МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Веб-сервіс автоматизації пошуку літератури

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма **«Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-42
_____Полосенко П. О._____
(прізвище та ініціали)

Керівник _____Сайко В. Г._____
(прізвище та ініціали)
_____д.т.н., професор_____
(науковий ступінь, звання)
Унікальність тексту 86% (https://my.plag.com.ua/)

Випускна кваліфікаційна робота бакалавра допущена до захисту
Рішенням кафедри *прикладних інформаційних систем*
Протокол № 14 від 23.05.2023 р.
зав. кафедри _____ Плескач В. Л.

Київ – 2023

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра прикладних інформаційних систем

Назва теми: «Веб-сервіс автоматизації пошуку літератури»

Освітня програма: Прикладне програмування

Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Полосенко Павло Олегович	

Назва роботи українською та англійською мовами:

Веб-сервіс автоматизації пошуку літератури
Web service for literature search automation

Мета дипломної роботи: є підвищення ефективності пошуку літератури в інтернеті за допомогою веб-сервісу автоматизації пошуку літератури.
План роботи: 1. Сучасні методи розробки веб-сервісів 2. Аналіз архітектурних рішень, вибір технологій для реалізації веб-сервісу 3. Реалізація веб-сервісу автоматизації пошуку літератури

ПІБ, ступінь, звання наукового керівника роботи:

Сайко Володимир Григорович, д.т.н., професор

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	14.10.2022	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	24.10.2022	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	31.10.2022	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.11.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2023	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2023	Виконано
9.	Подання роботи у першому варіанті	28.04.2023	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2023	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**22.05.2023**	Виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	26.05.2023	Виконано
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	12.06.2023	Виконано
14.	Захист кваліфікаційної роботи бакалавра	29.06.2023	

Здобувач вищої освіти _____

(підпис)

Керівник _____

(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Завдання до дипломної роботи	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік умовних позначень і скорочень	1
Вступ	2
1	16
2	17
3	12
Висновки	1
Перелік використаних джерел	2

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата.			
Розробн.	Полосенко П. О.			Відомість Дипломної роботи	Лист	Листів
Керівн.	Сайко В. Г.					
Н/контр.	Кравченко К.В.					
Зав.каф.	Плескач В. Л.					

АНОТАЦІЯ

Ця дипломна робот а присвячена проектуванню та розробленню веб-сервісу автоматизації пошуку літератури.

Метою дипломної роботи є підвищення ефективності пошуку літератури в інтернеті за допомогою веб-сервісу автоматизації пошуку літератури.

Для досягнення поставленої мети треба вирішити такі завдання:

- дослідити сучасні методи розробки веб-сервісів
- проаналізувати архітектурні рішення та обрати технології для реалізації веб-сервісу
- спроектувати, реалізувати веб-сервіс автоматизації пошуку літератури

Об’єкт дослідження

Процес надання автоматизованого пошуку літератури в інтернеті.

Предмет дослідження

Програмно-технічні, організаційні засади, принципи, підходи щодо побудови веб-сервісу автоматизації пошуку літератури.

Методи дослідження

Емпіричний аналіз і синтез систем, що застосовувався при вивченні прикладів сучасних методів побудови веб-сервісів, UML-моделювання, аналогія залучені в процесі проектування, розробки та побудови власного веб-сервісу, метод порівняння, що застосовано для аналізу наявних ресурсів та програмних систем веб-сервісів.

Ключові слова: веб-сервіс, література, автоматизація.

ABSTRACT

This thesis is devoted to the design and development of a literature search automation web service.

The purpose of the thesis is to increase the efficiency of searching for literature on the Internet with the help of a web service for automating the search for literature.

To achieve the goal, the following tasks must be solved:

• explore modern methods of web services development

• analyze architectural solutions and choose technologies for web service implementation

• to design and implement a web service for literature search automation

Object of study

The process of providing an automated literature search on the Internet.

Subject of study

Software and technical, organizational principles, principles, approaches to building a literature search automation web service.

Research methods

Empirical analysis and synthesis of systems used in the study of examples of modern methods of building web services, UML modeling, analogy involved in the process of designing, developing and building one's own web service, the comparison method applied to the analysis of available resources and software systems of web services.

Keywords: web service, literature, automation.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

БД – база даних

API (Application Programming Interface) - інтерфейс програмування додатків

XML (Extensible Markup Language) - розширювана мова розмітки

HTTP (HyperText Transfer Protocol) - протокол передачі гіпертексту

ВСТУП

В сучасному світі доступ до інформації став необхідністю для багатьох людей. Особливо актуальною є проблема пошуку літератури, яка є основою для вивчення, розвитку та збереження знань. У зв'язку з цим виникає потреба в створенні ефективних засобів автоматизації пошуку літератури. Один із способів автоматизації пошуку літератури - це створення веб-сервісу, який дозволить користувачам швидко та ефективно знайти потрібну літературу в Інтернеті. Такий сервіс може забезпечувати різноманітні функції, такі як пошук за ключовими словами, вибір релевантних джерел, збереження пошукових запитів та інші. Створення такого веб-сервісу є складним завданням, яке вимагає розробки технічного завдання, вибору оптимальних технологій розробки та проведення тестування на коректність роботи та швидкодію. Дослідження цієї теми є важливим для того, щоб розробити веб-сервіс, який буде максимально ефективним та зручним для користувачів.

Актуальність цієї теми зумовлено тим, що цифрові технології швидко розвиваються. Створення веб-сервісу автоматизації пошуку літератури обумовлена необхідністю вдосконалення процесу пошуку літератури та забезпечення доступу до неї. Інформаційне суспільство потребує ефективних засобів пошуку інформації. У сучасному світі, де інформаційні технології швидко розвиваються, пошук літератури стає все більш складним та часоємним процесом. Наявність великої кількості джерел та форматів публікацій, різноманітність мов та тематик робить пошук джерел надзвичайно складним завданням.

Метою дипломної роботи є підвищення ефективності пошуку літератури в інтернеті за допомогою веб-сервісу автоматизації пошуку літератури.

Для досягнення поставленої мети треба вирішити такі завдання:

- дослідити сучасні методи розробки веб-сервісів
- проаналізувати архітектурні рішення та обрати технології для реалізації веб-сервісу
- спроектувати, реалізувати веб-сервіс автоматизації пошуку літератури

Об’єкт дослідження

Процес надання автоматизованого пошуку літератури в інтернеті.

Предмет дослідження

Програмно-технічні, організаційні засади, принципи, підходи щодо побудови веб-сервісу автоматизації пошуку літератури.

Методи дослідження

Емпіричний аналіз і синтез систем, що застосовувався при вивченні прикладів сучасних методів побудови веб-сервісів, UML-моделювання, аналогія залучені в процесі проектування, розробки та побудови власного веб-сервісу, метод порівняння, що застосовано для аналізу наявних ресурсів та програмних систем веб-сервісів.

Практичне значення одержаних результатів Полягає у тому, що розроблений веб-сервіс може стати корисним, зручним та сучасним рішенням проблеми пошуку літератури в інтернеті. Розроблений веб-сервіс може бути застосований для більш ефективного та зручного пошуку літератури в інтернеті.

Структура роботи:

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів розподілених на підрозділи та висновку.

РОЗДІЛ 1

СУЧАСНІ МЕТОДИ РОЗРОБКИ ВЕБ-СЕРВІСІВ

1.1 Проблема пошуку літератури в сучасному світі

1.1.1 Розвиток літератури

Писемність це один з найдавніших способів передачі інформації, який з часом став одним з основних. Перша писемність виникла у вигляді невеличких піктограм. До першої піктографічної стадії писемності відносять 3500 р. до н. е. Далі писемність перейшла у вигляд ієрогліфів, вперше ієрогліфічна писемність з'явилась в Єгипті 3200 р. до н. е.. 3100 р. до н. е. виникає найдавніший відомий алфавіт. Але роком виникнення першої справжньої писемності вважається 2750р. до н. е., виникла вона в Шумері. З цього часу символьна писемність починає розвиватись з неймовірною швидкістю. З'являються все більше і більше різних видів писемності одні налічують більше букв і звуків інші менше. В 0 р. в Китаї винаходять папір а вже в 100 р. з'являється рукописний варіант (курсив – це спосіб написання шрифту, коли основні штрихи нахилені справа наліво зверху донизу) в латинській мові. 200 р. унціальне письмо латинського алфавіту. Саме починаючи з 200 років з'являються знайомі для нас книжки, проте написані були вони вручну. Перший друкарський станок з'явився лише в 1448-1455 р. винайшов його Йоганн Гутенберг в Німеччині, його робота полягала у тому, що він виготовляв з металу «рухливі» опуклі літери, відлиті в оберненому вигляді, набирав з них рядки та за допомогою спеціального ручного преса відтискував на папері. Література продовжувала набирати популярність. В наш час паперова література користується все меншим і меншим попитом, все менше і менше людей читає. Деякі люди

вдаються до так званих електронних книжок. Це пристрій який дозволяє завантажити в нього книгу спеціального формату і читати змінюючи розмір шрифту і так далі. Такий спосіб читання вважається менш шкідливим ніж читання з екрану комп'ютера або телефона, оскільки електронні книжки зазвичай або не світяться або виділяють менше світла.

1.1.2 Розвиток літератури в Україні

Першими пам'ятками української літератури були перекладені з грецької біблійні тексти та апокрифічні оповідання. З 11 ст. під впливом візантійських хронік започатковано літописання. Видатна пам'ятка історіографічної та оповідної прози — «Повість минулих літ», яку традиційно приписують Несторові Літописцю. Наприкінці 12 століття з'явилось «Слово о полку Ігоревім». В другій половині 16-го століття почався новий період історії української літератури. Цей період відзначився становленням нових жанрів а саме: полемічної публіцистики, шкільної драматургії, історичної прози, мемуарно-історичної прози. Поширенню цих жанрів сприяло книгодрукування яке вже на той час з'являлось в Україні і суміжних землях. Приблизно від середини 16 століття починається книжне віршування. Першим твором нової української літератури стала «Енеїда» Івана Котляревського яка була написана в 1798-у році. Література протягом всього часу існування України відігравала дуже важливу роль. Проте саме в часи гніту української культури її роль була незамінною. Багато істориків вважають що саме через літературу Україна змогла зберегти свої звичаї і традиції. І в сьогоденні література відіграє важливу роль, багато військових пишуть твори, для того щоб хоч якось відволіктись від війни і передати свої емоції читачу. Проте багато таких творів залишаються непрочитаними, саме для цього потрібний ефективний спосіб пошуку літератури

1.1.3 Необхідність ефективного пошуку літератури

Для більшості людей у всьому світі наша перша серйозна зустріч з літературою походить зі школи. Читати й писати ми навчаємося з раннього дитинства, і це починає діяти з початком іспитів.

Здатність співпереживати групі персонажів, написаних на сторінці, є категоричною та з точки зору студента необхідною навичкою. Крім того, здатність відчувати теми та повідомлення відкриває нам інший спосіб мислення. Література стає посудиною. 130 мільйонів книг, які були опубліковані в усьому світі, є путівниками для читача та створюють місток, щоб він дізнався щось нове.

Історія — це не лише ворота в минуле, вона також свідчить про наше сьогодення та майбутнє. У кожному періоді часу є різні люди, а всередині них різні етапи нашої постійно зростаючої культури. Кожна людина раніше була продуктом свого часу. Як вид ми еволюціонуємо щодня, і без цієї позначки часу, яку дає нам література, ми б нічого не знали про минуле.

Основним призначенням літератури є передача інформації, проте передача інформації значно ускладняється якщо немає ефективного способу її пошуку. Враховуючи що все менше людей використовують літературу, продавці не зацікавлені в продажі книг. Особливо це стосується літератури яку купляють найрідше а саме наукової. Веб-сервіс з автоматизації пошуку літератури може вирішити цю проблему також він може пришвидшити час пошуку любої літератури. Згідно з дослідженнями, сервіси пошуку товарів і порівняння цін можуть знизити вартість покупок в середньому на 10-15%, що може бути великим заощадженням для споживачів.

