МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

кафедра «Інформаційні системи та мережі»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи на тему:

Інформаційна система контролю сервісів, що надаються користувачу в мережі Інтернет

Студента групи ІТ-41, Шельваха М.А.
(шифр, прізвище та ініціали)

Керівник роботи _____(__Віктор ГРИГОРОВИЧ__)

Консультант _____(_____)

_____(_____)

Нормоконтроль _____(___Андрій ВАСИЛЮК__)

Завідувач кафедри ІСМ _____ (**Дмитро ДОСИН**)

«_04_» _червня__ **2025 р.**

ЛЬВІВ – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій

Кафедра «Інформаційні системи та мережі»

Спеціальність 126 "Інформаційні системи та технології"

Перший (бакалаврський) рівень вищої освіти

ОПП "Інтелектуальні інформаційні системи"

«ЗАТВЕРДЖУЮ»

Завідувач кафедри ІСМ _____

«_04_» _червня_ 2025 р.

ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу студента групи ІТ-41

Шельваха Максима Андрійовича

(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна система контролю сервісів, що надаються користувачу в мережі Інтернет затверджена наказом по НУ «ЛП» від «_11_»__березня__2025р. №_866-4-08_

2. Термін здачі студентом закінченої роботи 30.05.2025р.

3. Вихідні дані для роботи: літературні джерела, електронні ресурси, методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів освітньо-професійної програми спеціальності 126 «Інформаційні системи та технології» першого (бакалаврського) рівня вищої освіти

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити): провести аналіз предметної області, провести аналіз існуючих рішень, провести системний аналіз, обрати програмні засоби та технології для реалізації системи, реалізувати систему

5. Перелік графічного матеріалу: дерево цілей, знімки екрану, діаграми IDEF0, ієрархія завдань

3

6. Перелік програмних продуктів, які належить використати в процесі розроблення роботи (проекту): Opera GX, Visual Studio Code, Rest API, HTML/CSS (веб-сайт), JavaScript, React, Node.js, Express.js, Microsoft Word, AllFusion Process Modeler.

7. Консультування роботи, із зазначенням розділів роботи

Розділ	Консультанти	Підпис, дата	
		завдання видав	завдання отримав

8. Дата, коли видано завдання 24.02.2025 р.

Керівник _____

(підпис)

Завдання отримав до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Етапи кваліфікаційної роботи	Термін виконання етапів роботи	Примітки
1	Збір інформації	01.03.2025 - 01.04.2025	Виконано
2	Аналітичний огляд літературних та інших джерел	02.04.2025 - 08.04.2025	Виконано
3	Системний аналіз об’єкту дослідження	09.04.2025 - 11.04.2025	Виконано
4	Програмні засоби розв’язання задачі	12.04.2025 - 17.04.2025	Виконано
5	Практична реалізація	18.04.2025 - 23.04.2025	Виконано
6	Оформлення кваліфікаційної роботи	24.04.2025 - 07.05.2025	Виконано

Студент _____

(підпис)

Керівник роботи _____

(підпис)

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах стрімкого розвитку мережі Інтернет перед провайдерами різноманітних сервісів постає проблема своєчасного реагування на запити користувачів і постійної адаптації до їх потреб. Застарілі підходи та відсутність інновацій можуть призвести до значного зниження лояльності клієнтів, втрати конкурентних позицій та відчутних фінансових втрат. Ситуація з пандемією Covid-19 продемонструвала, наскільки важливим є надання онлайн-сервісів різного типу – від банківських послуг до розваг і навчання [12]. Через карантинні обмеження багато компаній змушені були перейти в онлайн, однак не всі виявилися готовими ефективно контролювати й забезпечувати якісний сервіс.

На ринку послуг в мережі Інтернет постійно зростає кількість сервісів, що пропонують користувачам свій спектр функціональності. У такому висококонкурентному середовищі виграють ті, хто надає зручні, швидкі та безпечні сервіси з орієнтацією на потреби клієнта. Однак багато сервісів стикаються з тим, що їхнє охоплення не є великим і не поширюється темпами, які очікувала сама компанія. Усе це зумовлює необхідність дослідження та впровадження сучасної інформаційної системи, здатної збирати такі сервіси, класифікувати їх та надавати користувачам для користування.

Мета і задачі дослідження. Метою цієї роботи є проєктування та розробка інформаційної системи, яка б забезпечувала ефективний контроль і управління сервісами, що надаються користувачу в мережі Інтернет. Для досягнення цієї мети передбачено здійснення таких ключових задач:

- Провести комплексний аналіз предметної області, зокрема вивчити принципи функціонування сучасних інформаційних систем у контексті контролю і забезпечення сервісів онлайн.
- Дослідити існуючі аналоги та провести детальний аналіз їх функціональних можливостей і обмежень.

- Використовуючи методи системного аналізу, побудувати дерево цілей, визначити структуру майбутньої системи та розробити концептуальну модель її бази даних.
- Обґрунтувати вибір оптимальних технологій та інструментів реалізації системи.
- Створити прототип інформаційної системи контролю сервісів, провести тестування розробленого програмного забезпечення та оцінити результати.

Об’єкт дослідження. Об’єктом дослідження є процеси надання та контролю сервісів користувачам в мережі Інтернет.

Предмет дослідження. Предметом дослідження є методи та засоби розробки та програмного втілення інформаційної системи, що гарантує накопичення інформації, відслідковування, вивчення і розподіл сервісів, а також надання цих сервісів користувачам відповідно до їх потреб [18]. Особлива увага приділяється етапам, які найбільше впливають на безперебійність та зручність користування системою: від моменту реєстрації та внесення вподобань до формування результату в реальному часі. Саме на цих етапах найбільш імовірним є виникнення помилок, що впливають на загальну продуктивність і надійність системи.

Практичне значення одержаних результатів. У результаті розробки даної інформаційної системи створено основу (каркас), здатну контролювати процес надання сервісів та забезпечувати користувачів бажаними продуктами (сервісами). Запропонована система має широкі можливості для масштабування, зміни конфігурацій під різні бази даних та гнучких налаштувань у реальному часі. Це дозволить користувачам отримувати в пропозиції більше сервісів за їхніми вподобаннями, а власникам сервісів – більше охоплення та рекламу.

РОЗДІЛ 1

Аналітичний огляд літературних та інших джерел

1.1 Основні засади дослідження

Інтеграція веб-платформ і мобільних додатків у повсякденне життя користувачів суттєво змінила спосіб взаємодії з онлайн-сервісами. Раніше користувачам доводилося самостійно шукати потрібні послуги, переглядати безліч сайтів, порівнювати їх вручну, а також зберігати посилання в закладках або нотатках. Сьогодні ж ці процеси автоматизовані завдяки спеціалізованим платформам, які надають централізований доступ до сотень сервісів з можливістю фільтрації, збереження, персоналізації та контролю. Це не лише робить користування зручнішим, але й допомагає компаніям краще розуміти потреби своєї аудиторії та адаптуватися до них.

З появою повноцінних інформаційних систем змінився і підхід до організації сервісів. Тепер користувачі не прив’язані до одного джерела інформації - платформи дозволяють не лише переглядати доступні онлайн-сервіси, а й самостійно додавати нові, класифікувати їх, обирати за своїми вподобаннями, а також додавати особисті позначки або теги. Водночас адміністрація таких систем отримує можливість перевіряти, управляти та позначати перевірені сервіси, зберігаючи їх у надійній базі.

Потреба в таких системах стала особливо актуальною під час пандемії Covid-19, коли більшість наших взаємодій перейшла в онлайн: оплата комунальних послуг, подання заявок, реєстрація на прийом, замовлення доставки, дистанційне навчання [4] [12]. Проте, потреба в швидкому орієнтуванні в цих нових сервісах залишилася такою ж важливою. А з початком повномасштабного вторгнення у 2022 році зросла потреба в платформах, які дозволяють централізовано отримувати доступ до важливих сервісів - як державних, так і приватних - швидко та безпечно.

Таким чином, виникає запит на створення інформаційної системи контролю сервісів, що надаються користувачу в мережі Інтернет, яка б об'єднувала такі функції:

- реєстрація користувачів та створення особистих кабінетів;
- персоналізація контенту: вибір категорій сервісів за вподобаннями;

- рекомендаційна система на основі обраного профілю;
- збереження улюблених сервісів і додавання власних тегів;
- можливість додавання сервісів самими користувачами із подальшою модерацією;
- маркування перевірених сервісів адміністрацією;

Враховуючи сучасні потреби користувачів, можна виділити кілька важливих аргументів на користь створення таких систем:

- Зручність: Користувачі можуть швидко знайти потрібний сервіс, не витрачаючи час на ручний пошук.
- Швидкість прийняття рішень: Упорядкована інформація, фільтри та рекомендації допомагають приймати рішення швидко і впевнено.
- Персоналізація: Кожен користувач формує свій унікальний інформаційний простір.
- Контроль та безпека: Завдяки модерації та маркуванню «перевірено», користувач отримує більше довіри до сервісу.

Інформаційна система такого типу – це не просто сайт з переліком послуг, а багатофункціональна платформа, яка слугує інструментом навігації, взаємодії та контролю в цифровому середовищі. Саме тому створення такої системи є актуальним і логічним кроком у напрямку розвитку зручної, прозорої та безпечної взаємодії користувача з сучасними онлайн-сервісами [28].

