається одним з найпопулярніших виборів для розробників Python для створення веб-додатків через свою простоту, гнучкість, велику спільноту та багатий екосистему розширень. Це робить його ідеальним інструментом для широкого спектра веб-проєктів, від простих персональних блогів до складних корпоративних систем.

Також було проведено огляд результатів, який показав, що навіть у порівнянні з популярними на поточний момент рішеннями, власна розробка має ряд переваг, які якісно виділяють її серед аналогів.

ВИСНОВКИ

Основні принципи функціонування систем пошуку інформації, які я розглянув, свідчать про значущість цих інструментів у сучасному світі. Незалежно від того, чи ми розглядаємо сервіси для пошуку авіабілетів або інформаційно-пошукові системи для авіаційної галузі, вони спрощують життя користувачів та роблять процес пошуку і аналізу інформації набагато ефективнішим.

Специфічність та функціонал кожного інструменту можуть варіюватися, але загальний принцип полягає в тому, щоб надати користувачам доступ до різноманітних даних та можливостей для прийняття обґрунтованих рішень. Це особливо актуально в авіаційній галузі, де велика кількість інформації про рейси, ціни, наявність квитків і додаткові послуги може здаватися складною для орієнтації.

Переваги використання таких систем включають у собі швидкий доступ до актуальних даних, можливість порівняння пропозицій від різних постачальників, а також спрощення процесу планування подорожей. Для фахівців в авіаційній галузі інформаційні системи надають можливість аналізу ринку та конкурентоспроможності, що є важливим для прийняття стратегічних рішень.

Google Flights та *Skyscanner*, як сервіси пошуку авіаквитків для подорожуючих, надають широкий функціонал та можливості налаштування пошуку, що дозволяє користувачам знаходити оптимальні варіанти та ефективно планувати свої подорожі. Однак важливо враховувати їхні обмеження, такі як обмежена кількість партнерських авіакомпаній та можливості вибору додаткових послуг.

Kayak, з іншого боку, відзначається широким вибором фільтрів та можливістю налаштовування пошуку за різними параметрами, що дозволяє користувачам більш детально контролювати свої пошуки. Але, як і в інших сервісів, існують певні обмеження.

Amadeus та *Sabre*, на відміну від перших трьох, це системи, призначені для авіакомпаній та туристичних агентств з метою оптимізації бізнес-процесів та поліпшення обслуговування клієнтів. Вони надають рішення для управління

бронюваннями, видачі квитків, аналітики та інших аспектів авіаційної галузі. Вони є важливими для великих гравців на ринку авіаперевезень, але вимагають складних інтеграцій та великих витрат.

Кожен з цих сервісів та систем грає важливу роль у спрощенні процесів пошуку та планування подорожей в авіаційній сфері. Користувачі мають можливість вибору сервісу, який найкраще відповідає їхнім потребам та очікуванням. Разом із тим, важливо враховувати, що кожен із них має свої переваги та недоліки, і вибір залежатиме від конкретних цілей користувача. З цим розумінням, користувачі зможуть знайти оптимальний спосіб планування та бронювання своїх майбутніх подорожей у світі авіації.

Звідси випливає, що важливість систем пошуку інформації в авіаційній сфері та інших галузях не може бути переоцінена. Вони сприяють покращенню доступу до інформації, роблячи її більш доступною та зрозумілою для всіх користувачів. Завдяки цим системам, подорожі стають зручнішими та більш доступними, а робота в галузі авіації стає більш ефективною та конкурентоспроможною.

Під час аналізу методів пошуку інформації про авіакомпанії, особлива увага приділялася ефективності та продуктивності алгоритмів, зокрема методу пошуку за ключами. Визначено, що пошук за ключами є важливим елементом розробки веб-додатку для забезпечення оперативного та точного доступу користувачів до інформації про авіакомпанії.

Згідно з проведеним аналізом, важливо підкреслити, що метод пошуку за ключами виявився особливо корисним у веб-додатку для пошуку інформації про авіакомпанії завдяки його здатності оперативно обробляти запити та забезпечувати точний вихідний результат. Застосування цього методу є ключовим для забезпечення надійності та продуктивності системи в умовах постійного оновлення даних про авіакомпанії та широкого спектру користувачів.

Важливим напрямком для подальшого вдосконалення може стати розгляд можливостей комбінування методу пошуку за ключами з іншими продвинутими техніками пошуку, що дозволить збалансувати його переваги та недоліки у різноманітних умовах використання. Такий підхід може сприяти ще більш

ефективному та гнучкому функціонуванню веб-додатку, враховуючи змінність та динамічність сучасного інформаційного середовища.