Отже ефективність послуг пошуку товарів і порівняння цін значно спрощує процес вибору товару, скорочує час, витрачений на пошук і порівняння товарів у різних магазинах. Завдяки сервісам порівняння цін користувачі можуть знайти найкращу пропозицію для конкретного товару та

заощадити гроші. Загалом послуги пошуку товарів і порівняння цін допомагають спростити процес пошуку та вибору, скоротити час і витрати та надати споживачам більш обґрунтований вибір.

1.2 Визначення веб-сервісу та етапи розвитку веб-сервісів в Україні

Різні книги та різні організації дають різні визначення веб-сервісів. Деякі з них перераховані тут.

- Веб-сервіс – це будь-яка частина програмного забезпечення, яка стає доступною через Інтернет і використовує стандартизовану систему обміну повідомленнями XML. XML використовується для кодування всіх повідомлень до веб-служби. Наприклад, клієнт викликає веб-службу, надсилаючи повідомлення XML, а потім чекає відповідної відповіді XML. Оскільки весь зв’язок відбувається в форматі XML, веб-сервіси не прив’язані до однієї операційної системи чи мови програмування — Java може спілкуватися з Perl; Програми Windows можуть спілкуватися з програмами Unix.
- Веб-сервіси — це автономні, модульні, розподілені, динамічні програми, які можна описувати, публікувати, локалізувати або викликати в мережі для створення продуктів, процесів і ланцюжків поставок. Ці програми можуть бути локальними, розподіленими або веб-базованими. Веб-сервіси побудовані на основі відкритих стандартів, таких як TCP/IP, HTTP, Java, HTML і XML.
- Веб-служби — це системи обміну інформацією на основі XML, які використовують Інтернет для прямої взаємодії між програмами. Ці системи можуть включати програми, об'єкти, повідомлення або документи.

- Веб-служба — це набір відкритих протоколів і стандартів, які використовуються для обміну даними між програмами або системами. Програмні програми, написані на різних мовах програмування та запущені на різних платформах, можуть використовувати веб-сервіси для обміну даними через комп'ютерні мережі, такі як Інтернет, у спосіб, подібний до міжпроцесного зв'язку на одному комп'ютері. Ця сумісність (наприклад, між програмами Java і Python або Windows і Linux) зумовлена використанням відкритих стандартів.

  Таким чином, повний веб-сервіс – це будь-який сервіс, який –

- Доступний через Інтернет або приватні (інтернет) мережі.
- Використовує стандартизовану систему обміну повідомленнями XML.
- Не прив'язаний до однієї операційної системи чи мови програмування.
- Самоописується за допомогою загальної граматики XML.
- Його можна знайти за допомогою простого механізму пошуку.

Як і більшість провідних країн світу веб-сервіси почали бути популярними в Україні протягом останніх десятиліть. Процес розвитку веб-сервісів можна поділити на три основні етапи:

1. Початок 2000-х років – це саме початок розвитку веб-сервісів в Україні. На цьому етапі компанії тільки почали розробляти перші веб-сервіси, які були спрямовані на автоматизацію бізнес-процесів, підвищення ефективності роботи та забезпечення доступу до даних з різних пристроїв. Саме в цей час вже почали використовувати XML як основний формат передачі даних між веб-сервісами. Також почали використовуватися такі стандарти як: WSDL(Web Services Description Language) SOAP(Simple Object Access Protocol) для опису веб-сторінки та їх виклику.

2. Кінець 2000-х років – середина 2010-х – період інтенсивного розвитку веб-сервісів в Україні. Ринок веб-розробок на цьому етапі став набагато більше конкурентним. Компанії почали використовувати більше веб-технологій включаючи хмарні сервіси та інші. В цей час з'явилась REST(Representational State Transfer) архітектура, яка стала альтернативою для SOAP. REST використовує формат JSON для передачі даних, а також HTTP-запити для виклику веб-сервісів.
3. Середина 2010-х років – сьогодення – цей етап це етап вдосконалення веб-сервісів в Україні. На цьому етапі компанії розробляють все більш складні та інноваційні веб-сервіси. Для розробки веб-сервісів уже використовуються такі інструменти як: штучний інтелект, блокчейн та інші. В цей час розвивається технологія використання багатьох взаємодіючих між собою мікросервісів. Для взаємодії між мікросервісами використовують веб-сервіси на основі REST та JSON.

1.3 Види та переваги веб-сервісів

Існує велика кількість різних веб-сервісів і вона постійно збільшується в залежності від потреб тут я наведу основні види веб сервісів:

1. Дошки оголошень. Цей вид веб сервісів дозволяє користувачам можливість розміщувати або шукати оголошення наприклад з продажу або купівлі товарів і послуг. Мінімальний набір функцій дощок оголошень це: додавання/видалення оголошень, пошук, фільтрація за кількома параметрами і класифікація за рубриками. Також додатково можна реалізувати систему рейтингів, коментарі, чат і так далі.
2. Каталоги. Каталоги вступають в ролі онлайн вітрини для продукції підприємства це може бути як каталог для товарів так і для послуг. Також бувають галузеві директорії, на яких розміщуються пропозиції від різних компанії певної індустрії. Мінімальний набір функцій каталогів це: додавання/видалення товарів, пошук, фільтрація, розбиття на категорії,

кошик, оплата. Також додатково можна реалізувати відгуки користувачів, додаткову детальну інформацію про товар, відстеження замовлення.

3. Пошук товарів і порівняння цін. Веб-сервіси для пошуку товарів і порівняння цін збирають інформацію з інтернет-магазинів, каталогів і маркетплейсів на одній платформі. Вони допомагають користувачам обирати найкращого продавця, по товару або послугах що їх цікавлять. Порівнювати пропозиції можна за ціною або відгуками. Мінімальний набір функцій включає: пошук, фільтрація за кількома параметрами, порівняння цін, можливість додавання товарів до списку бажань, реєстрація користувача, можливість переглянути сторінку товару. Також додатково можна реалізувати нагадування про зміну ціни на товар та отримання сповіщень про акції і знижки від продавців.
4. Рішення для бізнесу. Рішення для автоматизації бізнес процесів можна винести в окрему категорію. Існує безліч способів посилити бізнес за допомогою цифрових технологій наприклад: CRM система управління взаємодією з клієнтами, MES система управління виробництвом, PM система управління проектами та ERP система управління ресурсами підприємства. Також сюди можна віднести програми для електронного документообігу та фінансового обліку. Мінімальний набір функцій зазвичай відрізняється в залежності від потреб.
5. Маркетпейси. На маркетплейсах розміщуються товари і послуги віб безлічі постачальників. Зазвичай власник або оператор маркетплейса не займається продажем своїх товарів, його основною метою є створення зручної площадки та сприяння умовам для успішної торгівлі. Мінімальний набір функцій може включати в себе: реєстрація користувачів, пошук товарів, система оплати, корзина, можливість переглядати профіль продавця та можливість залишати відгуки. Також додатково можна реалізувати чат з продавцем та можливість повернення товару.

6. Системи бронювання. Системи бронювання дають користувачу можливість обрати кращу пропозицію і зарезервувати її для себе наприклад квиток на літак, поїзд, тощо або номер в готелі або тур. Такі системи поєднують в собі функціонал системи з пошуку товарів і порівняння цін і сайту електронної комерції. Також бувають внутрішні системи бронювання які обслуговують одну компанію. До мінімального функціоналу можна віднести: систему реєстрації користувача, можливість порівняння цін, можливість бронювання пропозицій, фільтрація, пошук, можливість перейти на сторінку пропозиції, інформація про надавача послуг. Також додатково можна реалізувати систему рейтингів, чат.

Також варто зазначити переваги використання веб-сервісів. Нижче перераховані основні переваги використання веб сервісів.

- Висока масштабованість. Веб-сервіси підключаються до ядра додатка як окремі модулі. Це дозволяє швидко масштабувати IT-проект без необхідності вносити глобальні зміни в основний код.
- Спрощена взаємодія. Одне з ключових переваг веб-сервісній архітектури - злагодженість обміну даними між модулями. При цьому компоненти програми можуть бути написані на різних мовах і під різні операційні системи. Наприклад, до Java-сервера, що працює під Linux, можна підключити Windows (C #) і MacOS (Swift) клієнти.
- Проста інтеграція. За рахунок застосування універсального протоколу веб-проект можна підключити відразу до декількох додатків і сайтів. Це дозволяє автоматизувати внутрішній обмін даними на підприємстві або розширити функціонал завдяки інтеграції з зовнішніми сервісами.
- Зв’язок через HTTP. Веб-служби працюють за допомогою протоколу обміну повідомленнями SOAP найпоширенішого в глобальній мережі

протоколу - HTTP. Таким чином при розробці та налаштування веб-сервісів можна використовувати звичайний інтернет.

- Єдиний стандарт передачі даних. Веб-служби використовують стандартний чотирьохрівневий протокол для передачі даних. Уніфікований мережевий стек веб-служб спрощує налаштування обміну даними між різними веб-сервісами та дає гнучкі можливості для інтеграції.
- Універсальність рішень. Окремі компоненти системи можна розробляти на різних мовах програмування. При цьому модулі можуть знаходитися на будь-яких вузлах мережі, незалежно від того, де лежить основна програмна частина. Така гнучкість веб-сервісної архітектури дозволяє вибирати оптимальний мову на кожній ділянці проекту.

Таким чином можна зробити висновок, що використання веб-сервісів має досить велику кількість переваг. Веб-сервіс є ефективним і гнучким інструментом для побудови складних програмних систем і автоматизації бізнес-процесів. Використання веб-сервісів може значно спростити розробку та налаштування програмних продуктів та покращити їх ефективність.

1.4 Технології розробки веб-сервісів

Веб-розробка містить величезний набір правил і прийомів, про які повинен знати кожен розробник веб-сайтів. Важливо правильно підібрати технологій та інструменти веб-розробки зважаючи на масштаб проекту, складність, функціонал який має бути реалізований а також на потреби користувачів. Основні технології веб розробки це:

- HTML(Hypertext Markup Language) – це стандартизована мова розмітки документів для перегляду веб-сторінок у браузері. Відповідно браузери

отримують документ HTML за протоколами HTTP/HTTPS і інтерпретують код в інтерфейс з яким буде взаємодіяти користувач.

- CSS(Cascading Style Sheets) – це спеціальна мова яка використовується в парі з HTML. Це невід'ємна частина веб-дизайну. Ця мова робить приємним для перегляду сторінки сайту а також частково відповідає за анімацію різних об'єктів написаних мовою HTML.
- JavaScript – це динамічна об'єктно орієнтована мова програмування. На цій мові пишуться скрипти, вона використовується для взаємодії користувача з сайтом. Також вона дозволяє обмінюватись даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.
- PHP(Hypertext Preprocessor) – скриптова мова програмування яка була створена для генерації HTML-сторінок. Часто використовується разом з JavaScript. В PHP вбудовані багато бібліотек для роботи з MySQL, PostgreSQL, SQLite і так далі.
- Python – це інтерпретована об'єктно-орієнтована мова програмування. Надзвичайно гнучка мова програмування яка дозволяє створювати веб-додатки та веб-сервіси з використанням фреймворків, таких як Django та Flask.
- Ruby – це інтерпретована об'єктно-орієнтована мова програмування. Ця мова є високоефективною в розробці програм. Також на цій мові можна створювати веб-додатки та веб-сервіси використовуючи фреймворки, такі як Ruby on Rails.
- REST(Representational State Transfer) – це архітектурний стиль в який закладено принципи функціонування всесвітньої павутини. Він використовує HTTP-запити для виклику веб-сервісів а також формат JSON для передачі даних.
- GraphQL - це мова запитів для API та середовище виконання для виконання цих запитів із наявними даними. GraphQL надає повний і

зрозумілий опис даних у вашому API, дає клієнтам можливість вимагати саме те, що їм потрібно, і нічого більше, полегшує розвиток API з часом і надає потужні інструменти розробника.