1.2 Стан та перспективи досліджень

Класифікація інформаційних систем, які автоматизують взаємодію користувачів із онлайн-сервісами, є досить складним завданням. Це пов'язано з величезною різноманітністю платформ, постійними змінами в світі цифрових технологій і різними запитами як від користувачів, так і від власників онлайн-бізнесу [6]. Хоча ці системи можуть мати спільні функції, такі як авторизація, фільтрація, збереження

вподобаного та рекомендації, підходи до їх реалізації, дизайн і логіка взаємодії часто суттєво відрізняються.

Умовно, інформаційні системи для контролю онлайн-сервісів можна поділити на два основні типи:

«Сервіс-Користувач» – це система, розроблена спеціально для одного конкретного онлайн-сервісу (або його власника), з акцентом на автономність, брендування та персоналізовані сценарії взаємодії. Такий сайт або додаток зазвичай орієнтований на вузький функціонал, але має гнучкість у реалізації індивідуальних рішень, наприклад, інтерфейс, що адаптується до геолокації користувача або спеціальні акції для певних категорій клієнтів.

«Платформа-Користувач» – це система, яка об'єднує різноманітні незалежні або споріднені сервіси на одній платформі. Вона дає можливість користувачеві порівнювати сервіси за різними характеристиками та рейтингами, зберігати улюблені варіанти, додавати власні нотатки, а також отримувати автоматичні рекомендації. Така система охоплює широкий сегмент ринку і має потенціал для масштабування.

Обидва типи інформаційних систем націлені на автоматизацію рутинних завдань, таких як реєстрація, вибір сервісу, фільтрація та збереження [11]. Вони також покликані підвищити зручність для користувачів і забезпечити аналітичні дані для подальшого вдосконалення сервісу. Проте, кожен з цих підходів має свої плюси і мінуси.

Системи «Сервіс-Користувач» надають максимальний контроль для власників сервісу, дозволяють впроваджувати індивідуальні рішення та зосереджуватись на специфічних функціях. Але варто зазначити, що вони мають обмежене охоплення, потребують більших витрат на розробку і часто не підтримують складні функції, які характерні для масштабованих платформ.

Системи «Платформа-Користувач» дозволяють зручно взаємодіяти з безліччю сервісів, надаючи можливість порівняльного аналізу, покращуючи видимість менш популярних проєктів і знижуючи витрати на обслуговування завдяки масштабам.

Однак такі платформи потребують складної технічної підтримки, і в них може зникати індивідуальність окремих сервісів [9].

Успішність впровадження інформаційної системи контролю онлайн-сервісів значною мірою залежить від кількох важливих чинників, таких як:

- готовність користувачів до взаємодії з платформою;
- адаптація системи до вже існуючих цифрових звичок;
- забезпечення кібербезпеки даних;
- зручність використання та адаптивність інтерфейсу;
- регулярне оновлення і масштабування функціоналу.

З огляду на це, майбутні дослідження таких систем відкривають безліч нових і цікавих напрямків:

Інтеграція штучного інтелекту – для автоматичного створення рекомендацій, які базуються на діях користувачів, персоналізації головної сторінки, аналізу вподобань та раннього виявлення потреб.

Покращення користувацького досвіду – через впровадження адаптивного дизайну, зрозумілих інтерфейсів, функцій розширеного пошуку, голосового введення та інтеграції з месенджерами [35].

Аналітика та інсайти – використання методів машинного навчання для аналізу дій користувачів, виявлення популярних тегів і сервісів, а також створення автоматизованих дашбордів для адміністраторів.

В цілому, глибше розуміння різних типів інформаційних систем для взаємодії з онлайн-сервісами допомагає створити більш ефективну, персоналізовану та масштабовану платформу [17]. Це не лише задовольняє потреби користувачів, а й підвищує загальну ефективність цифрової екосистеми.

1.3 Аналіз відомих програмних систем

У сучасному світі, якщо ви хочете створити будь-яку цифрову платформу або організацію, яка функціонує в інформаційному середовищі, вам просто необхідно використовувати сучасні технології та розвинене програмне забезпечення. Інакше ви

ризикуєте знизити ефективність, витратити зайві гроші та зусилля, а також втратити потенційних користувачів. Це особливо важливо для галузей, які активно взаємодіють із кінцевими споживачами в онлайн-середовищі, таких як IT, електронна комерція, мультимедійні послуги та освітні сервіси. Автоматизація процесів, централізація управління та персоналізація доступу до сервісів стають критично важливими [18].

На українському ринку сьогодні є безліч сервісів, які допомагають користувачам знаходити та використовувати онлайн-послуги. Це можуть бути платформи для оплати послуг, запису до лікаря, подання документів до державних органів, а також спеціалізовані освітні, розважальні чи комунікаційні інструменти. Більшість з них представлені у вигляді веб-сервісів або мобільних додатків, які можна завантажити з відповідних магазинів (Google Play, App Store тощо). Деякі з цих сервісів мають обмежений набір функцій або орієнтовані на певну аудиторію[21].

Незважаючи на це, більшість систем мають спільні функції:

- Пошук сервісів за назвою, категорією, тегами або коротким описом;
- Фільтрація сервісів за різними параметрами (тип, рейтинг, популярність, новизна);
- Можливість зберігати сервіси у власному списку або переглянути раніше;
- Додавання власних тегів, коментарів або вподобань;
- Контроль доступу до сервісів – авторизація, підтвердження електронною поштою або через інтеграцію зі сторонніми платформами;
- Інформація про доданий сервіс, включаючи посилання на офіційний сайт, опис, категорію, мову інтерфейсу, відгуки та інше;
- Механізми перевірки або верифікації сервісів, які були додані до платформи адміністратором або модераторами.

Незважаючи на схожий функціонал, кожен з цих сервісів має свої плюси і мінуси, з якими користувачі стикаються під час використання. У деяких випадках обмеженням є закритість функцій, в інших – відсутність модерації або занадто складний інтерфейс.

Щоб знайти найкращі підходи до реалізації інформаційної системи контролю сервісів, варто провести огляд і аналіз кількох відомих аналогічних систем, які вже працюють на ринку. Це допоможе:

- виявити типові сценарії використання;
- зрозуміти потреби користувачів;
- визначити недоліки, яких можна уникнути у власній реалізації;
- обґрунтувати доцільність створення нової системи, враховуючи сильні сторони існуючих рішень.

Щоб зрозуміти, чому варто створити інформаційну систему для контролю сервісів, давайте розглянемо вже існуючі сервіси, які частково виконують цю функцію. Хоча вони не можуть повністю замінити запропонований проєкт, їх аналіз допоможе виявити основні тенденції та потреби користувачів.

«Product Hunt» – це онлайн-платформа, яка дозволяє користувачам відкривати нові цифрові продукти, такі як сервіси, застосунки, розширення та інструменти [23]. Вона в основному зосереджена на новинках у світі технологій, стартапів і програмного забезпечення. Користувачі можуть не лише переглядати список продуктів, але й додавати свої власні, коментувати, голосувати та створювати добірки. На рис. 1.1 зображено головну сторінку платформи з прикладами популярних сервісів.

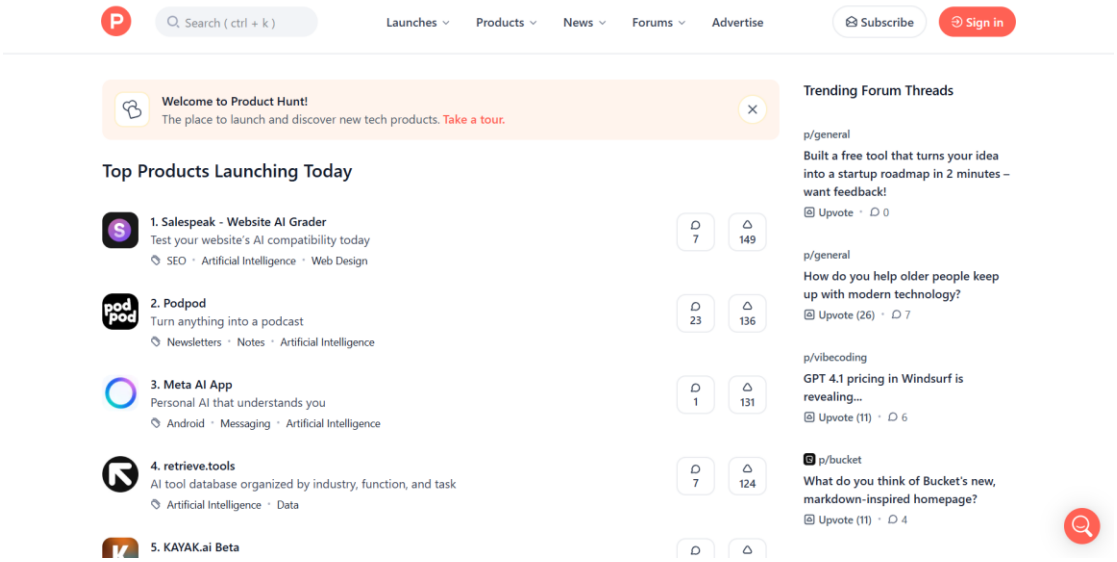

Рис. 1.1. Головна сторінка «Product Hunt».

Основні переваги:

- Швидкий пошук нових сервісів
- Голосування за проєкти
- Можливість додавання сервісу кожним користувачем
- Щоденні дайджести новинок
- Коментування та обговорення продуктів

Хоча «Product Hunt» дуже зручний, він не має системи персоналізованих вподобань або фільтрації за інтересами конкретного користувача. Вся структура базується на популярності, а не на релевантності. Крім того, теги створюються вручну і не мають чіткої системи, що ускладнює пошук.