Переваги методу пошуку за ключами виявилися вельми значущими. Ефективність цього підходу виявляється у швидкому визначенні місця ключа в індексованій базі даних та витяганні необхідної інформації. Завдяки використанню унікальних ключів, алгоритм дозволяє системі точно визначити інформацію про конкретну авіакомпанію, уникнувши зайвого перегляду бази даних та забезпечивши найшвидший можливий час відповіді на запити користувачів.

Однак слід враховувати певні недоліки методу пошуку за ключами. Зокрема, обмеженість структурою даних та залежність від унікальних ключів можуть створити складнощі в обробці складних запитів або у випадках втрати ключа. Крім того, великі обсяги даних можуть призвести до високого навантаження на систему, що вимагає додаткових ресурсів та уваги до оптимізації.

У підсумку, метод пошуку за ключами залишається досить перспективним і ефективним інструментом для побудови веб-додатків, спрямованих на зручний та швидкий доступ до інформації про авіакомпанії. Однак для максимальної ефективності важливо ретельно розробляти та вдосконалювати структуру бази даних, враховуючи можливі складнощі та оптимізуючи алгоритми для оптимальної продуктивності в різних сценаріях використання.

В третьому розділі було проведено дослідження і розробку системи пошуку даних про авіакомпанії на основі алгоритму пошуку по ключам. В першу чергу було проведено огляд життєвого циклу і архітектури майбутнього програмного продукту, в ході якого було обрано модель водоспаду і монолітну архітектуру. Наступним кроком було проведено дослідження функціоналу системи і її внутрішньої будови, розроблено основні алгоритми програми та створено графічний інтерфейс.

Окрім цього було проведено аналітичний огляд інструментальних засобів, які були використані під час розробки програмного продукту, до яких входять *Python*, *PyCharm* та фреймворк *Flask*.

Також було проведено огляд результатів, який показав, що навіть у порівнянні з популярними на поточний момент рішеннями, власна розробка має ряд переваг, які якісно виділяють її серед аналогів

У процесі виконання кваліфікаційної роботи було створено веб-застосунок для пошуку інформації про авіакомпанії, що дозволяє користувачам ефективно знаходити, аналізувати та порівнювати різноманітні авіаційні послуги. Додаток відповідає сучасним тенденціям і вимогам в авіаційній галузі та є зручним інструментом для пасажирів, інвесторів та аналітиків, забезпечуючи швидкий доступ до актуальної інформації.

Розробка веб-застосунку була здійснена з використанням мови програмування *Python* та фреймворка *Flask*, що дозволило інтегрувати потужні інструменти для роботи з даними та створити інтуїтивно зрозумілий інтерфейс. Проєктування бази даних та алгоритмів пошуку було виконано з урахуванням потреб користувачів, що дозволило забезпечити високу точність та швидкість обробки запитів.

У результаті тестування веб-застосунок продемонстрував стабільність і надійність, а також високу швидкість роботи. Оптимізація процесів була здійснена з урахуванням великої кількості одночасних користувачів, що зробило додаток масштабованим і здатним ефективно функціонувати в умовах реального часу.

Додаток має значний потенціал для подальшого розвитку та вдосконалення. Можливості для розширення функціоналу включають інтеграцію з додатковими даними, розширення бази даних і використання штучного інтелекту для підвищення точності прогнозування та персоналізації пропозицій для користувачів.

Таким чином, розроблений веб-застосунок є актуальним та перспективним інструментом для різних категорій користувачів, який сприяє зростанню інформованості та прозорості у сфері авіаційних послуг. Результати проєкту підтверджують його важливість та практичну цінність, а розроблена система може бути рекомендована до широкого впровадження та використання в галузі.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.
2. ДСТУ 3008–95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. – Введ. 1995-23-02. – ІПрІн, УкрІНТЕЇ, Головний відділ стандартизації Технічного центру НАН України, 1995. – *No*58, 88 с.
3. Ноздріна Л., Ящук В., Полотай О. Управління проєктами. Київ: Центр навчальної літератури, 2020. 432 с.
4. *Croft W., Metzler B., Strohman D. Search Engines: Information Retrieval in Practice. Addison-Wesley,* 2010. 624 *p.*
5. *Christopher D., Prabhakar R., Schütze H. Introduction to Information Retrieval. Cambridge University Press,* 2008. 496 *p.*
6. *Grossman D., Frieder O., Karlgren J. Information Retrieval: Algorithms and Heuristics. Springer,* 2004. 292 *p.*
7. *Google Flights. URL: https://www.google.com/travel/flights* (дата звернення: 20.10.2023).
8. *Skyscanner. URL: https://www.skyscanner.com.ua/* (дата звернення: 20.10.2023).
9. *Kayak. URL: https://www.ua.kayak.com/* (дата звернення: 20.10.2023).
10. *Amadeus. URL: https://amadeus.com/* (дата звернення: 20.10.2023).
11. *Sabre. URL: https://www.sabre.com/* (дата звернення: 20.10.2023).
12. *Büttcher S., Charles L., Clarke A., Gordon V. Information Retrieval: Implementing and Evaluating Search Engines,* 2010. 560 *p.*
13. *Garcia-Molina H., Jeffrey D., Widom J. Database Systems: The Complete Book. Pearson,* 2001. 1120 p.
14. *Edelkamp S., Schröd S. Heuristic Search: Theory and Applications. Morgan Kaufmann,* 2011. 392 *p.*
15. *Glover F., Kochenberger G. Introduction to Metaheuristics. Springer,* 2003. 272 *p.*