- Docker - набір інструментів для керування ізольованими контейнерами Linux. Docker доповнює набір інструментів LXC API вищого рівня, який дозволяє керувати контейнерами на окремих рівнях ізоляції процесу. Таким чином, Docker дозволяє запускати довільні процеси ізольовано, не турбуючись про вміст контейнера, а потім передавати та клонувати контейнери, створені для цих процесів, на інші сервери, виконуючи роботу зі створення, підтримки та обслуговування контейнерів.
- Kubernetes - відкрите програмне забезпечення для оркестрування контейнеризованих програм - автоматизації їх розгортання, масштабування та координації в умовах кластера. Підтримує основні технології контейнеризації, включаючи Docker, rkt, можлива також підтримка технологій апаратної віртуалізації.
- SQL – це мова структурованих запитів. Це мова програмування, яка використовується для керування реляційними базами даних. SQL дозволяє користувачам створювати, змінювати та запитувати бази даних, а також виконувати різні операції, такі як вставка, оновлення, видалення та отримання даних.

1.5 Мікросервісна архітектура

До сучасних методів розробки веб-сервісів можна віднести мікросервісну архітектуру. Мікросервісна архітектура — це підхід до створення програмних додатків як набору невеликих незалежних сервісів, кожен з яких відповідає за виконання певної функції. Ці служби взаємодіють одна з одною через API, їх можна розробляти та розгортати незалежно одна від одної, що забезпечує більшу гнучкість і масштабованість.

Кожен мікросервіс зазвичай працює у своєму власному процесі та може бути написаний на різних мовах програмування або з використанням різних технологій, якщо вони спілкуються один з одним через стандартизовані інтерфейси. Це дозволяє використовувати найкращі у своєму класі технології для кожної конкретної функції та дозволяє командам швидше розробляти та розгортати нові функції.

Деякі з ключових переваг архітектури мікросервісів включають покращену масштабованість, стійкість і відмовостійкість, а також можливість швидше й ефективніше впроваджувати інновації. Однак розробка та реалізація архітектури мікросервісів може бути складною, оскільки вимагає ретельного планування та координації між різними командами та службами.

1.6 RESTful API

RESTful API — це архітектурний стиль для створення веб-служб, які використовують HTTP для зв’язку між клієнтами та серверами. REST означає Representational State Transfer, що означає, що кожен запит, зроблений клієнтом серверу, містить усю необхідну інформацію для виконання потрібної дії, а сервер надсилає відповідь із результатами цієї дії.

RESTful API зазвичай використовують методи HTTP (такі як GET, POST, PUT і DELETE), щоб представити різні дії, які можна виконати над ресурсом, який ідентифікується унікальною URL-адресою (також називається ідентифікатором ресурсу або URI). Наприклад, клієнт може надіслати запит GET, щоб отримати інформацію про певний ресурс, або запит POST, щоб створити новий ресурс.

RESTful API не мають стану, що означає, що кожен запит містить усю необхідну інформацію для виконання запитаної дії, а сервер не підтримує жодного клієнтського стану між запитами. Це робить RESTful API високо

масштабованими та простими для кешування, оскільки відповіді можуть кешуватися проміжними серверами для підвищення продуктивності.

Деякі з ключових переваг RESTful API включають їхню простоту, гнучкість і масштабованість, а також їх широку підтримку в структурах веб-розробки та бібліотеках. Однак розробка та впровадження RESTful API також може бути складним завданням, оскільки вимагає ретельного розгляду ресурсів, дій і форматів даних, які будуть використовуватися, а також належного використання методів HTTP та кодів стану.

1.7 GraphQL

GraphQL — це мова запитів і середовище виконання для створення API, розроблене Facebook. Він надає більш ефективну, потужну та гнучку альтернативу RESTful API для отримання та оновлення даних.

За допомогою GraphQL клієнти можуть точно вказати, які дані їм потрібні, а сервер відповідає лише цими даними. Це забезпечує більш ефективну передачу даних, оскільки клієнтам не потрібно отримувати непотрібні дані. Крім того, GraphQL дозволяє клієнтам отримувати дані з кількох ресурсів за допомогою одного запиту.

Однією з ключових особливостей GraphQL є його система типів, яка дозволяє розробникам визначати структуру даних, які можна запитувати. Це дозволяє таким інструментам, як GraphiQL, надавати розробникам можливість дослідження схем і автозаповнення запитів. Крім того, оскільки схема строго типізована, розробникам легше зрозуміти структуру даних і уникнути помилок.

GraphQL також надає API мутації для оновлення даних на сервері. Це дозволяє клієнтам виконувати складні операції на сервері за допомогою одного запиту та забезпечує послідовний спосіб обробки помилок і перевірки вхідних даних.

Деякі з переваг використання GraphQL включають його гнучкість, ефективну передачу даних і потужні можливості запитів. Однак розробка та впровадження GraphQL API може бути складним завданням, особливо коли мова йде про керування складними зв’язками даних і вимогами до авторизації/автентифікації.

1.8 Безсерверна архітектура

Безсерверна архітектура — це модель хмарних обчислень, у якій хмарний провайдер керує інфраструктурою, необхідною для запуску та масштабування програм, і стягує плату на основі фактичного використання ресурсів, а не фіксованої суми. У безсерверній архітектурі розробники пишуть функції, які виконуються на вимогу, не вимагаючи надання або керування серверами чи інфраструктурою.

Коли функція запускається, хмарний провайдер автоматично виділяє необхідні ресурси для її запуску та масштабує ці ресурси вгору або вниз відповідно до вимог. Це дозволяє розробникам зосередитися на написанні та розгортанні коду, не турбуючись про базову інфраструктуру, і дозволяє їм створювати високомасштабовані та стійкі додатки, які можуть справлятися з непередбачуваними навантаженнями.

Деякі з ключових переваг безсерверної архітектури включають зменшення операційних витрат, нижчі витрати та підвищену масштабованість і відмовостійкість. Оскільки функції виконуються лише після запуску, немає потреби платити за неактивні ресурси, і розробники можуть легко збільшувати або зменшувати масштаб залежно від попиту.

Однак існують також деякі проблеми, пов’язані з безсерверною архітектурою, як-от необхідність ретельного керування залежностями, обробки холодних запусків (які можуть статися, коли функція запускається вперше) і забезпечення належної безпеки та відповідності. Крім того, безсерверні функції

найкраще підходять для короткочасних операцій без збереження стану та можуть бути не найкращим вибором для тривалих або інтенсивних обчислювальних завдань.

1.9 Контейнеризація та оркестрування

Контейнерізація — це технологія, яка дозволяє розгортати та запускати програми в легких ізольованих середовищах, які називаються контейнерами. Контейнери забезпечують спосіб упакувати програму та всі її залежності в єдиний портативний блок, який можна легко переміщувати між середовищами та працювати узгоджено на різних платформах.

Оркестровка, з іншого боку, відноситься до автоматизованого керування та масштабування контейнерних програм у кластері хостів. Інструменти оркестровки дозволяють розробникам розгортати, керувати та масштабувати контейнери на кількох хостах, полегшуючи керування та масштабування складних програм.

Деякі популярні технології контейнеризації включають Docker і Kubernetes. Docker — це платформа для створення, доставки та запуску додатків у контейнерах, а Kubernetes — це платформа оркестровки контейнерів з відкритим кодом, яка автоматизує розгортання, масштабування та керування контейнерними додатками.

У типовій контейнеризованій архітектурі програма розбивається на кілька контейнерів, у кожному з яких працює певний компонент програми. Наприклад, веб-програма може мати контейнери для компонентів інтерфейсу, серверної частини та бази даних. Ці контейнери можна розгортати та масштабувати незалежно один від одного, що полегшує керування та масштабування програми в цілому.

Такі інструменти оркестровки, як Kubernetes, надають такі функції, як автоматичне балансування навантаження, автоматичне масштабування на основі використання ресурсів, а також автоматичне розгортання та відкат, що полегшує керування та масштабування контейнерних програм у кластері хостів.

1.10 Аналіз подібних програмних систем

Серед всіх сервісів пошуку літератури можна виділити три основних це: Google Scholar, Scopus та PubMed. Google Scholar, Scopus та PubMed – це найбільш відомі та популярні бази даних наукової літератури, які забезпечують доступ до статей та джерел з різних галузей науки. Нижче наведені детальніші відомості про кожен з них:

Google Scholar є безкоштовним пошуковим сервісом, який дозволяє знайти наукові статті, тези, книги та інші джерела з різних галузей науки. Google Scholar використовує алгоритм, який шукає інформацію на університетських веб-сторінках, сайтах наукових журналів та інших ресурсах. Google Scholar надає можливість пошуку за авторами, ключовими словами, та іншими параметрами. Також, він надає можливість створення власного профілю, де можна зберігати статті та додавати відгуки про них.

Scopus є базою даних наукової літератури, яка містить більше 76 мільйонів записів зі книгами, статтями, конференціями та іншими джерелами з різних галузей науки. Scopus надає можливість пошуку за авторами, ключовими словами, інституціями та іншими параметрами. Більше того, Scopus забезпечує можливість оцінювання наукових журналів за імпакт-фактором, що дозволяє визначити вплив журналу на наукову громадськість. Scopus також забезпечує можливість аналізу цитувань, що дозволяє встановити, які наукові статті є найбільш цитованими в певній галузі науки.

PubMed є базою даних медичної літератури, яка містить більше 30 мільйонів записів зі книгами, статтями, конференціями та іншими джерелами з

медичної науки. PubMed забезпечує можливість пошуку за авторами, ключовими словами, назвою журналу та іншими параметрами. Крім того, PubMed надає можливість фільтрувати результати пошуку за типом джерела (стаття, книга, клінічний випадок тощо), датою публікації та іншими параметрами. PubMed також містить більше 5 мільйонів записів про медичні препарати, що дозволяє лікарям та науковцям знайти інформацію про різні ліки та їх взаємодію з організмом.

Висновок до розділу.

Отже було розглянуто важливість літератури в сучасному світі також було висвітлено проблему пошуку літератури. Було проаналізовано тенденцію розвитку веб-сервісів в Україні. Було розглянуто і проаналізовано технології розробки веб-сервісів та архітектурні рішення. Було розглянуто особливості популярних систем пошуку літератури.

РОЗДІЛ 2

АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ, ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-СЕРВІСУ

2.1 Аналіз сценаріїв використання

Сценарії використання – це опис взаємодії користувача з веб-сервісом та опис його функціональних можливостей. Далі наведено декілька сценаріїв використання веб-сервісу автоматизації пошуку літератури:

- Пошук за автором: Користувач вводить ім’я автора, який написав книгу. Веб-сервіс відображає результати пошуку, які включають список книг, опублікованих цим автором, та іншу інформацію про них.
- Пошук за назвою: Користувач вводить назву або частину назви книги, яку інформацію про яку користувач хоче отримати. Веб-сервіс відображає список книг по назві або які включають введену частину назви книги.
- Фільтрація за ціною: Користувач вводить максимальну і/або мінімальну ціну книги. Веб-сервіс відображає список всіх книг які попали в цю цінову категорію.
- Фільтрація за жанром: Користувач обирає один або декілька з представлених жанрів, по яким він хоче відфільтрувати всі книги. Веб-сервіс виводить список всіх книг які включають виділені жанри.

Ці сценарії використання демонструють різні можливості веб-сервісу автоматизації пошуку літератури. Кожен з них дозволяє користувачу здійснювати пошук та отримувати інформацію про книги за великою кількістю параметрів, які відповідають його потребам. Використання такого веб-сервісу може значно спростити пошук необхідної літератури для наукових досліджень, допомогти в роботі з науковими джерелами, збільшити продуктивність роботи дослідника та зекономити гроші на закупівлі літератури.