«AlternativeTo» – сервіс, який допомагає знайти альтернативу будь-якій програмі чи вебсайту [1]. Користувач просто вводить назву відомого сервісу, і система миттєво формує список альтернатив з короткими описами, рейтингами та позначками (безкоштовний, платний, платформа, ліцензія). Інтерфейс платформи можна побачити на рис. 1.2.

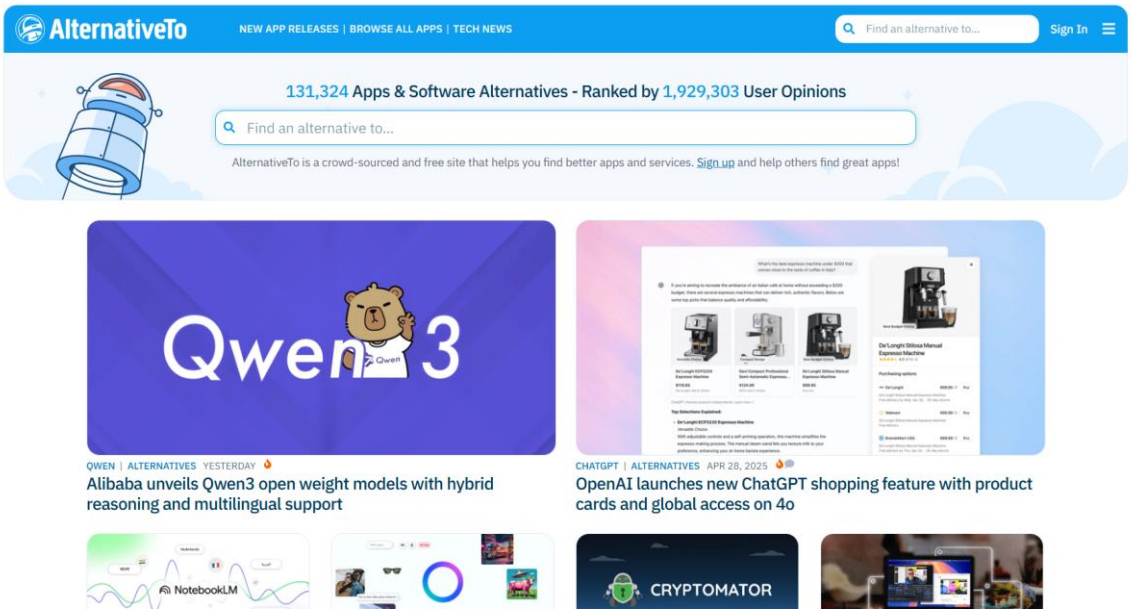

Рис. 1.2. Головна сторінка «AlternativeTo».

Переваги:

- Дуже зручний пошук альтернатив для будь-якого цифрового продукту

- Велика база сервісів
- Підтримка фільтрів за платформою та типом ліцензії
- Оцінки та короткі рецензії від користувачів

Але варто зазначити, що платформа не має авторизації або особистого кабінету. Всі добірки є тимчасовими. Також немає можливості зберігати улюблені сервіси, додавати свої теги або створювати повноцінні облікові записи з персоналізованим контентом.

«SimilarSites» – сервіс, що допомагає знайти сайти, які схожі на той, що ви ввели [33]. Просто введіть URL, наприклад «youtube.com», і ви отримаєте список сайтів з подібною аудиторією або функціоналом. На рис. 1.3 ви можете побачити приклад результату пошуку.

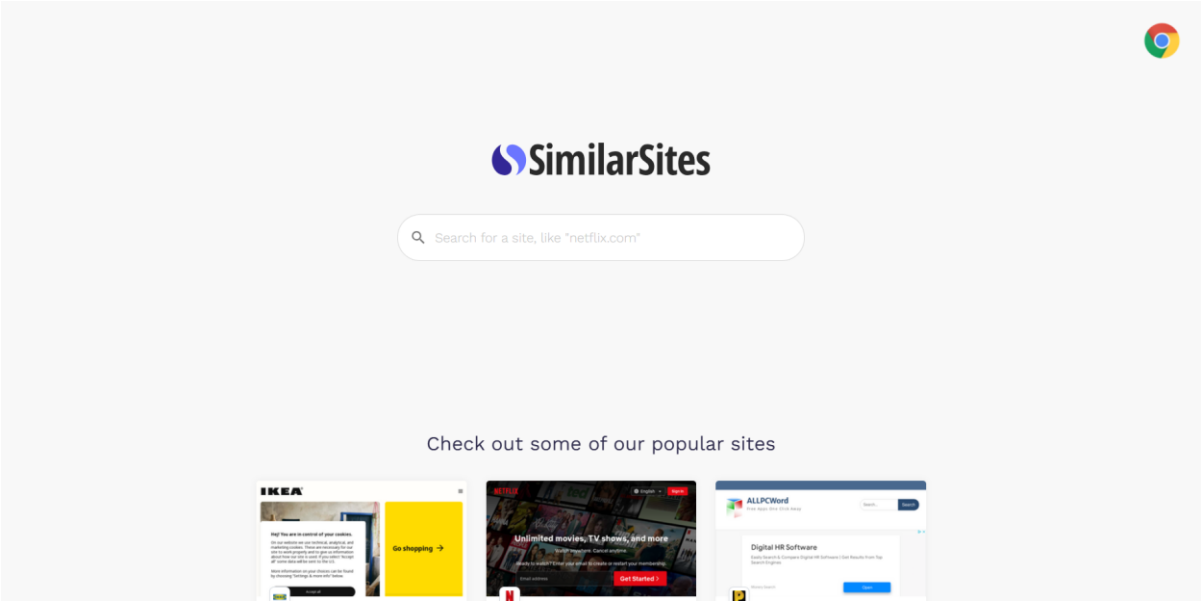

Рис. 1.3. Головна сторінка «SimilarSites».

Переваги:

- Миттєвий результат за введеним сайтом
- Корисно для виявлення конкурентів
- Простий інтерфейс без реєстрації

Проте, «SimilarSites» більше схожий на інструмент для одноразового використання, ніж на повноцінну платформу. Тут неможливо створювати профілі,

обирати категорії або зберігати знайдені сервіси. Також ви не можете додавати свої сайти або отримувати персоналізовані рекомендації.

Отже, після детального аналізу представлених аналогічних сервісів, можна виділити кілька ключових аспектів, на які варто звернути особливу увагу при проєктуванні інформаційної системи контролю сервісів. По-перше, це функціональні можливості системи, швидкість роботи, стабільність функціонування, а також обробка можливих виняткових ситуацій (наприклад, додавання некоректного сервісу, спроба подвійного запису, відсутність мережі тощо). Інші аспекти – такі як дизайн, адаптивність, наявність мобільного додатку або додаткові візуальні функції – безумовно, покращують враження від платформи та підвищують її конкурентоспроможність, але не є вирішальними для базової ефективності системи.

Щоб підсумувати результати аналізу та створити основу для подальшої розробки, нижче наведено порівняльну таблицю основних функцій, реалізованих в аналізованих сервісах. Вона дозволяє візуально порівняти можливості кожної з платформ:

Таблиця 1.1

Порівняльна таблиця сервісів

Сервіс / Функціонал	Product Hunt	AlternativeTo	SimilarSites
Багатофункціональний акаунт	+	+	-
Персональні вподобання	-	-	-
Додавання власних сервісів	+	-	-
Рекомендації на основі вподобань	-	-	-
Система тегів	+	+	-
Маркування «Перевірено»	-	-	+
Класифікація та фільтрація	+	+	-

Висновок до розділу 1

Враховуючи проведений аналіз теоретичних основ, сучасного стану досліджень та вже реалізованих аналогів, можна дійти висновку, що одним із найважливіших завдань при створенні інформаційної системи контролю сервісів є вибір оптимальної архітектури платформи та надійної бази даних. Вона повинна забезпечувати швидке оброблення запитів, стабільне зберігання великої кількості записів, а також можливість масштабування з урахуванням потенційного зростання користувачів і сервісів.

Не менш важливою є зручність взаємодії користувача з системою. Як показав аналіз існуючих платформ, критично важливими для успіху є: персоналізація, простота навігації, швидкість пошуку сервісів, можливість збереження та класифікації обраного, а також чітко виражена цінність для користувача (у вигляді рекомендацій чи додаткового функціоналу).

Щоб виділитися серед конкурентів, майбутня система повинна не просто копіювати функції існуючих сервісів, а враховувати їхні недоліки, впроваджувати нові підходи до організації даних та мати чітко визначену цінність для користувача.

РОЗДІЛ 2

Системний аналіз об’єкта дослідження

Системний аналіз – це структурований і цілеспрямований підхід до вивчення складних систем, будь то бізнес-процеси, програмні продукти чи соціальні явища. Його головна мета полягає в тому, щоб зрозуміти логіку роботи системи, її цілі, структуру та зв’язки з навколишнім середовищем. Уявіть собі, що це як розбирання механізму на частини, щоб зрозуміти, як він працює і як окремі елементи взаємодіють, забезпечуючи цілісність системи.

Цей підхід не обмежується лише вирішенням конкретних проблем – він також допомагає виявляти слабкі місця, знаходити можливості для покращення або навіть розробляти абсолютно нові рішення, які відповідають актуальним потребам користувачів.

Системний аналіз включає кілька ключових етапів. Спочатку визначаються межі досліджуваної системи: які її основні компоненти та як вони взаємодіють між собою. Потім аналізуються входи та виходи – які ресурси або дані надходять у систему і які результати вона генерує. Особливу увагу також звертають на зовнішні чинники та обмеження, які можуть впливати на ефективність функціонування системи, зокрема технічним, економічним чи часовим аспектам.