16. Gormley C., Tong Z. Mastering Elasticsearch: Building a Scalable, Open Source Search Engine. O'Reilly Media, 2013. 354 p.
17. Bacchus, Barua (January 2013). Provincial Healthcare Index 2013 (PDF) (Report). Fraser Institute. p. 25.
18. Professor Raymond Flood. Turing and von Neumann (video). Gresham College – via YouTube.
19. Maschler, Michael; Solan, Eilon; Zamir, Shmuel (2013). Game Theory. Cambridge University Press. pp. 176–180. ISBN 9781107005488.
20. Osborne, Martin J.; Rubinstein, A. (1994). A Course in Game Theory (print ed.). Cambridge, MA: MIT Press. ISBN 9780262150415.
21. Russell, Stuart J.; Norvig, Peter. (2021). Artificial Intelligence: A Modern Approach (4th ed.). Hoboken: Pearson. pp. 149–150. ISBN 9780134610993. LCCN 20190474.
22. Hsu, Feng-Hsiung (1999). "IBM's Deep Blue chess grandmaster chips". IEEE Micro. Los Alamitos, CA, USA: IEEE Computer Society. 19 (2): 70–81. doi:10.1109/40.755469. During the 1997 match, the software search extended the search to about 40 plies along the forcing lines, even though the non-extended search reached only about 12 plies.
23. Noam Chomsky and John Halle, "An Eight Point Brief for LEV (Lesser Evil Voting)," New Politics, June 15, 2016.
24. Rawls, J. (1971). A Theory of Justice. p. 152.
25. Arrow, K. (May 1973). "Some ordinalist-utilitarian notes on Rawls's Theory of Justice". Journal of Philosophy. 70 (9): 245–263.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
#Імпорт бібліотек
from flask import Flask, request, render_template
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
#Створення аплікації
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///airlines.db'
db = SQLAlchemy(app)
#Модель авіалінії
class Airline(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    flights = db.relationship('Flight', backref='airline', lazy=True)
#Модель маршруту
class Route(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    origin = db.Column(db.String(50), nullable=False)
    destination = db.Column(db.String(50), nullable=False)
    flights = db.relationship('Flight', backref='route', lazy=True)
#Модель авіарейсу
class Flight(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    airline_id = db.Column(db.Integer, db.ForeignKey('airline.id'), nullable=False)
    route_id = db.Column(db.Integer, db.ForeignKey('route.id'), nullable=False)
    departure_time = db.Column(db.DateTime, nullable=False)
#Створення моделей в БД
with app.app_context():
```

```
    db.create_all()
#Маршрутизація головної сторінки
@app.route('/')
def index():
    return render_template('index.html')
#Маршрутизація пошуку
@app.route('/search', methods=['GET'])
def search():
    airline_name = request.args.get('airline_name')
    origin = request.args.get('origin')
    destination = request.args.get('destination')
    date_from = request.args.get('date_from')
    date_to = request.args.get('date_to')
    query = Flight.query
    if airline_name:
        query = query.join(Airline).filter(Airline.name == airline_name)
    if origin and destination:
        query = query.join(Route).filter(Route.origin == origin, Route.destination ==
destination)
    if date_from and date_to:
        date_from = datetime.strptime(date_from, '%Y-%m-%d')
        date_to = datetime.strptime(date_to, '%Y-%m-%d')
                    query = query.filter(Flight.departure_time >= date_from,
Flight.departure_time <= date_to)
    flights = query.all()
    return render_template('search_results.html', flights=flights)
#Запуск додатку
if __name__ == '__main__':
    app.run(debug=True)
```