2.2 Архітектурні рішення веб сервісів

Архітектура програмного забезпечення відноситься до базової структури будь-якої програмної системи та включає будь-який аспект, завдяки якому система функціонує та поводиться належним чином. Хоча термін архітектура зазвичай відноситься до фізичних проектів, у програмних системах він охоплює дизайн компонентів, зв’язки між компонентами, взаємодію з користувачем, а також потреби користувача в системі. Декілька архітектурних підходів, які використовуються в сучасних веб-додатках, включають:

- RESTful архітектура: Цей підхід заснований на принципах архітектури REST (Representational State Transfer) і використовує HTTP-протокол для передачі даних між клієнтом та сервером. Він сприяє взаємодії між клієнтом та сервером, що дозволяє взаємодіяти з ресурсами через їх URI.
- Мікросервісна архітектура: Цей підхід полягає в тому, щоб розбити веб-додаток на набір невеликих незалежних сервісів, які можуть взаємодіяти між собою через API. Цей підхід дозволяє швидко масштабувати окремі частини додатку, що може бути корисним у випадку високої навантаженості на певні сервіси.
- Архітектура з панелями: Цей підхід використовується для створення веб-додатків, що складаються з незалежних панелей, які можуть взаємодіяти між собою. Кожен панель може містити свою логіку та може бути оновлений окремо від інших панелей.
- Архітектура зі шаруванням: Цей підхід полягає в тому, щоб розбити веб-додаток на набір шарів, які взаємодіють між собою. Кожен шар містить свою логіку та може бути змінений без впливу на інші шари.

У Таблиці 2.1 наведено переваги і недоліки використання перелічених типів архітектур веб-сервісів.

Таблиця 2.1

Архітектура	**Плюси**	**Мінуси**
RESTful	Простота взаємодії між клієнтом та сервером, повторне використання коду,	Обмеження використання HTTP-протоколу, складність в обробці запитів з

	швидкість розробки	багатьох різних джерел
Мікросервісна	Швидке масштабування окремих сервісів, незалежність компонентів	Складність взаємодії між сервісами, більш важке управління та координація сервісів
З панелями	Простота оновлення окремих компонентів, швидкість розробки	Обмежена можливість масштабування та взаємодії між компонентами
З шаруванням	Чітка організація логіки додатку, можливість розширення та зміни окремих шарів	Обмежена можливість масштабування та складність управління багатьма шарами

2.2.1 RESTful архітектура

REST розшифровується як Representational State Transfer. Це був термін, спочатку введений Роєм Філдінгом, який також був одним із творців протоколу HTTP. Відмінною особливістю сервісів REST є те, що вони дозволяють якнайкраще використовувати протокол HTTP.

Коли вводиться URL-адреса, наприклад www.google.com, на сервер надсилається запит на веб-сайт, ідентифікований цією URL-адресою. Потім цей сервер формує та видає відповідь. Важливим є формат цих запитів та відповідей. Ці формати визначаються протоколом HTTP – Hyper Text Transfer Protocol.

Коли ви набираєте URL-адресу в браузері, він надсилає запит GET на вказаний сервер. Потім сервер відповідає HTTP-відповіддю, що містить дані у форматі HTML – Hyper Text Markup Language. Потім браузер отримує цей HTML-код та відображає його на екрані.

Є чотири основних методів HTTP-запиту це:

- GET – отримати інформацію про ресурс
- POST – створити новий ресурс
- PUT – оновити існуючий ресурс
- DELETE – видалити ресурс

Ресурс — це ключова абстракція, де концентрується протокол HTTP. Ресурс – це все, що ви хочете показати зовнішньому світу через вашу програму. Наприклад, якщо ми пишемо додаток для управління завданнями, екземпляри ресурсів будуть такі:

- Конкретний користувач
- Конкретне завдання
- Список завдань

В REST архітектурі найважливіше правильно обрати ресурси які будуть показуватись користувачу. Ось як завжди реалізується служба REST:

- Формат обміну даними: тут немає жодних обмежень. JSON — дуже популярний формат, хоча можна використовувати інші, такі як XML
- Транспорт: завжди HTTP. REST повністю побудований на основі HTTP.
- Визначення сервісу: немає стандарту при цьому, а REST є гнучким. Це може бути недоліком у деяких сценаріях, оскільки споживає додаток може бути необхідно розуміти формати запитів і відповідей. Однак

широко використовуються такі мови визначення веб-застосунків, як WADL (Web Application Definition Language) і Swagger.

REST фокусується на ресурсах і тому, наскільки ефективно ви виконуєте операції з ними, використовуючи HTTP.

Отже HTTP є основним будівельним блоком REST сервісів. HTTP – це протокол, який використовується для визначення структури запитів та відповідей браузера. HTTP має справу переважно з ресурсами, доступними на веб-серверах.

2.2.2 Мікросервісна архітектура

Мікросервіс – це найпростіша одиниця, сервіс, який приймає вхідні запити для здійснення дії. Це може бути бекенд сервіс, який доступний цілодобово та без вихідних, або функція або набір функцій, доступних через певний API через мережу. Отже, це бекенд служба, розгорнута на сервері.

На Рисунку 2.1 зображено найпоширеніші яруси що використовуються в мікросервісній архітектурі. Назви можуть відрізнятись, але структура дуже схожа на класичну багатошарову архітектуру.

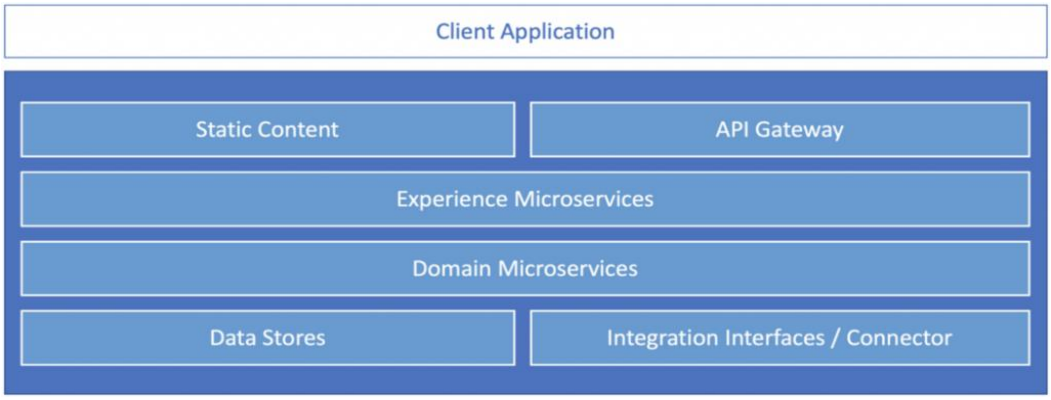

Рисунок 2.1 – Архітектура мікросервісного веб-додатку

Далі наведено опис кожного ярусу:

- Client Application: У випадку веб-програми, клієнтом, як правило, є веб-браузер який завантажує статичний контент із віддаленого сервера і рендерить UI всередині себе. І він також обробляє фізичні HTTP запити ща надходять до сервера.
- Static Content: Найкраще розділити відповідальність шляхом відокремлення логічних компонентів, щоб жоден компонент не відповідав за все. З цієї причини рекомендується обслуговувати статичний вміст через окремий серверний компонент. Це дозволяє відокремити статичний вміст від API. Кілька важливих принципів, які слід враховувати, це рендеринг на стороні сервера, кешування та контроль доступу.
- API Gateway: Це центральні двері до всіх публічних API. Це єдина точка входу для так званих Experience API, доступних для клієнтських програм, і є важливим компонентом найкращих практик керування API. По суті, він передає запити реальним серверним службам і може приймати рішення. Найчастіше API Gateway відповідає за такий функціонал як:
    - Request handling
    - Upstream (або downstream) data transfer (або transformation)
    - Access control
    - Security policies
    - Throttling
    - Rate limiting
    - Analytics
    - Caching
- Experience microservices: У неструктурованому розподіленому середовищі дуже легко втратити контроль над безліччю мікросервісів. З цієї причини рекомендується контролювати мікросервіси, структуруючи їх на рівні. Існують різні архітектурні моделі, які можуть допомогти. Спільним для цих шаблонів є те, що всі вони зосереджені на experience.

- Domain microservices: Мікросервіси домену — це нижній рівень, який відповідає за бізнес-логіку і від якого залежать experience microservices. Таким чином, experience microservices зосереджені на користувачеві та продукті, тоді як domain microservices відповідають за дані та ресурси під їх контролем. Таким чином, мікросервіси домену надають детальні API нижчого рівня, які дозволяють створювати різні experience microservices, повторно використовуючи доступну логіку домену. У результаті ми маємо ситуацію, коли додавання кожної нової служби experience не вимагає додаткового часу, витраченого на повторну розробку логіки домену.
- Data stores: Найкраще, щоб база даних належала лише одному мікросервісу. Жодні інші служби не повинні мати прямий доступ до бази даних, лише через API служби-власника. Це дозволяє кожній службі керувати послідовністю та структурою бази даних, якою вона володіє. Це також дозволяє вибрати найбільш підходящу базу даних для конкретних цілей без будь-якої залежності від вимог системи в цілому. Наприклад, одна служба може використовувати нормалізовану базу даних SQL, тоді як інша може використовувати базу даних NoSQL. Однак один мікросервіс може керувати кількома базами даних.
- Integration Interfaces/Connector: Окрім внутрішніх джерел даних, програмі може знадобитися доступ до інших підсистем і сторонніх API. Корисною практикою є відокремлення інтеграційного рівня шляхом введення окремих служб-конекторів.

Архітектура мікросервісу — це, по суті, шаблон, який навчає, як застосовувати різні архітектурні шаблони, стилі та підходи в одній розподіленій програмі.

2.2.3 Архітектура з панелями

У цьому типі архітектури веб-служба складається з серії панелей, кожна з яких міститиме певний набір інформації або функціональних можливостей. Ці панелі можуть містити такі речі, як панель входу, панель пошуку, панель результатів і панель налаштувань.

Веб-служба розробляється таким чином, щоб дозволити користувачам взаємодіяти з цими панелями різними способами, залежно від їхніх потреб і вподобань. Наприклад, користувачі можуть перетягувати панелі, щоб змінювати їх розташування на екрані, або налаштовувати інформацію, що відображається на кожній панелі. На рисунку 2.2 зображено приклад панельної архітектури веб-сервісу.

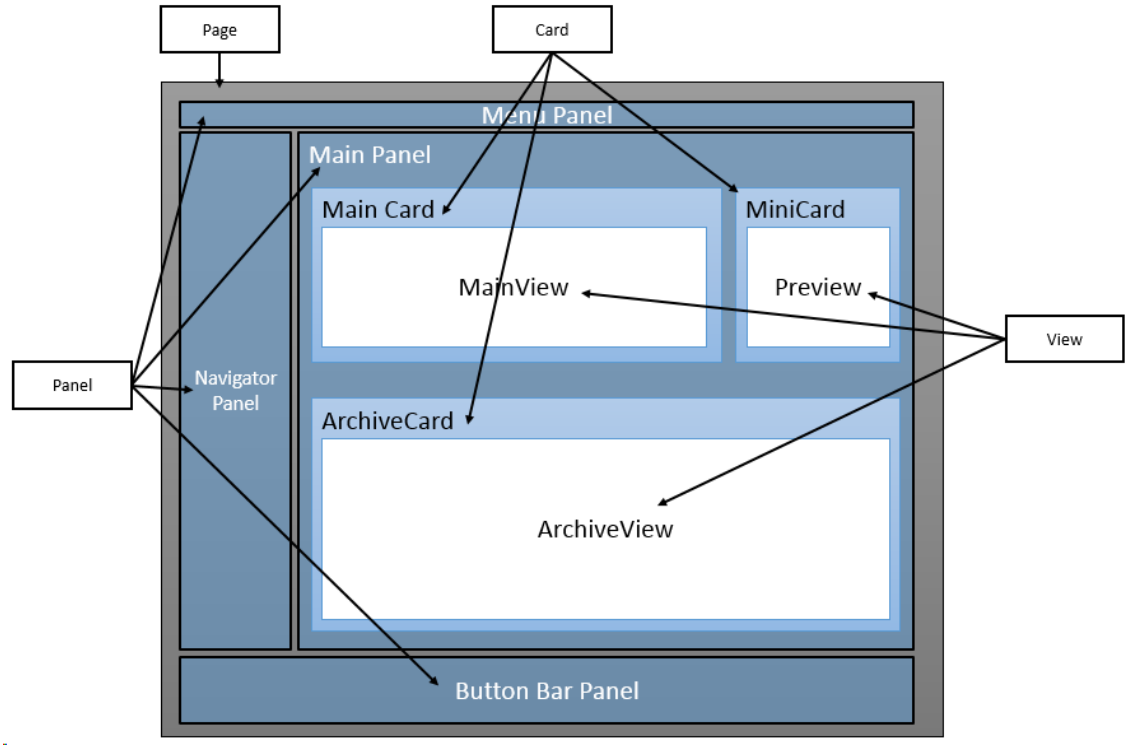

Рисунок 2.2 – Панельна архітектура

Щоб реалізувати цей тип архітектури, веб-розробники можуть використовувати різноманітні технології, такі як HTML, CSS, JavaScript і мову програмування на стороні сервера, наприклад Python або Ruby. Вони також

можуть використовувати такі веб-фреймворки, як React або Angular, щоб допомогти керувати інтерфейсом користувача та взаємодіями користувачів.