Одним із найефективніших інструментів системного аналізу є моделювання. Створення спрощеного уявлення про систему – через графічні діаграми або математичні моделі – допомагає аналітикам краще зрозуміти її поведінку. Такі моделі відкривають можливість перевіряти різні сценарії розвитку подій, вносити зміни без ризику для реальної системи та прогнозувати наслідки тих чи інших управлінських рішень.

Системний підхід – це не просто набір технік, а цілісна філософія дослідження. Він ґрунтується на чіткому дотриманні методології, що забезпечує всебічне охоплення об'єкта аналізу та послідовність дій. Завдяки цьому можна досягти глибокого розуміння суті проблем, вийти за межі поверхневих спостережень і сформувати дієві рішення, орієнтовані на довгостроковий результат.

В рамках даної роботи системний аналіз слугує основою для збагнення роботи інформаційної системи контролю онлайн-сервісів, яку планується створити. Саме системний підхід дозволив впорядкувати об'єкт дослідження, визначити його основні елементи, вибудувати логіку взаємодії користувачів із платформою та створити основу для майбутнього зростання.

2.1 Дерево цілей

Дерево цілей, або цільова ієрархія, являє собою ключовий інструмент системного аналізу, що надає можливість виразно окреслити, розташувати за пріоритетами та логічно структурувати завдання та орієнтири системи чи проєкту [13]. Цей підхід допомагає візуалізувати очікувані результати, формувати узгоджені етапи досягнення мети та забезпечує єдину логіку функціонування розроблюваної системи.

Основна задача дерева цілей – чітко сформулювати кінцеву мету, розділивши її на проміжні цілі та конкретні завдання. Цей підхід допомагає не тільки визначити ключову концепцію системи, але й розбити її на елементи, які можливо втілити в рамках проєкту. Власне, так дерево цілей служить логічною основою для стратегічного планування, раціонального розподілу ресурсів та прийняття обґрунтованих рішень на кожному етапі функціонування системи [10].

Загалом, структура дерева цілей складається з трьох основних рівнів:

Головна ціль – це те, що визначає загальну мету створення системи або її глобальне призначення. Це той кінцевий результат, до якого спрямовані всі дії та рішення в рамках проєкту.

Функціональні підцілі – вони деталізують головну ціль, розбиваючи її на конкретні напрямки реалізації. Це своєрідні орієнтири, за якими можна оцінити прогрес у досягненні мети.

Оцінювальні критерії – це кількісні або якісні показники, які допомагають визначити, чи були досягнуті поставлені цілі. Саме на цьому етапі закладаються

основи для контролю ефективності системи та аналізу результативності її впровадження.

Правильно побудоване дерево цілей допомагає глибше зрозуміти функціональне призначення системи, забезпечує логічну зв’язність її компонентів і дозволяє сформувати обґрунтовані вимоги до проєктування [36]. Визначаючи чіткі завдання та критерії успішності, зацікавлені сторони можуть системно контролювати реалізацію, коригувати стратегії та підвищувати загальну ефективність розробки.

Що стосується створення дерева цілей для нашої системи, основною метою є розробка інформаційної системи контролю сервісів, що надаються користувачу в мережі Інтернет. Вона має забезпечити централізований доступ до ресурсів, персоналізувати взаємодію та підвищити довіру користувачів до інтернет-сервісів. По суті, це веб-платформа, яка дозволяє звичайним користувачам швидко знаходити потрібні онлайн-сервіси, додавати свої, зберігати обране та орієнтуватися серед великої кількості інформації завдяки модерації та системі тегів.

Відповідно до принципів побудови дерева цілей, доцільно визначити три ключові підцілі, які конкретизують загальну мету. З огляду на функціональність системи та запити цільової аудиторії, логічно виділити такі напрямки [37] [38]:

- Покращення користувацького досвіду (UI/UX) і персоналізація
- Централізований контроль і модерація онлайн-сервісів
- Аналітика та формування рекомендацій

Перед тим як перейти до критеріїв оцінки ефективності системи або її візуалізації, важливо деталізувати ці три напрямки, виокремити для кожного з них ключові задачі та визначити пріоритетний напрямок – той, що має найбільше значення.

1. Персоналізація взаємодії з сервісами

Підпункти:

- Реєстрація користувача та створення особистого кабінету
- Вибір категорій і тегів за інтересами
- Збереження улюблених сервісів

2. Якість та достовірність сервісної інформації

Підпункти:

- Перевірка сервісів адміністрацією
- Позначення сервісів як «перевірених»
- Забезпечення актуальності даних у базі

3. Аналітика та рекомендації

Підпункти:

- Збір статистики щодо використання сервісів
- Вивчення інтересів користувачів (класифікація/кластеризація)
- Генерація персоналізованих рекомендацій

Після ретельного аналізу загальної мети інформаційної системи та деталізації ключових напрямків і завдань, наступним логічним кроком є визначення критеріїв оцінювання. Ці показники формують заключний, але не менш важливий рівень структури дерева цілей.

У контексті інформаційної системи контролю онлайн-сервісів ці критерії виступають як індикатори ефективності. Вони не лише фіксують, наскільки успішно виконуються поставлені завдання, але й дозволяють об'єктивно оцінити, як система працює в реальному середовищі.

Критерії виконують роль вимірювальних орієнтирів, за допомогою яких можна оцінити якість користувацького досвіду, рівень довіри до системи, швидкість її реакції та актуальність запропонованих функцій. Вони також допомагають виявити слабкі місця в реалізації або використанні системи та швидко вносити необхідні корективи.

Нижче наведено перелік основних критеріїв оцінки ефективності впровадження, які були визначені відповідно до функціональних цілей системи:

Актуальність – це про те, як добре система може надати користувачеві саме ту інформацію, яка відповідає його вподобанням і потребам.

Зручність – це оцінка того, наскільки простий інтерфейс, логічна навігація та легкість доступу до основних функцій [34].

Продуктивність – показує, як швидко і ефективно система виконує запити, обробляє дані та формує рекомендації.

Безпека – це про рівень захисту особистих даних користувача та стабільність роботи системи під навантаженням.

Підсумовуючи всі пункти нашої ієрархії цілей, нарешті можна візуалізувати дерево (Рис. 2.1) і зрозуміти, як воно має виглядати в нашій системі.

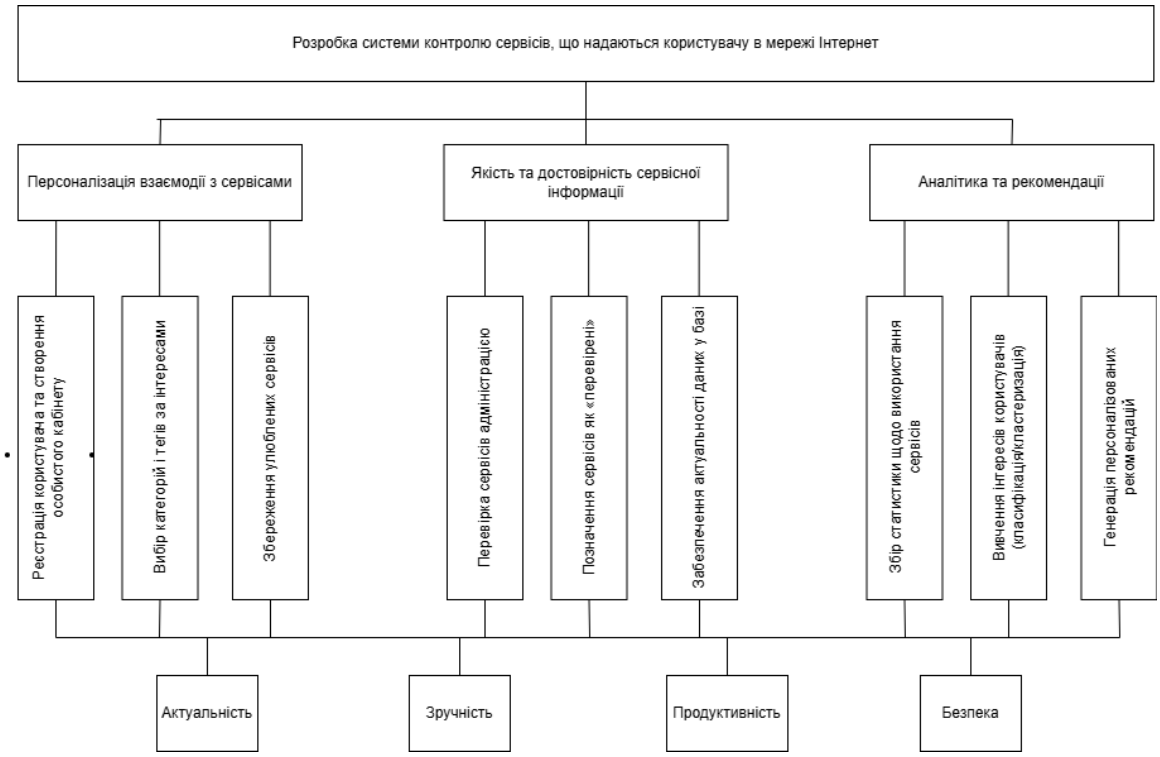

Рис. 2.1. Дерево цілей системи.

2.2 Застосування методу аналізу ієрархій

Для глибшого аналізу функціональних можливостей системи та визначення пріоритетності процесів варто скористатися методом аналізу ієрархій (AHP). Це один з найпопулярніших методів, що допомагає у прийнятті рішень. Цей підхід дозволяє кількісно оцінити важливість окремих елементів системи через попарне порівняння та подальший розрахунок вагових коефіцієнтів [2].