Загалом, використання панелей в архітектурі веб-сервісу може допомогти покращити зручність використання та гнучкість сервісу, оскільки користувачі можуть легко отримати доступ до потрібної їм інформації та налаштувати інтерфейс відповідно до своїх потреб. Однак розробка та реалізація такої архітектури може бути складною та вимагає пильної уваги до потреб і поведінки користувачів.

2.2.4 Архітектура з шаруванням

Вважається, що багаторівневі архітектури є найпоширенішою і широко використовуваною архітектурною структурою в розробці програмного забезпечення. Вона також відома як n-рівнева архітектура та описує архітектурний шаблон, що складається з кількох окремих горизонтальних рівнів, які функціонують разом як єдиний блок програмного забезпечення. Рівень — це логічне розділення компонентів або коду. У цих структурах компоненти, які пов’язані або схожі, зазвичай розміщуються на тих самих шарах. Однак кожен рівень відрізняється і вносить свій внесок у іншу частину загальної системи.

Основною характеристикою цього фреймворку є те, що шари підключаються лише до шарів безпосередньо під ними. Іншою характеристикою є концепція шарів ізоляції. Це означає, що шари можна змінювати, і ця зміна не вплине на інші шари. Коротше кажучи, зміни відносяться до конкретного шару, який змінено. Поділ завдань — це ще одна помітна особливість, яка говорить про те, як модулі на одному рівні разом виконують одну функцію.

Кількість рівнів у багатошаровій архітектурі не встановлюється на конкретне число і зазвичай залежить від розробника чи архітектора

програмного забезпечення. Важливо зазначити, що ця структура зазвичай завжди має рівень взаємодії з користувачем, рівень обробки та рівень обробки даних. Вони описані далі як:

- Презентаційний рівень – відповідає за взаємодію користувача з програмною системою
- Прикладний/бізнес-рівень – обробляє аспекти, пов’язані з виконанням функціональних вимог
- Domain Layer – відповідає за алгоритми та компоненти програмування
- Рівень інфраструктури/постійності/бази даних – відповідає за обробку даних, баз даних

На рисунку 2.3 зображено приклад архітектури з шаруванням з перерахованими вище рівнями.

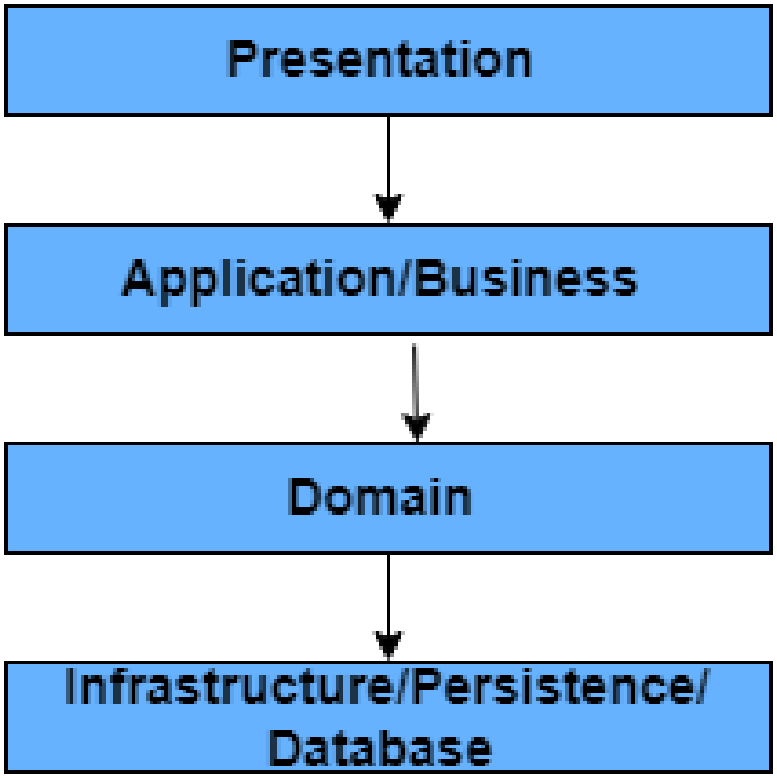

Рисунок 2.3 – Архітектура з шаруванням

Крім того, у деяких програмах деякі шари поєднуються. Наприклад, зазвичай бізнес-рівень і рівень стійкості об’єднані в один рівень. Це лише означає, що функції та обов’язки цих двох рівнів згруповано для виконання на одному рівні.

Нижче наведено переваги та недоліки цього шаблону програмного забезпечення.

Переваги:

- Фреймворк простий і легкий для вивчення та впровадження.
- Залежність зменшується, оскільки функції кожного шару відокремлені від інших.
- Тестування легше завдяки роздільним компонентам, кожен компонент можна тестувати окремо.
- Накладні витрати досить низькі.

Недоліки

- Масштабування складне, оскільки структура фреймворку не дозволяє рости.
- Їх важко підтримувати. Зміна в одному рівні може вплинути на всю систему, оскільки вона працює як єдине ціле.
- Існує взаємозалежність між шарами, оскільки рівень залежить від рівня над ним для отримання даних.
- Паралельна обробка неможлива.

При розробці простих невеликих додатків доцільно реалізувати багаторівневу архітектуру, оскільки це найпростіша структура. Однак деякі розробники вважають, що оскільки їх важко підтримувати, краще застосовувати їх до більших проектів. Незважаючи на це, фреймворк можна

використовувати для додатків, які потрібно створювати швидко, оскільки його легко вивчити та реалізувати. Це також добре у випадках, коли розробники не мають достатньо знань про архітектуру програмного забезпечення або коли вони не визначилися, яку з них використовувати.

2.3 Бази даних

База даних — це набір даних, організованих у структурований спосіб, щоб забезпечити ефективне зберігання, пошук і керування інформацією. Бази даних використовуються в широкому діапазоні додатків, від простого ведення записів до складних ділових і наукових додатків. Бази даних відіграють вирішальну роль у веб-розробці, оскільки вони використовуються для зберігання та керування даними для веб-додатків. Деякі способи використання баз даних у веб-розробці:

- Автентифікація користувача: багато веб-додатків вимагають автентифікації користувача, яка передбачає збереження інформації користувача, такої як імена користувачів, паролі та особиста інформація, у базі даних. База даних використовується для перевірки облікових даних користувача, коли вони намагаються увійти, а також для збереження налаштувань користувача та інших даних.
- Управління вмістом: веб-сайти часто мають динамічний вміст, який потрібно часто оновлювати. Бази даних використовуються для зберігання цього вмісту та дозволяють користувачам створювати, оновлювати та видаляти його через систему керування вмістом (CMS).
- Електронна комерція: Інтернет-магазинам потрібні бази даних для зберігання інформації про продукт, клієнта та історії замовлень. Ці дані

використовуються для обробки замовлень, керування запасами та створення звітів.

- Аналітика: веб-програми генерують багато даних, таких як поведінка користувачів, перегляди сторінок і показники кліків. Бази даних використовуються для зберігання цих даних і створення звітів і статистичних даних, які допомагають власникам веб-сайтів оптимізувати свої сайти.
- Обмін повідомленнями: програми обміну повідомленнями та чату використовують бази даних для зберігання профілів користувачів, повідомлень і розмов. Ці дані використовуються для доставки повідомлень правильному одержувачу та надання користувачам доступу до історії повідомлень.
- Соціальні медіа: Платформи соціальних мереж значною мірою покладаються на бази даних для зберігання профілів користувачів, публікацій і взаємодії. Ці дані використовуються для створення персоналізованих каналів і рекомендацій, а також для показу цільової реклами.

Для досягнення цих цілей існує кілька типів баз даних, зокрема:

1. Реляційні бази даних: це найпоширеніший тип бази даних, який базується на реляційній моделі даних. Дані в реляційній базі даних організовані в таблиці, кожна з яких містить рядки та стовпці. Зв’язки між таблицями визначаються ключами, які використовуються для об’єднання даних між таблицями. Прикладами реляційних баз даних є MySQL, Oracle і Microsoft SQL Server.
2. Бази даних NoSQL: це нереляційні бази даних, призначені для обробки великих обсягів неструктурованих даних. Бази даних NoSQL часто використовуються в програмах, які вимагають обробки даних у

реальному часі, таких як соціальні мережі та електронна комерція. Приклади баз даних NoSQL включають MongoDB і Cassandra.

3. Об’єктно-орієнтовані бази даних: ці бази даних зберігають дані у формі об’єктів, якими можна маніпулювати за допомогою методів об’єктно-орієнтованого програмування. Об’єктно-орієнтовані бази даних часто використовуються в програмах, які вимагають складних зв’язків даних, таких як наукове та інженерне моделювання.
4. Графічні бази даних: ці бази даних зберігають дані у формі вузлів і ребер, які використовуються для представлення складних зв’язків між даними. Графові бази даних часто використовуються в програмах, які вимагають обробки даних у реальному часі, таких як соціальні мережі та механізми рекомендацій. Прикладами графових баз даних є Neo4j і OrientDB.

Окрім цих типів баз даних, існують також бази даних у пам’яті, бази даних часових рядів та інші спеціалізовані бази даних, які призначені для обробки певних типів даних або програм. Далі наведено декілька типів найпопулярніших баз даних:

1. MySQL: MySQL — це система керування реляційною базою даних (RDBMS) із відкритим кодом, яка широко використовується у веб-додатках. Він відомий своєю простотою використання, масштабованістю та гнучкістю.
2. PostgreSQL: PostgreSQL — це об’єктно-реляційна система керування базами даних (ORDBMS) із відкритим кодом, відома своєю надійністю, масштабованістю та розширеними функціями. Він зазвичай використовується в програмах корпоративного рівня.
3. Oracle: Oracle — це комерційна реляційна система керування базами даних, яка широко використовується у великих корпоративних програмах. Він відомий своєю високою доступністю, масштабованістю та функціями безпеки.

4. SQL Server: Microsoft SQL Server — це реляційна система керування базами даних, яка широко використовується в програмах на базі Windows. Він відомий своєю інтеграцією з іншими продуктами Microsoft і простотою використання.
5. MongoDB: MongoDB — це популярна документоорієнтована база даних NoSQL, призначена для обробки великих обсягів неструктурованих даних. Він зазвичай використовується у веб-додатках, які потребують обробки даних у реальному часі.
6. Firebase: Firebase — це хмарна база даних NoSQL, яка зазвичай використовується в мобільних і веб-додатках. Він відомий своєю простотою використання, синхронізацією даних у реальному часі та інтеграцією з іншими службами Firebase.
7. Cassandra: Apache Cassandra — це розподілена база даних NoSQL, призначена для обробки великих обсягів даних на кількох серверах. Він відомий своєю масштабованістю, високою доступністю та відмовостійкістю, і зазвичай використовується у великомасштабних програмах.
8. Redis: Redis — це сховище структур даних у пам’яті з відкритим кодом, яке можна використовувати як базу даних, кеш-пам’ять і брокер повідомлень. Він відомий своєю швидкістю, масштабованістю та підтримкою розширених типів даних.
9. SQLite: SQLite — це легка вбудована система керування реляційною базою даних, яка широко використовується в мобільних і веб-додатках. Він відомий своєю простотою, швидкістю та малим обсягом пам’яті.
10. Amazon Aurora: Amazon Aurora — це хмарна система керування реляційною базою даних, яка розроблена для високої масштабованості та доступності. Він відомий своєю високою продуктивністю, низькою затримкою та сумісністю з MySQL і PostgreSQL.

Це лише кілька прикладів із багатьох доступних сьогодні баз даних, кожна з яких має свої сильні та слабкі сторони. Вибір бази даних залежить від конкретних потреб програми, включаючи обсяг даних, складність, масштабованість і вимоги до продуктивності.

В таблиці 2.2 наведено порівняння баз даних за такими критеріями як: тип, масштабованість, оновлення в реальному часі, безпека та особливості.

Таблиця 2.2

База даних	**Тип**	**Можливість розширення**	**Оновлення в реальному часі**	**Захист**	**Особливості**
MySQL	Реляційна	Висока	-	Високий	Відповідність ACID, реплікація, збережені процедури
PostgreSQL	Об’єктно-реляційна	Висока	-	Високий	Відповідність ACID, реплікація, збережені процедури, додаткові функції
Oracle	Реляційна	Висока	-	Дуже високий	Відповідність ACID, реплікація, розділення, розширена безпека

SQL Server	Реляційна	Висока	-	Високий	Відповідність ACID, реплікація, збережені процедури, інтеграція з продуктами Microsoft
MongoDB	Не реляційна	Висока	+	Високий	Гнучка модель документа, висока доступність, шардинг
Firebase	Не реляційна	Середня	+	Дуже високий	Автентифікація, хостинг

Продовження таблиці 2.2

Cassandra	Не реляційна	Висока	+	Високий	Розподіленість, висока доступність, відмовостійкий
Redis	Сховище даних у пам'яті	Висока	+	Високий	Обмін повідомленнями Pub/Sub, кешування, транзакції
SQLite	Реляційна	Обмежена	-	Високий	Легкість, вбудовані, транзакції, відповідність ACID
Amazon Aurora	Реляційна	Висока	+	Дуже високий	Відповідність ACID, автоматичне масштабування,

					реплікація, резервне копіювання та відновлення

Варто зауважити що всі ці бази даних постійно розвиваються і змінюються. Вони оновлюються разом з іншими технологіями сучасності.

Висновок до розділу:

Отже було проведено аналіз сценаріїв використання, розглянуто види архітектур, їх особливості використання, переваги та недоліки. Також було більш детально розглянуто кожну архітектуру окремо. Було здійснено аналіз найсучасніших баз даних і розроблено результуючу таблицю їх порівняння.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ВЕБ СЕРВІСУ АВТОМАТИЗАЦІЇ ПОШУКУ ЛІТЕРАТУРИ

3.1 Вибір бази даних

Провівши аналіз найпопулярніших баз даних я зупинив свій погляд на Firebase. Firebase — це платформа Backend-as-a-Service (BaaS), яка надає розробникам різноманітні інструменти та служби для створення мобільних і веб-додатків. Він належить Google і пропонує низку функцій, зокрема:

1. База даних у реальному часі: база даних NoSQL, яка забезпечує синхронізацію в реальному часі й автоматичну синхронізацію даних на всіх клієнтах і пристроях.
2. Автентифікація: дозволяє користувачам реєструватися та входити за допомогою електронної пошти та пароля, облікових записів у соціальних мережах та іншими методами.

3. Хмарні функції: безсерверні обчислення, які дозволяють розробникам запускати серверний код у відповідь на події, ініційовані Firebase або сторонніми службами.
4. Хмарне сховище: масштабоване та безпечне хмарне рішення для зберігання файлів і медіа.
5. Хостинг: веб-хостинг для статичного та динамічного вмісту з автоматичним шифруванням SSL.
6. Аналітика: надає інформацію про поведінку користувачів, продуктивність програми та інші показники.
7. Remote Config: дозволяє розробникам налаштовувати поведінку та зовнішній вигляд програми без необхідності випускати нову версію програми.
8. Test Lab: хмарна тестова платформа, яка дозволяє розробникам тестувати свої програми на реальних пристроях і середовищах.

Firebase відомий своєю простотою використання та інтеграцією з іншими службами Google. Він пропонує широкий спектр функцій, які можуть допомогти розробникам створювати високоякісні та масштабовані мобільні та веб-додатки, не турбуючись про керування інфраструктурою. Однак Firebase не є повноцінним серверним рішенням і може не підходити для всіх типів програм, особливо тих, які потребують більшого контролю над серверною частиною.

Firebase пропонує два варіанти основних баз даних: Firebase Realtime Database і Cloud Firestore. Хоча обидві бази даних розроблено для забезпечення синхронізації даних у реальному часі та підтримки мобільних і веб-додатків, вони відрізняються кількома ключовими параметрами:

1. Модель даних: Firebase Realtime Database зберігає дані як дерево JSON, тоді як Cloud Firestore використовує колекції та документи для організації даних. Це робить Cloud Firestore більш гнучким і краще підходить для складних структур даних.

2. Запити: Cloud Firestore підтримує більш складні та гнучкі запити, ніж Firebase Realtime Database. Cloud Firestore підтримує запити, які можуть масштабуватися до мільйонів документів і надавати оновлення в реальному часі, тоді як запити до бази даних у реальному часі мають більш обмежені можливості.
3. Масштабованість: Cloud Firestore розроблено для горизонтального масштабування, що означає, що він може виконувати більший обсяг операцій читання та запису, ніж Realtime Database. База даних реального часу може стати менш чуйною, якщо їй потрібно обробляти велику кількість з'єднань.
4. Ціноутворення: Cloud Firestore пропонує більш гнучку модель ціноутворення, ніж Realtime Database. У Cloud Firestore розробники платять лише за обсяг збережених даних, тоді як у Realtime Database розробники стягують плату за обсяг переданих даних.
5. Безпека: обидві бази даних пропонують такі функції безпеки, як автентифікація, контроль доступу та перевірка даних. Однак Cloud Firestore пропонує більш детальний контроль доступу, ніж Realtime Database, дозволяючи розробникам обмежувати доступ до окремих документів або полів.

Загалом Cloud Firestore є більш розширеною та гнучкою базою даних, ніж Realtime Database, і краще підходить для програм, які потребують складних запитів і масштабованості. Проте Realtime Database все ще є потужним варіантом для програм, які потребують синхронізації даних у реальному часі та простішої моделі даних.

Проаналізувавши послуги Firebase я обрав для реалізації веб-сервісу Realtime database. Firebase Realtime Database — це хмарна база даних NoSQL, яка зберігає та синхронізує дані в режимі реального часу на всіх підключених клієнтах і пристроях. Це ключова функція Firebase, платформи Backend-as-a-Service (BaaS), що належить Google, і широко використовується розробниками

для створення програм реального часу, таких як програми чату, онлайн-ігри, інструменти для спільної роботи тощо.

Firebase Realtime Database зберігає дані як об’єкти JSON, що забезпечує гнучке моделювання даних і легку інтеграцію з веб-програмами та мобільними додатками. Він підтримує автоматичну синхронізацію, що означає, що будь-які зміни, внесені в базу даних, миттєво поширюються на всі підключені пристрої без необхідності оновлення сервера. Це полегшує створення додатків у реальному часі, які вимагають від декількох користувачів перегляду та взаємодії з тими самими даними в режимі реального часу.

Firebase Realtime Database також підтримує офлайн-доступ до даних, що означає, що користувачі можуть отримувати доступ до даних і змінювати їх, навіть якщо вони не підключені до Інтернету. База даних автоматично синхронізує зміни з хмарою, щойно підключення до мережі стає доступним.

Однією з ключових переваг Firebase Realtime Database є простота її використання. Розробники можуть розпочати роботу з базою даних за лічені хвилини, не встановлюючи сервер чи керуючи інфраструктурою. Вони можуть просто інтегрувати Firebase у свою програму, визначити структуру даних і правила доступу та почати читати та записувати дані. Firebase Realtime Database також надає потужні можливості запиту даних, дозволяючи розробникам отримувати та фільтрувати дані на основі різних критеріїв.

Firebase Realtime Database також має високу масштабованість і може обробляти мільйони одночасних підключень і складних моделей даних. Він пропонує розраховану модель ціноутворення, що означає, що розробники платять лише за те, що вони використовують, що робить його економічно ефективним варіантом як для малих, так і для великих програм.

Загалом Firebase Realtime Database — це потужне та гнучке рішення для баз даних, яке пропонує синхронізацію в реальному часі, офлайн-доступ до даних і легку інтеграцію з веб- та мобільними програмами. Це ідеальний вибір

для розробників, які хочуть створювати програми в реальному часі швидко та легко, не турбуючись про керування інфраструктурою.

3.2 Вибір фреймворка

Vue.js і React є популярними фреймворками JavaScript, які використовуються для створення веб-додатків. Хоча обидва фреймворки мають певну схожість, вони також мають низку ключових відмінностей:

1. Крива навчання: Vue.js має простіший та інтуїтивніший API, ніж React, що полегшує розробникам навчання та використання. React, з іншого боку, має крутішу криву навчання та вимагає більше знань про JavaScript та екосистему.
2. Шаблони: Vue.js використовує синтаксис на основі шаблонів для створення компонентів інтерфейсу користувача, що полегшує читання та запис. React використовує JSX, який вимагає від розробників змішувати HTML і JavaScript у своєму коді.
3. Управління станом: Vue.js має вбудоване рішення для керування станом під назвою Vuex, яке спрощує керування та обмінюватися станом між компонентами. React використовує сторонні бібліотеки, такі як Redux, для керування станом.
4. Директиви: Vue.js має потужний набір вбудованих директив, таких як v-model і v-if, які спрощують маніпулювання DOM і обробку взаємодії користувача. React використовує обробники подій і методи життєвого циклу для маніпулювання DOM.
5. Продуктивність: Vue.js має менший розмір файлу та вищу продуктивність відтворення, ніж React. Це пояснюється тим, що Vue.js використовує систему візуалізації на основі шаблонів, яка дозволяє оптимізувати та ефективніше кешувати компоненти.

6. Екосистема: React має більшу та зрілішу екосистему, ніж Vue.js, із широким спектром сторонніх бібліотек та інструментів, доступних для створення складних програм. Vue.js, незважаючи на швидкий розвиток, має меншу спільноту та менше сторонніх опцій.

Загалом Vue.js і React є потужними та гнучкими фреймворками, які пропонують широкий спектр можливостей для створення веб-додатків. Вибір між двома зрештою зводиться до потреб проекту, досвіду та вподобань команди розробників, а також загальних цілей і завдань програми.

Отже для реалізації веб-сервісу я обрав фреймворк React. React — популярна платформа JavaScript з відкритим вихідним кодом для створення інтерфейсів користувача та веб-додатків. React, розроблений Facebook, широко використовується розробниками та компаніями по всьому світу завдяки простоті використання, масштабованості та гнучкості.

React використовує компонентну архітектуру, що дозволяє розробникам створювати багаторазові компоненти інтерфейсу користувача, які можна комбінувати та вкладати для створення складних інтерфейсів користувача. Кожен компонент відповідає за рендеринг певної частини інтерфейсу, і його можна легко оновити або замінити за потреби.

Однією з ключових переваг React є його здатність легко обробляти великі та складні програми. React використовує віртуальну DOM (об’єктну модель документа), яка дозволяє ефективно оновлювати та відображати лише компоненти, які змінилися, замість повторного відтворення всієї сторінки.

React також має активну екосистему сторонніх бібліотек та інструментів, які можна використовувати для вдосконалення та розширення його функціональності. Такі популярні бібліотеки, як Redux і React Router, надають рішення для управління станом і маршрутизації відповідно, а такі інструменти, як Next.js і Gatsby.js, пропонують рендеринг на стороні сервера та інші розширені функції.

Крім того, React підтримує широкий спектр платформ і пристроїв, від настільних комп’ютерів до мобільних, і може використовуватися для створення власних мобільних додатків за допомогою React Native.

React має велику та активну спільноту розробників, що дозволяє легко знайти ресурси та підтримку під час створення програм. Документація React також є вичерпною та добре організованою, надає докладні посібники та приклади для початку роботи з фреймворком.

Також для кращого вигляду сайту я використав фреймворк Bootstrap. Bootstrap — це популярна інтерфейсна платформа з відкритим кодом для створення адаптивних веб-додатків, орієнтованих на мобільні пристрої. Розроблений Twitter, Bootstrap надає набір готових компонентів HTML, CSS і JavaScript, які можна легко налаштувати та комбінувати для створення сучасних і візуально привабливих інтерфейсів.