Визначення альтернатив:

- A1 – Система рекомендацій сервісів (СРС) – фокусується на персоналізованій видачі сервісів за інтересами користувача.

- A2 – Платформа каталогізації та тегування (ПКТ) – реалізує класифікацію, додавання тегів та фільтрацію сервісів.
- A3 – Інтерактивна система дій користувача (ІСЗ) – дозволяє додавати власні сервіси користувачу, створювати користувацькі теги.
- A4 – Комбінована інформаційна система (КІС) – поєднує усі перелічені підходи в одному інтерфейсі.

Визначення критеріїв:

- Швидкість взаємодії з користувачем – наскільки оперативно система реагує на дії.
- Можливість персоналізації – наскільки гнучко система адаптується до індивідуальних вподобань.
- Зрозумілість інтерфейсу – зручність для користувачів без технічної підготовки.
- Простота адміністрування – наскільки зручно адміністратору керувати сервісами, тегами, запитами.

У рамках розробки інформаційної системи контролю сервісів, які надаються користувачам в Інтернеті, можна створити ієрархічну модель для оцінки ваги критеріїв якості системи, що впливають на її ефективність.

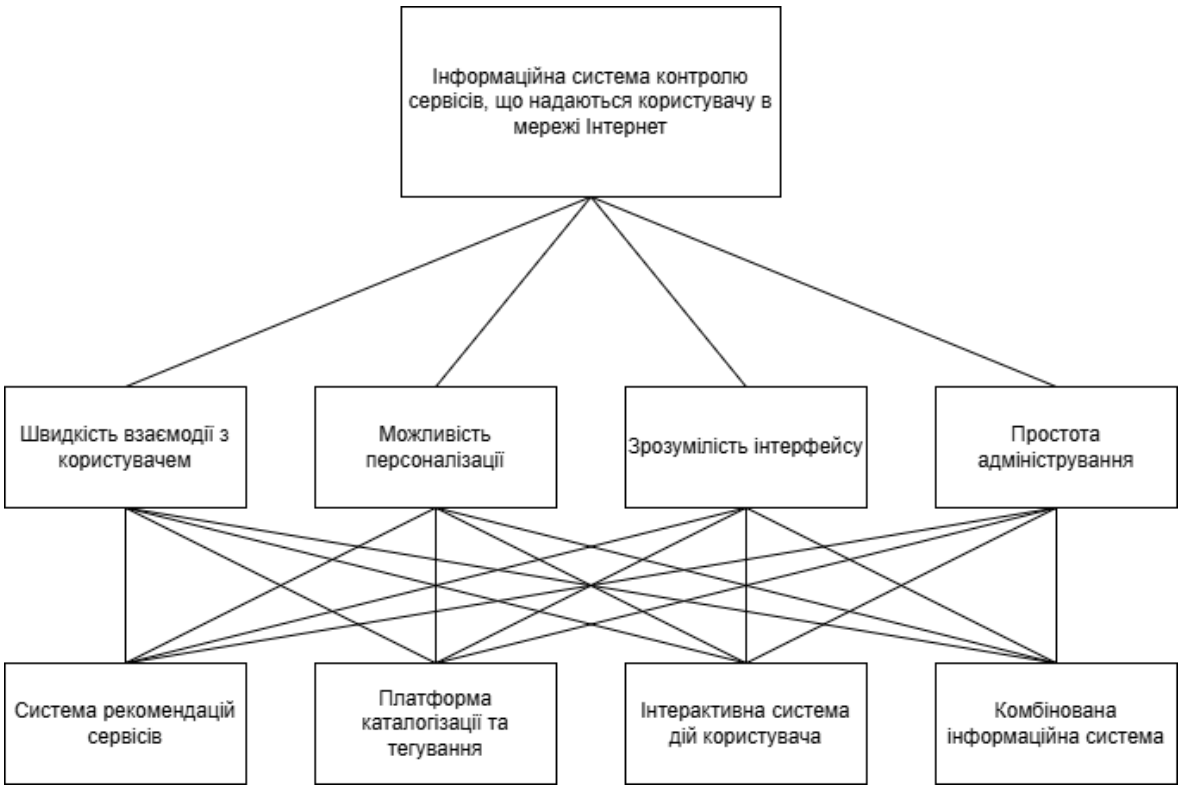

Рис. 2.2. Ієрархія MAI проєктованої системи.

Ціль: Визначити найважливіші критерії якості інформаційної системи контролю сервісів, що надаються користувачу в мережі Інтернет.

У Таблиці 2.1 буде представлено оцінку відносної ваги критеріїв, де за шкалою Сааті від 1 до 9: 1 означає однакову важливість двох критеріїв, а 9 – повну перевагу одного критерію над іншим [29].

Таблиця 2.1

Матриця попарного порівняння

Критерії	Шв. взаємодії	Зрозумілість	Простота адм.	Персоналізація
Швидкість взаємодії	1	2	3	4
Зрозумілість	1/2	1	2	3
Простота адміністр.	1/3	1/2	1	2
Персоналізація	1/4	1/3	1/2	1

Далі проводиться оцінювання альтернатив за кожним критерієм окремо:

Таблиця 2.2

Матриця порівняння альтернатив за критерієм «Швидкість взаємодії з користувачем»

Альтернатива	A1	A2	A3	A4
A1	1	2	3	1/2
A2	1/2	1	2	1/3
A3	1/3	1/2	1	1/5
A4	2	3	5	1

Таблиця 2.3

Матриця порівняння альтернатив за критерієм «Зрозумілість інтерфейсу»

Альтернатива	A1	A2	A3	A4
A1	1	2	3	2
A2	1/2	1	2	1
A3	1/3	1/2	1	1/2
A4	1/2	1	2	1

Таблиця 2.4

Матриця порівняння альтернатив за критерієм «Простота адміністрування»

Альтернатива	A1	A2	A3	A4
A1	1	1/2	1/3	1/4
A2	2	1	1/2	1/3
A3	3	2	1	1/2
A4	4	3	2	1

Таблиця 2.5

Матриця порівняння альтернатив за критерієм «Можливість персоналізації»

Альтернатива	A1	A2	A3	A4
A1	1	2	3	2
A2	1/2	1	2	1
A3	1/3	1/2	1	1/2
A4	1/2	1	2	1

Завершальним етапом є обчислення глобальних пріоритетів:

Таблиця 2.6

Обчислення глобальних пріоритетів альтернатив

Альтернатива	Швидкість (0.465)	Інтерфейс (0.278)	Адміністрування (0.160)	Персоналізація (0.097)	Глобальний пріоритет
Система рекомендацій сервісів (A1)	0.40	0.43	0.10	0.38	0.336
Платформа каталогізації та тегування (A2)	0.24	0.30	0.20	0.30	0.266
Інтерактивна система дій користувача (A3)	0.12	0.14	0.30	0.20	0.174
Комбінована інформаційна система (A4)	0.24	0.13	0.40	0.12	0.224

2.3 Конкретизація функціонування системи

В процесі розробки дієвої інформаційної системи для управління онлайн-сервісами критично важливо конкретно окреслити внутрішню логіку та порядок її функціонування. Одним з найбільш дієвих засобів для цього є метод IDEF0 , який надає можливість візуально показати структуру функцій, їх взаємодію, а також входи, виходи, обмеження і контролюючі впливи.

У багатьох веб-системах функціональні можливості зазвичай описуються у вигляді списків або технічних завдань, що не завжди дає змогу скласти цілісне уявлення про логіку роботи всієї системи. Метод IDEF0 вирішує цю проблему, пропонуючи стандартизовану графічну мову, яка дозволяє візуалізувати повний процес функціонування системи.

Використання IDEF0 у нашій системі дозволяє:

- побудувати послідовну модель взаємодії між користувачами, адміністрацією та аналітичними модулями;
- наочно показати, як користувачі взаємодіють із сервісами, як відбувається модерація контенту та як працюють механізми аналітики та рекомендацій;

- уникнути помилок на етапі планування архітектури завдяки чітко визначеним процесам.