Однією з ключових переваг Bootstrap є простота використання. Завдяки простому та інтуїтивно зрозумілому API розробники можуть швидко створювати адаптивні макети, меню навігації, форми, кнопки та інші компоненти інтерфейсу без необхідності писати розширений код CSS або JavaScript.

Bootstrap також розроблено для мобільних пристроїв, що означає, що він надає пріоритет потребам мобільних користувачів під час створення інтерфейсів. Такий підхід гарантує, що інтерфейс виглядає та добре працює на різноманітних пристроях, від настільних ПК до смартфонів.

Крім того, Bootstrap має велику та активну спільноту розробників, яка сприяла розробці широкого спектру сторонніх плагінів та інструментів. Вони включають додаткові компоненти інтерфейсу користувача, теми та шаблони, які можна використовувати для подальшого налаштування та покращення функціональності Bootstrap.

Bootstrap також має широкі можливості налаштування з широким набором параметрів конфігурації, які дозволяють розробникам змінювати вигляд і поведінку окремих компонентів. Ця гнучкість робить Bootstrap чудовим вибором для проектів будь-якого розміру, від невеликих веб-сайтів до великомасштабних веб-додатків.

3.3 Заповнення таблиць даними

Для заповнення таблиць даними я вирішив використовувати парсери написані мовою python. Парсер – програма, що забезпечує парсинг, тобто синтаксичний аналіз контенту в мережі по певній математичній моделі. Python — це інтерпретована мова програмування високого рівня, яка була вперше випущена в 1991 році Гвідо ван Россумом. Відтоді вона стала однією з найпопулярніших мов програмування у світі, відомою своєю простотою, універсальністю та легкістю використання.

Однією з ключових переваг Python є його читабельність. Код Python розроблено таким чином, щоб його було легко читати та писати, а його синтаксис є простим і зрозумілим. Це робить його популярним вибором для початківців, а також для досвідчених розробників, які хочуть швидко створювати прототипи нових ідей.

Python також відомий своєю універсальністю. Його можна використовувати для широкого спектру додатків, від веб-розробки та аналізу даних до наукових обчислень і машинного навчання. Python має велике та активне співтовариство розробників, яке сприяло розробці широкого спектру бібліотек та інструментів для різних випадків використання.

Ще однією важливою перевагою Python є його підтримка багатьох парадигм програмування, включаючи об’єктно-орієнтоване, функціональне та процедурне програмування. Це робить його гнучкою мовою, яку можна використовувати в багатьох різних контекстах і для різних типів проектів.

Популярність Python також призвела до розвитку великої та різноманітної екосистеми інструментів і фреймворків. До них належать такі веб-фреймворки, як Django та Flask, інструменти аналізу даних, такі як NumPy і Pandas, а також бібліотеки машинного навчання, такі як TensorFlow і PyTorch.

Python також відомий своєю зосередженістю на читабельності коду з набором інструкцій, відомих як PEP 8, які заохочують послідовний і легкий для читання код. Це робить код Python зручнішим для обслуговування та зручнішим для роботи в командному середовищі.

3.4 Інструкція користувача

При переході на сайт відкривається головна сторінка Рисунок 3.1 на якій можна переглянути книги та знайти статті. Книги можна фільтрувати за ціною та жанром. Також присутній пошук книг по автору та назві.

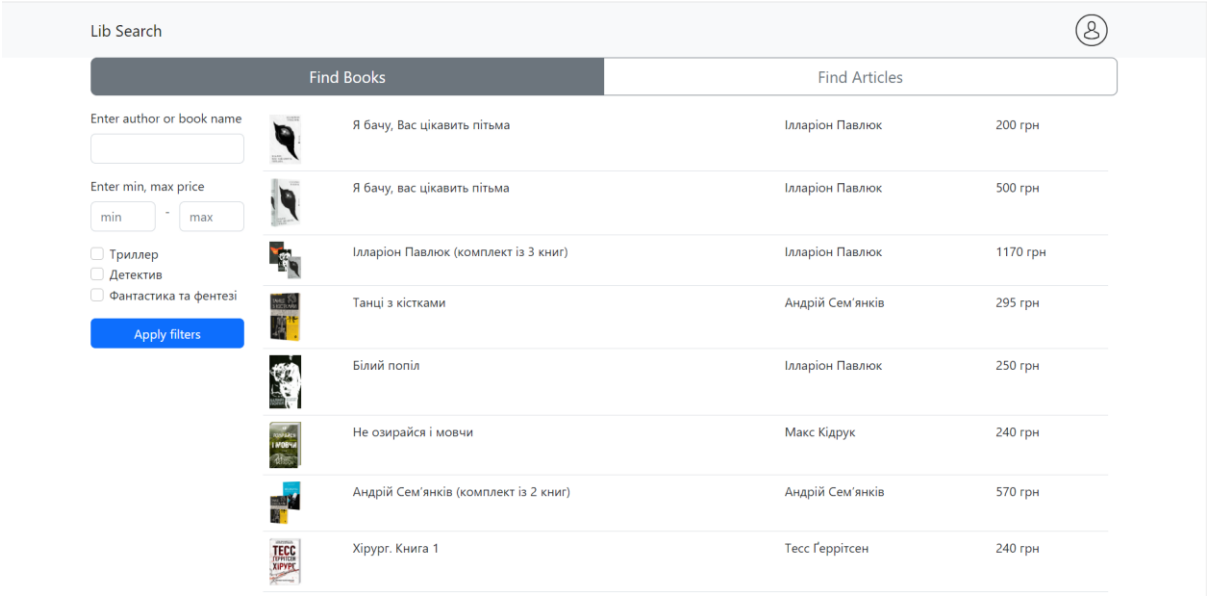

Рисунок 3.1 – Головна сторінка

При натисканні на клітинку можна перейти на сторінку книги в електронному магазині.

Також при натисканні на значок акаунту в верхній правій частині сторінки ми переходимо на сторінку входу Рис. 3.2. Ввійти в акаунт можна за допомогою пошти і паролю або за допомогою акаунту Google.

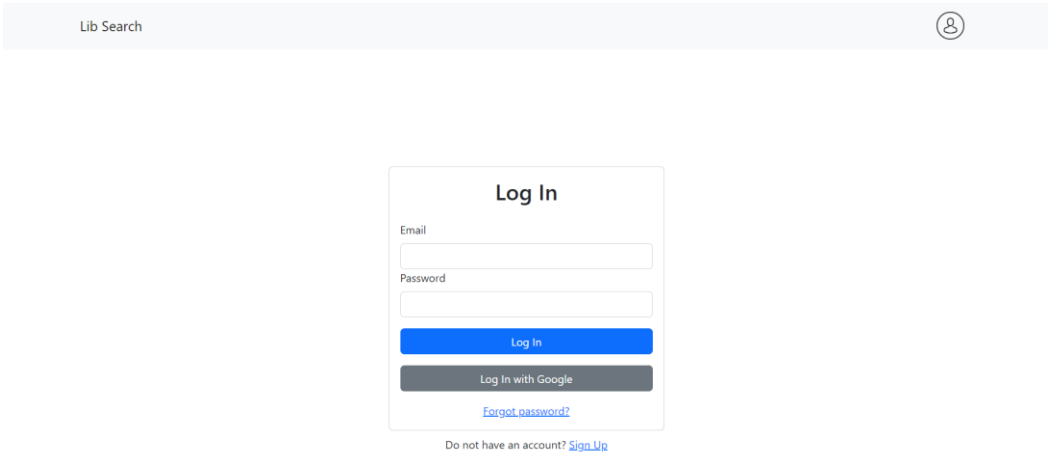

Рисунок 3.2 – Сторінка входу

Також присутня можливість реєстрації якщо відсутній акаунт. Для реєстрації необхідно ввести електронну пошту, пароль і підтвердження паролю.

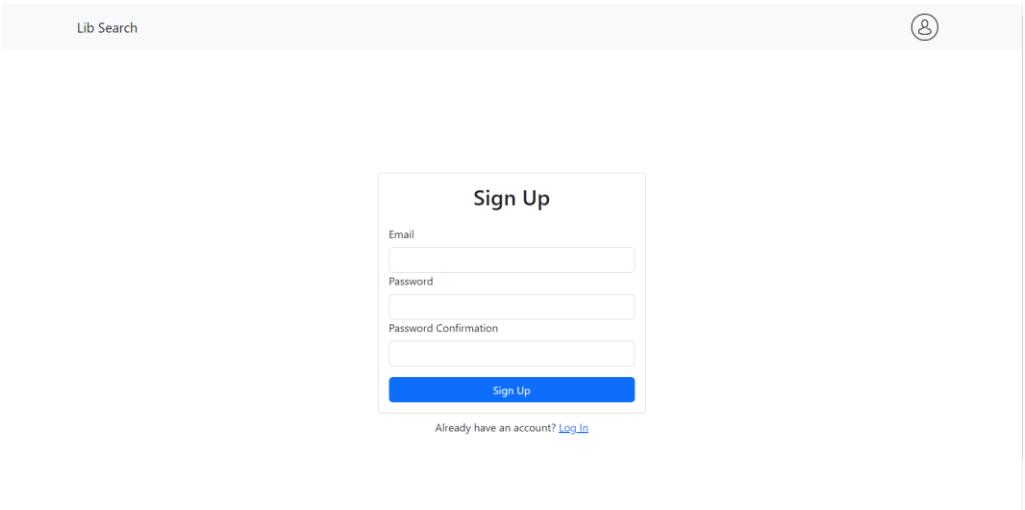

Рисунок 3.3 – Сторінка реєстрації

Також є можливість відновити пароль за допомогою електронної пошти, для цього потрібно на сторінці входу перейти за посиланням «Forgot password?».

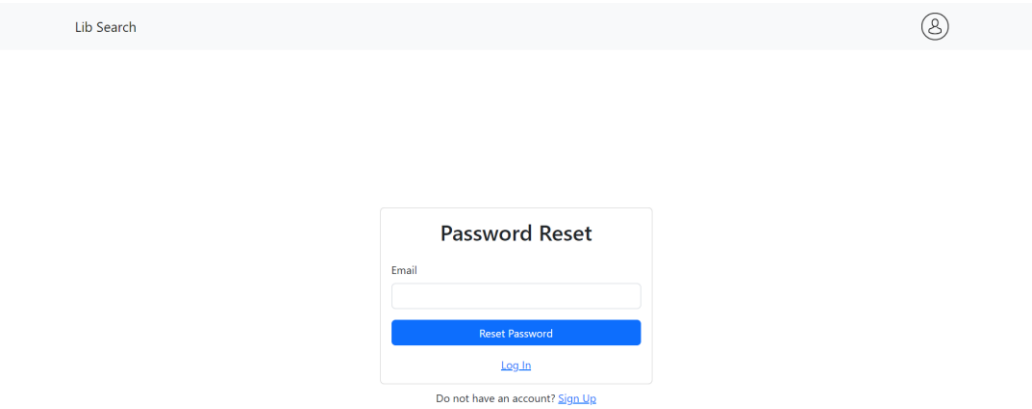

Рисунок 3.4 – Сторінка відновлення паролю

Для того щоб відновити пароль необхідно ввести свою пошту. Далі на пошту прийде повідомлення де потрібно перейти за посиланням і ввести новий пароль. І вже за новим паролем можливо буде здійснити вхід.

Для зареєстрованого користувача головна сторінка виглядає іншим чином. З'являється можливість додавати книги до «Улюблених».

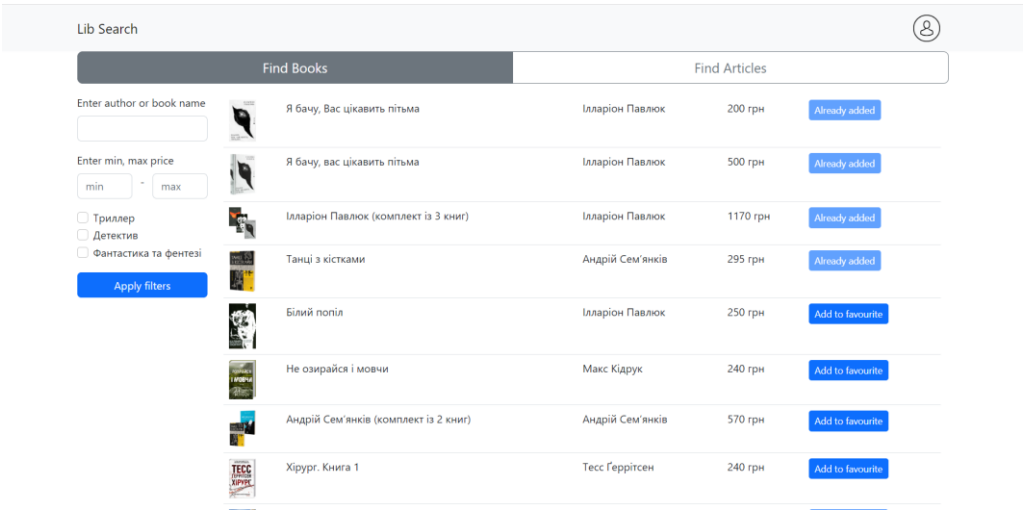

Рисунок 3.5 – Головна сторінка для зареєстрованого користувача

Якщо зареєстрований користувач натисне на іконку акаунту в верхній правій частині сайту він перейде на сторінку акаунту, де він може вийти з акаунту або переглянути книги та статті які були додані до списку улюблених.

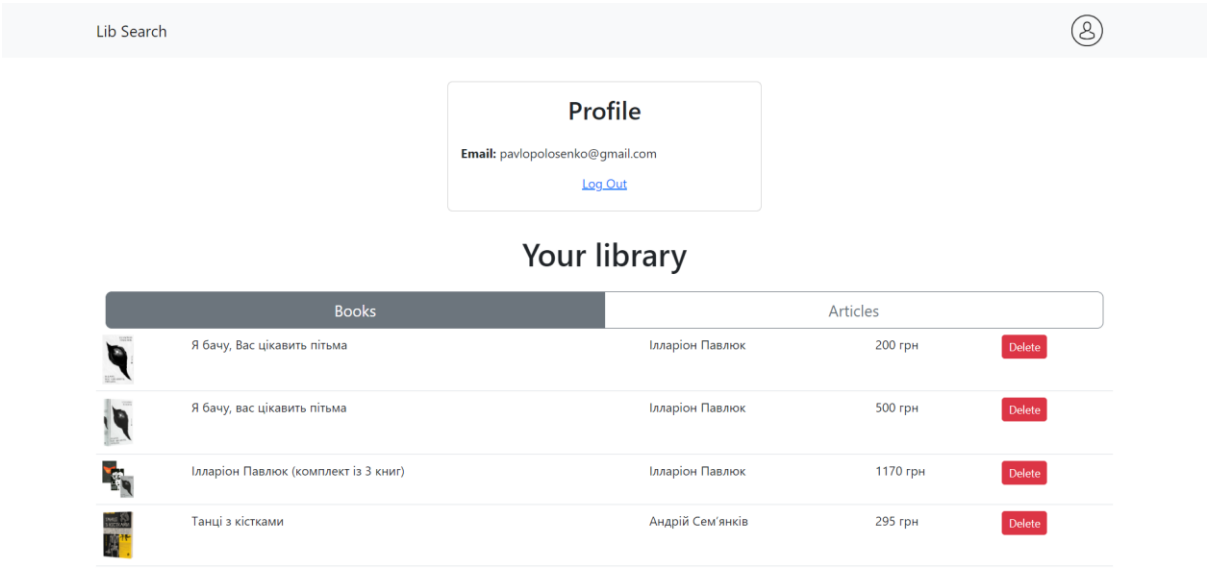

Рисунок 3.6 – Сторінка акаунту

Також присутня можливість пошуку статей. Статті можна шукати за допомогою ключових слів, авторів та назви. При натисканні на назву статті можна перейти на ресурс звідки була взята стаття. Також статті можна сортувати за трьома полями це: відповідність, дата останнього оновлення та дата подання. На рисунку 3.7 зображено головну сторінку з відкритою вкладкою зі статтями.

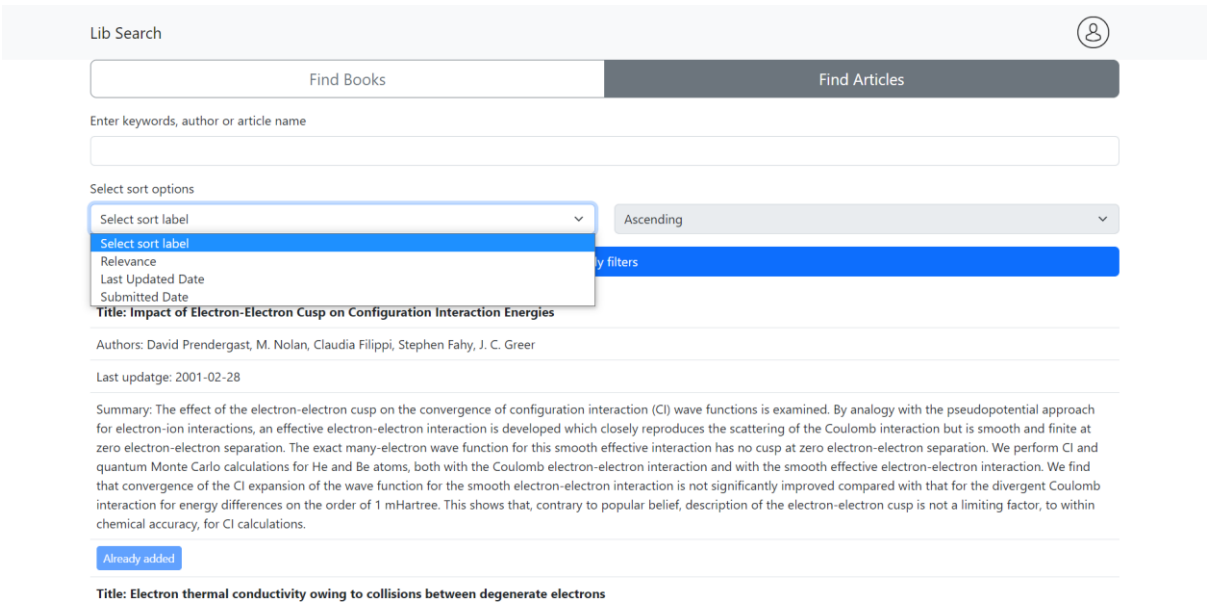

Рисунок 3.7 – Головна сторінка з статтями

Також статті можна додавати до улюблених на рисунку 3.8 зображено сторінку акаунту з відкритими доданими статтями.

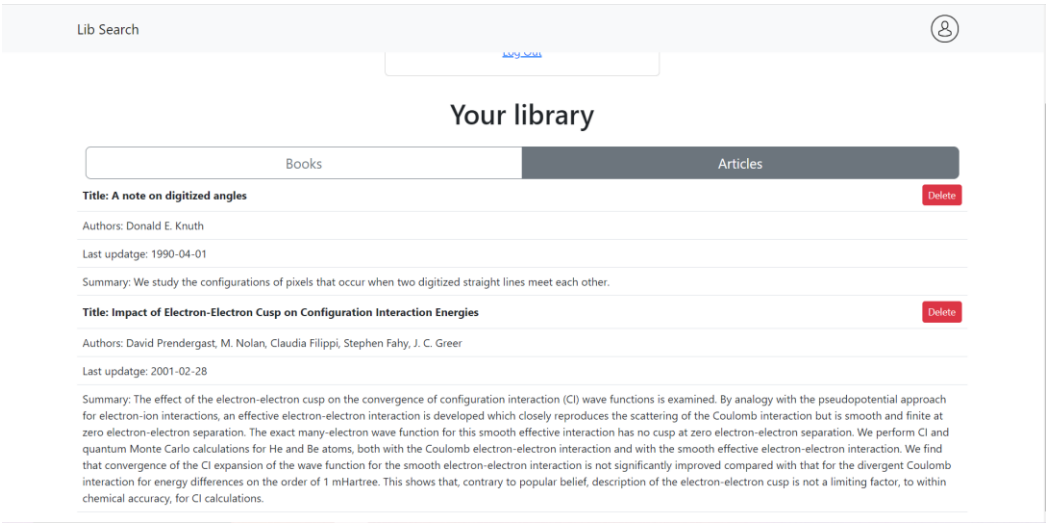

Рисунок 3.8 – Сторінка акаунту з доданими статтями

Висновок до розділу.

Під час виконання цього розділу було спроектовано веб-сервіс пошуку літератури. Також було обрано відповідні фреймворки і було обрано базу даних для збереження інформації. Також було сформовано інструкцію користувача.

ВИСНОВОК

В результаті виконаної дипломної роботи було розглянуто важливість літератури в сучасному світі також було висвітлено проблему пошуку літератури. Було проаналізовано тенденцію розвитку веб-сервісів в Україні. Було розглянуто і проаналізовано технології розробки веб-сервісів та архітектурні рішення. Було розглянуто особливості популярних систем пошуку літератури.

Також було проведено аналіз сценаріїв використання, розглянуто види архітектур, їх особливості використання, переваги та недоліки. Також було більш детально розглянуто кожну архітектуру окремо. Було здійснено аналіз найсучасніших баз даних і розроблено результуючу таблицю їх порівняння.

В результаті було спроектовано веб-сервіс пошуку літератури. Також було обрано відповідні фреймворки і було обрано базу даних для збереження інформації. Також було сформовано інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка веб сервісів з нуля : веб-сайт. URL: https://webcase.com.ua/uk/web-services-development/ (дата звернення: 12.03.2023).
2. What are Web Services? : веб-сайт. URL: https://www.tutorialspoint.com/webservices/what_are_web_services.htm (дата звернення: 12.03.2023).
3. Layered Architecture | Baeldung on Computer Science : веб-сайт. URL: https://www.baeldung.com/cs/layered-architecture (дата звернення: 13.03.2023).

4. Мікросервісна архітектура для початківців : веб-сайт. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/ (дата звернення: 14.03.2023).
5. Richardson L., Ruby S. RESTful Web Services : навч. посіб., 2007. 448 с.
6. Graham S. Building Web Services with Java : навч. посіб., 2001. 704 с.
7. Laurent, S. S., Johnston, J. Programming Web Services with XML-RPC : навч. посіб., 2002. 288 с.
8. Richardson, L., Amundsen, M. RESTful Web APIs : навч. посіб., 2013. 268 с.
9. Alonso, G. Web Services: Concepts, Architectures and Applications : навч. посіб., 2004. 336 с.
10. Seely, S. Building Web Services with Microsoft Azure : навч. посіб., 2015. 306 с.
11. Cerami, E. Web Services Essentials : навч. посіб., 2002. 224 с.
12. Kalin, M. Java Web Services: Up and Running : навч. посіб., 2013. 320 с.
13. Mulloy, B. Web API Design: Crafting Interfaces that Developers Love : навч. посіб., 2014. 120 с.
14. "Web Services Architecture" : веб-сайт. URL: https://www.w3.org/TR/ws-arch/ (дата звернення: 16.03.2023).
15. "Web Services Description Language (WSDL)" : веб-сайт. URL: https://www.w3.org/TR/wsdl/ (дата звернення: 16.03.2023).
16. "Simple Object Access Protocol (SOAP)" : веб-сайт. URL: https://www.w3.org/TR/soap/ (дата звернення: 17.03.2023).
17. "Representational State Transfer (REST)" : веб-сайт. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 18.03.2023).
18. "RESTful Web Services" : веб-сайт. URL: https://martinfowler.com/articles/richardsonMaturityModel.html (дата звернення: 18.03.2023).

19. "Building Web Services with Java" : веб-сайт. URL: https://docs.oracle.com/javaee/6/tutorial/doc/bnayn.html (дата звернення: 16.03.2023).
20. "Web Services Security" : веб-сайт. URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss (дата звернення: 20.03.2023).
21. "arXiv API User's Manual" : веб-сайт. URL: https://info.arxiv.org/help/api/user-manual.html#Architecture (дата звернення: 04.04.2023).