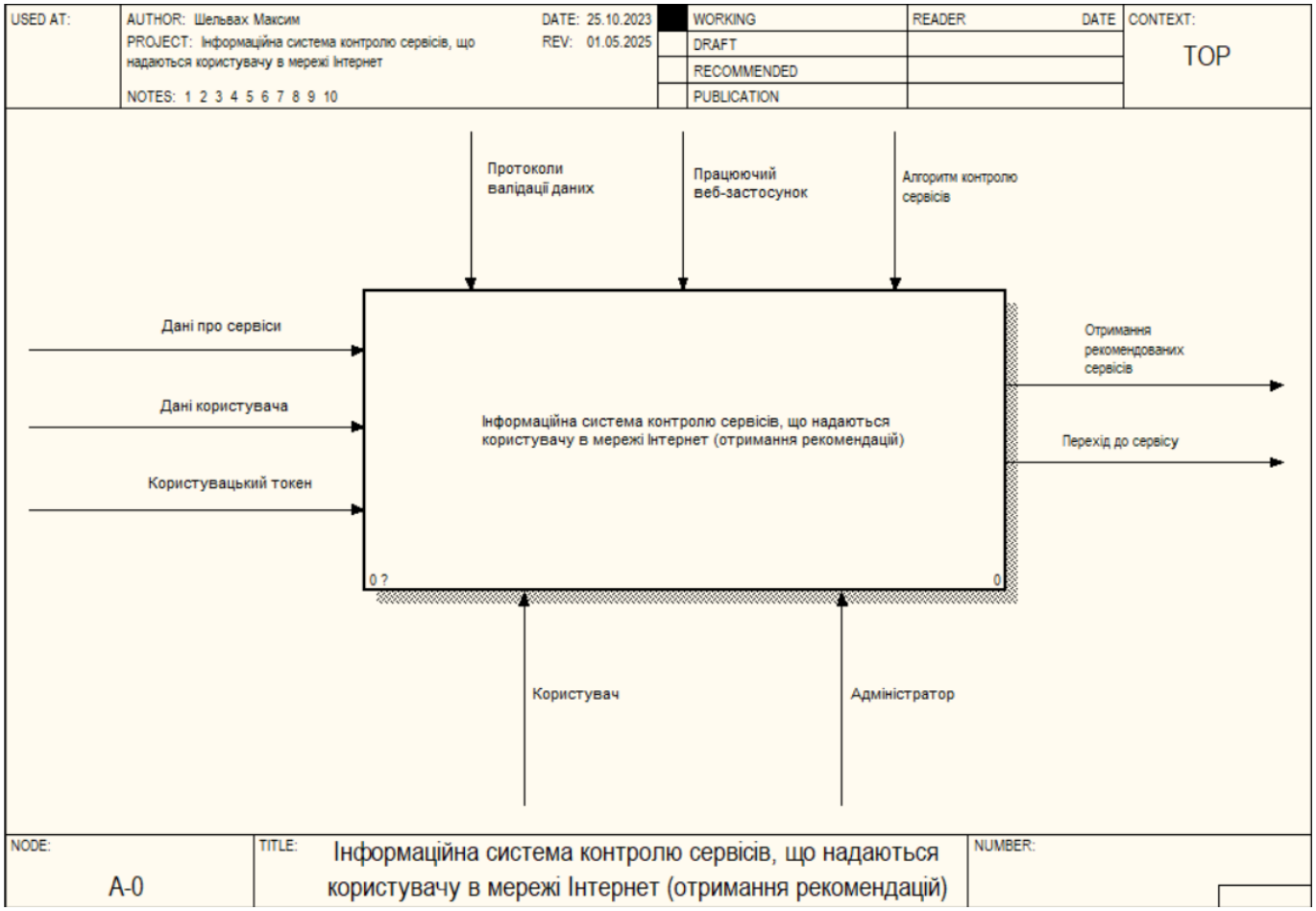

Рис. 2.3. Контекстна діаграма інформаційної системи.

Контекстна діаграма – це найвищий рівень абстракції в методології IDEF0. Вона показує основну функцію системи, охоплюючи весь її функціонал, але без заглиблення в деталі внутрішньої структури.

На діаграмі (Рис. 2.3) представлена загальна функція інформаційної системи – «Інформаційна система контролю сервісів, що надаються користувачу в мережі Інтернет (отримання рекомендацій)». Ця схема ілюструє зв’язки між основною функцією системи та зовнішніми елементами, такими як користувач, адміністратор, веб-застосунок, алгоритми контролю сервісів, протоколи валідації даних та інші.

На вході система отримує:

- дані про сервіси,

- дані користувача,
- користувацький токен.

У ролі керуючих впливів виступають:

- протоколи валідації даних,
- працюючий веб-застосунок,
- алгоритм контролю сервісів.

На виході система формує:

- отримання рекомендованих сервісів,
- перехід до конкретного сервісу.

Основними учасниками взаємодії є користувач і адміністратор. Завдяки цій моделі візуалізується логіка роботи системи на макрорівні, що допомагає уникнути помилок на етапі планування архітектури та спрощує подальше проєктування функціональних модулів.

Нова схема (Рис. 2.4) демонструє, як працює інформаційна система, розбиваючи її на три основні процеси:

- «Заповнення системи даними (сервіси, характеристики, рейтинги)»,
- «Вибір тегів для рекомендацій»,
- «Перегляд інформації та вибір сервісу».

На діаграмі також видно зв’язки між цими процесами та ключовими елементами контролю й механізмами, такими як «Протоколи валідації даних», «функціональний веб-застосунок» та «Адміністратор». Їхня основна мета – забезпечити актуальність даних у системі та контролювати її коректну роботу.

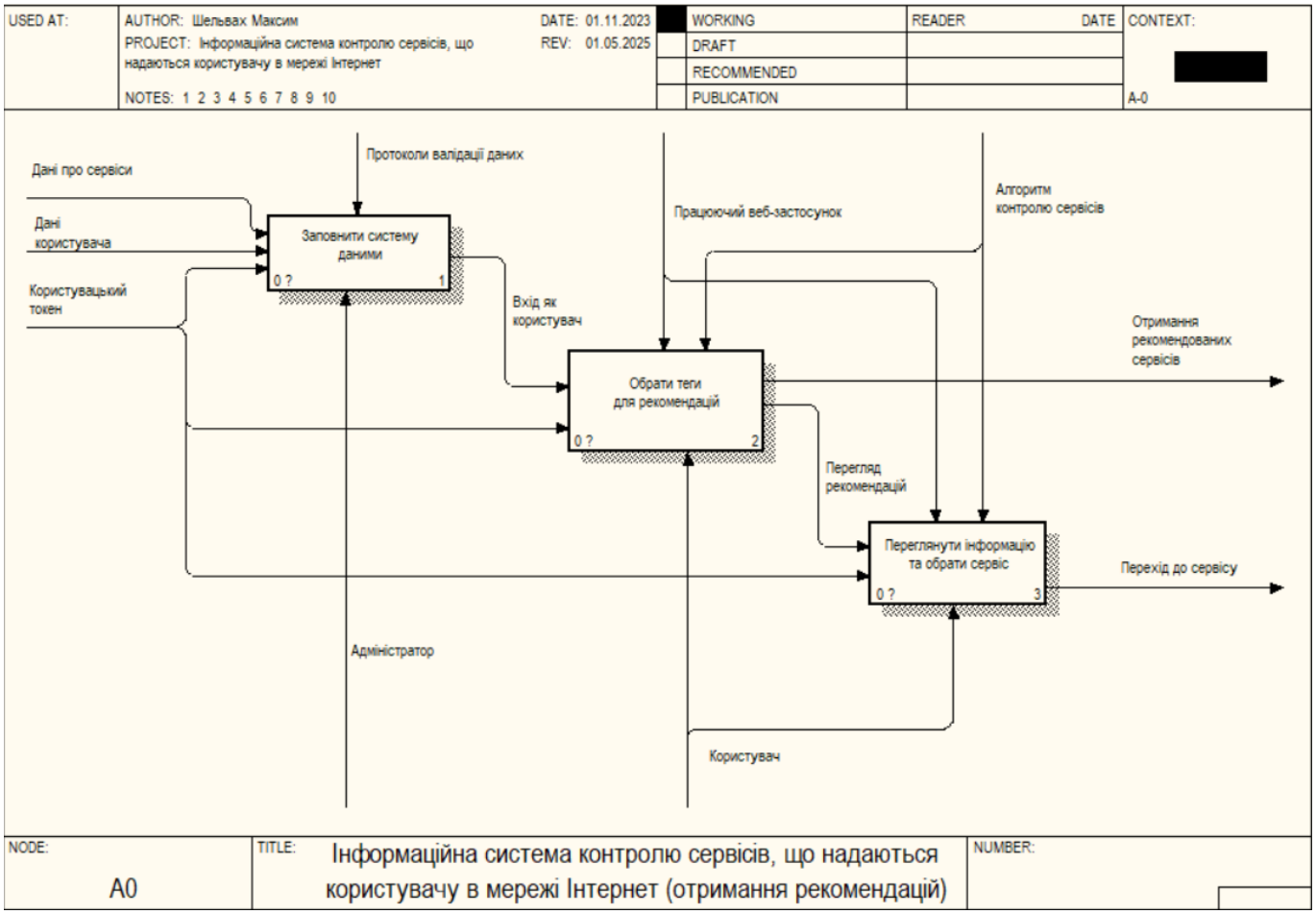

Рис. 2.4. Діаграма декомпозиції першого рівня.

Варто зазначити, що на діаграмі також представлені додаткові механізми: «Алгоритм контролю сервісів» та «Користувач».

Алгоритм контролю сервісів виконує повний цикл перевірки та формування рекомендацій: від аналізу тегів, введених користувачем, до підбору відповідних сервісів і перевірки їх актуальності та надійності. Назва цього механізму чітко відображає його призначення – забезпечити точний, безпечний та персоналізований результат для кінцевого користувача.

Користувач, у свою чергу, є основною стороною, яка взаємодіє із системою: він обирає теги, переглядає запропоновані сервіси та приймає рішення про перехід до конкретного ресурсу. Водночас, використовуючи систему, користувач «тестує» її на надійність, продуктивність та здатність коректно виконувати всі закладені алгоритми.

Таким чином, завдяки включенню цих механізмів, система забезпечує не лише автоматизовану обробку даних, а й постійний контроль якості роботи через фактичну взаємодію з кінцевим користувачем.

На діаграмі нижче (Рис. 2.5) показано, як виглядає процес, що називається «Заповнити систему даними».

Механізм під назвою «Адміністратор» постачає систему всією необхідною інформацією: даними про сервіси, інформацією про користувача та користувацькими токенами. Основне завдання адміністратора – внести ці дані в систему для подальшого використання.

Коли настає етап збереження даних, активуються «Протоколи валідації даних», які перевіряють правильність введеної інформації та запобігають дублюванню записів. Це допомагає підтримувати чистоту й актуальність даних у системі, а також уникати помилок, які можуть виникнути через повторний або некоректний ввід.

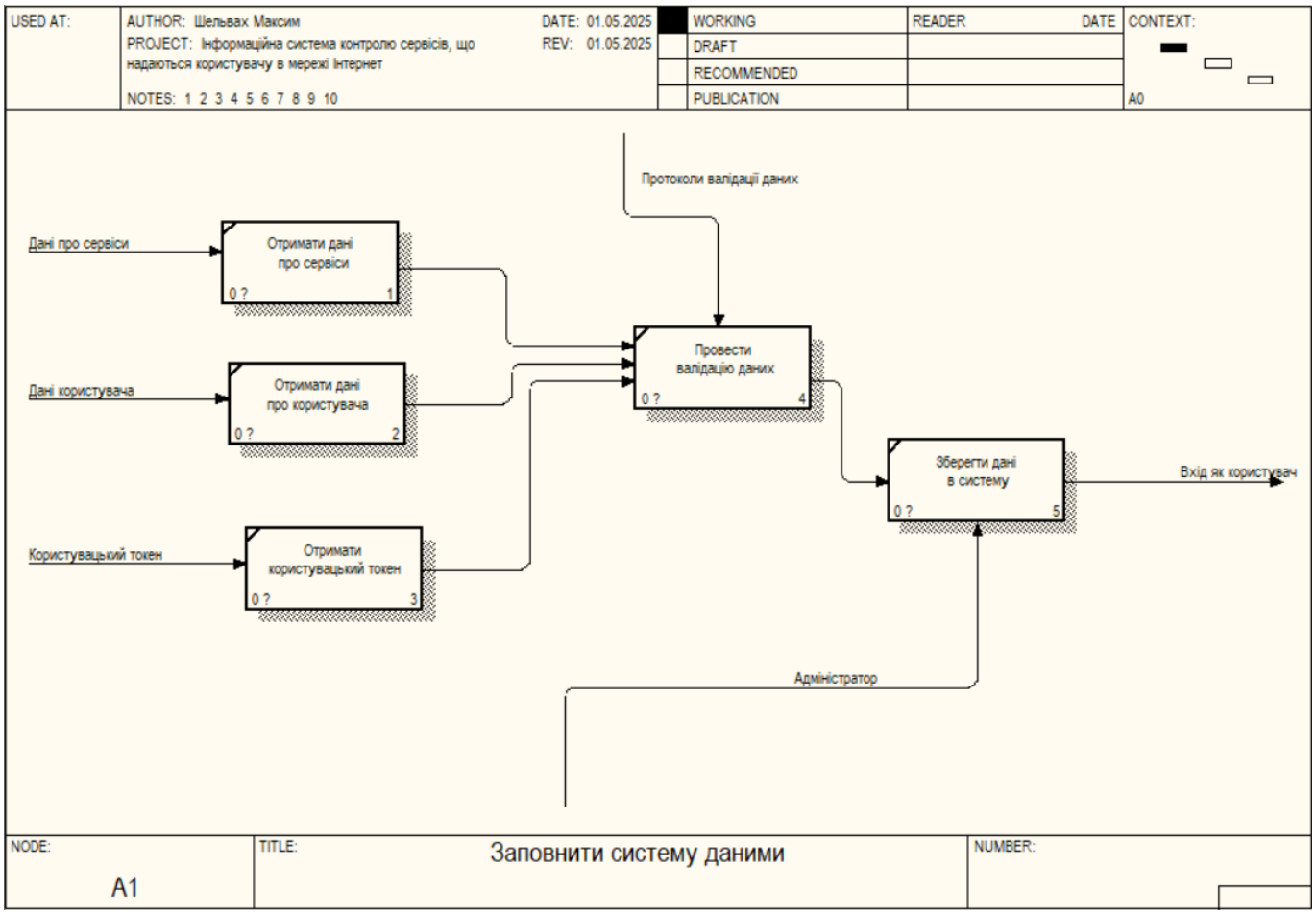

Рис. 2.5. Діаграма декомпозиції другого рівня процесу «Заповнити систему даними».

Далі розглянуто декомпозицію процесу, який називається «Обрати теги для рекомендацій» (Рис. 2.6). Ця схема наочно демонструє дії, які виконує звичайний користувач під час роботи з веб-застосунком.

Зокрема, система проходить через кілька кроків:

1. Отримати запит користувача – надсилається запит безпосередньо від користувача, або ж він визначає тематичні вподобання (теги), що йому до душі.
2. Шукати сервіси – система аналізує запит і підбирає відповідні сервіси з бази даних.
3. Відфільтрувати результати – на цьому етапі застосовуються алгоритми контролю сервісів для перевірки актуальності, надійності та відповідності вибраним тегам.
4. Відобразити результати – система формує остаточний список сервісів, який передається користувачеві.

Завдяки такій декомпозиції можна чітко простежити, як саме користувач взаємодіє з платформою, починаючи від формування запиту і закінчуючи отриманням персоналізованих рекомендацій та переглядом пропозицій.

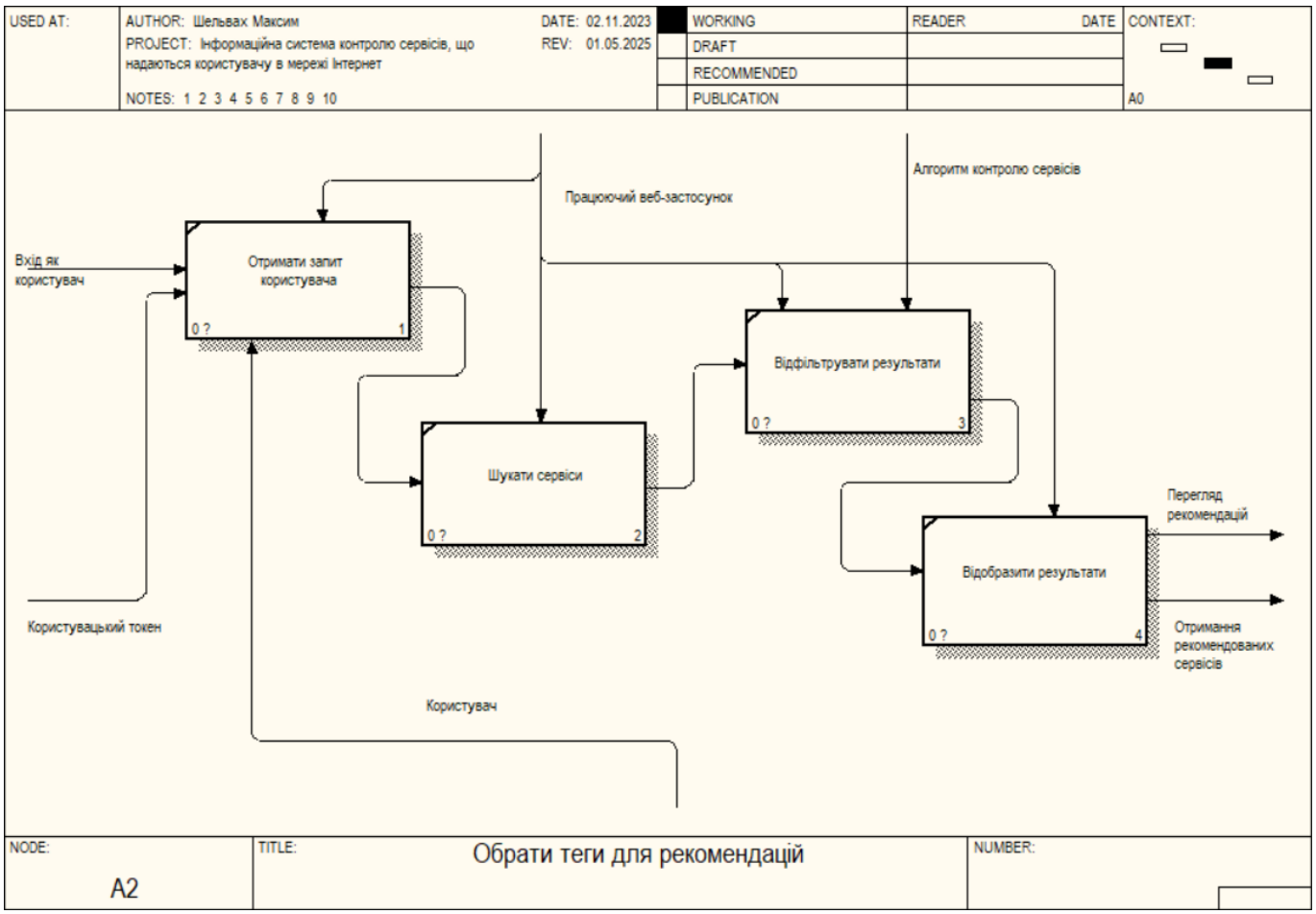

Рис. 2.6. Діаграма декомпозиції другого рівня процесу «Обрати теги для рекомендацій».

Наступним, і завершальним етапом у декомпозиції є «Переглянути інформацію та обрати сервіс» (Рис. 2.7). Ця схема показує основні дії користувача перед тим, як перейти на обраний сервіс.

Декомпозиція включає такі кроки:

- Отримати детальну інформацію про сервіс – користувач знайомиться з описом, рейтингами та умовами використання сервісу.
- Перевірити права користувача – система здійснює верифікацію доступу, перевіряючи, зокрема, чи наділений користувач правом на використання певної послуги.
- Виконати перехід до сервісу – це фінальний крок, який дозволяє користувачеві перейти на сайт або платформу обраного сервісу.

На цьому етапі вже працюють знайомі «Алгоритм контролю сервісів» та «Функціональний веб-застосунок», які забезпечують правильне відображення обраних сервісів і безпеку взаємодії. Процес «Перехід до сервісу» завершує логіку користувацької взаємодії, надаючи можливість фактично скористатися рекомендованим ресурсом.

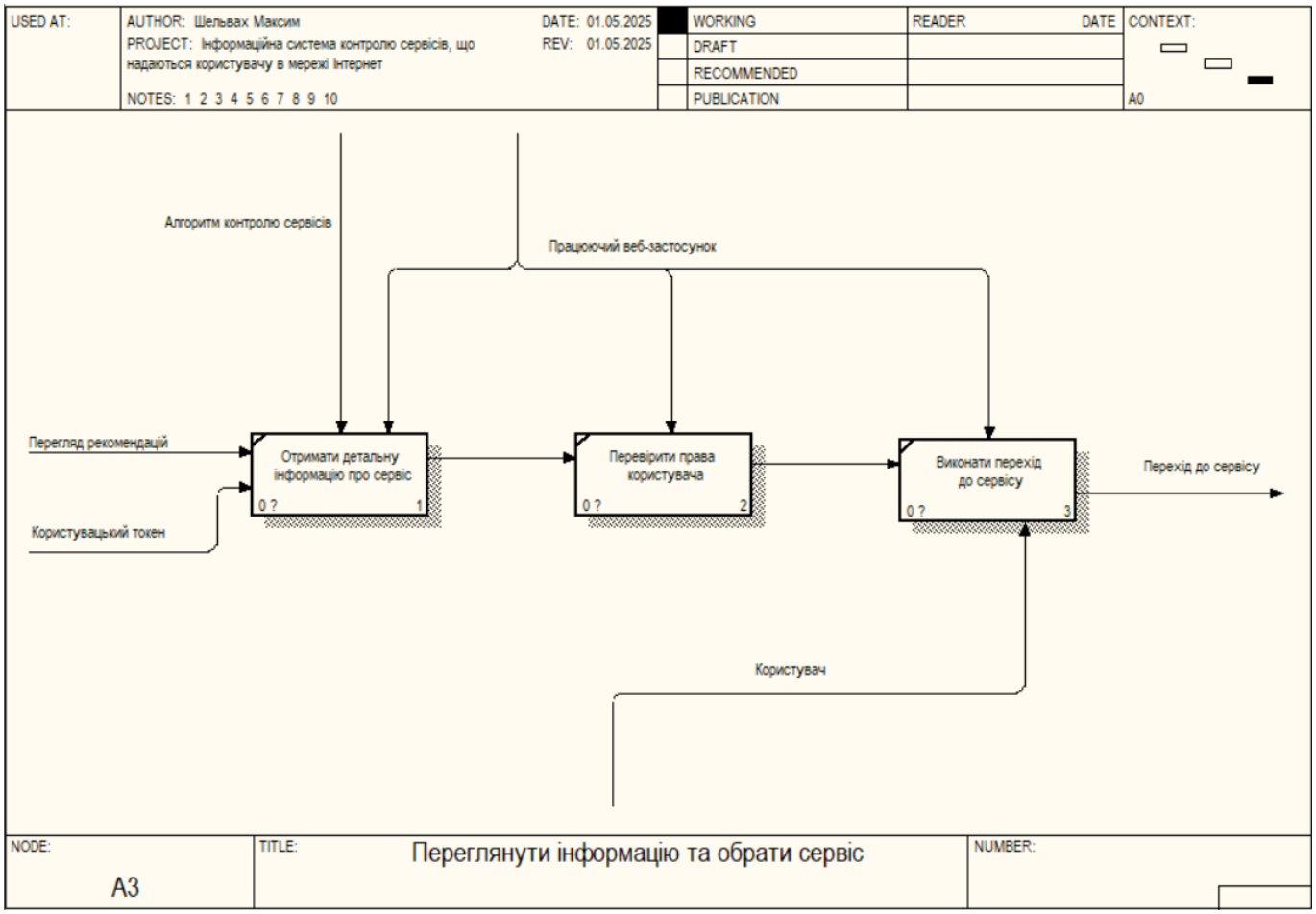

Рис. 2.7. Декомпозиція другого рівня процесу «Переглянути інформацію та обрати сервіс».

2.4 Побудова ієрархії процесів

Діаграма ієрархії процесів – це графічна модель, яка допомагає логічно організувати та відобразити структуру функціональних процесів у системі. Вона базується на методології "зверху вниз", де загальний процес поділяється на менші підпроцеси, які далі дробляться на конкретні функціональні частини. Цей спосіб

допомагає детально розібратися з роботою цілої системи, відслідковуючи взаємозв'язки між всіма її компонентами.

Що стосується діаграми ієрархії процесів (Рис. 2.8), то зверху зображено головний процес: «Інформаційна система контролю сервісів, що надаються користувачу в мережі Інтернет (отримання рекомендацій)».

Від нього вниз розходяться основні підпроцеси, які описані раніше під час декомпозиції:

- Заповнити систему даними,
- Обрати теги для рекомендацій,
- Переглянути інформацію та обрати сервіс.

Кожен із цих підпроцесів детально розглядається далі, наприклад:

- для заповнення системи даними – отримання інформації про сервіси, користувача, токен, валідація та збереження;
- для вибору тегів – отримання запиту, пошук, фільтрація й відбір результатів;
- для перегляду сервісу – отримання детальної інформації.

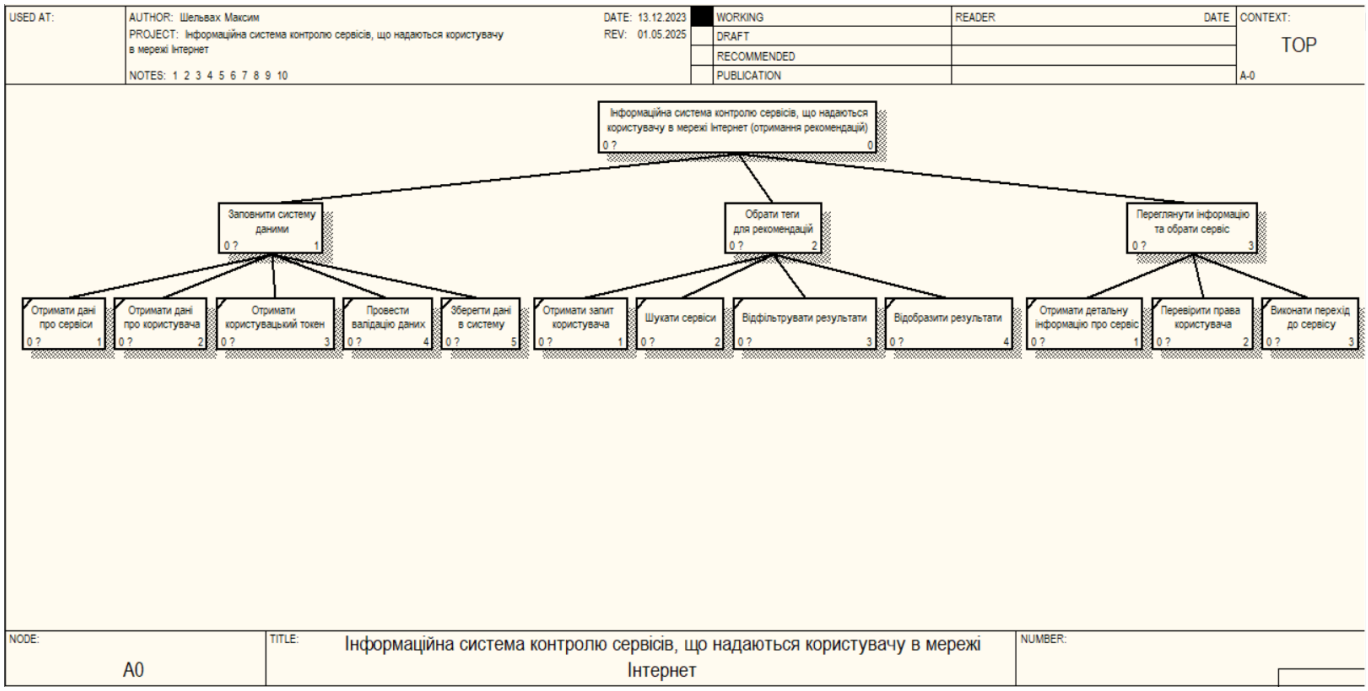

Рис. 2.8. Ієрархія процесів.

Висновок до розділу 2

У цьому розділі представлено детальний системний аналіз інформаційної системи керування сервісами, доступними онлайн. Ключову увагу приділено оцінці ергономічності взаємодії користувача з системою, як при ознайомленні та виборі сервісів, так і при керуванні власними налаштуваннями.

На початковому етапі побудовано дерево цілей, де генеральна мета – «Створення інформаційної системи керування сервісами» – розбита на ключові підцілі, які охоплюють персоналізацію взаємодії, надійність інформації та аналітику. З цієї структури було виділено чотири показники якості системи: Оперативність взаємодії з користувачем, Інтуїтивність інтерфейсу, Простота адміністрування та Можливість персоналізації.

Для визначення важливості кожного з критеріїв було застосовано метод аналізу ієрархій (AHP). Шляхом попарного порівняння встановлено, що найбільший вплив на якість системи має швидкість взаємодії з користувачем (46.5%), що дає змогу зосередити подальшу розробку на підвищенні продуктивності інтерфейсу та реакції системи.

Відповідно до створених матриць оцінювання альтернатив, найвищий загальний пріоритет отримала система рекомендацій сервісів, що демонструє найкращу відповідність вимогам до зручності, персоналізації та швидкодії. Таким чином, ця система є оптимальною платформою для реалізації інформаційної платформи, яка дозволить користувачам швидко знаходити актуальні сервіси відповідно до їхніх інтересів.

Додатково, у рамках методології IDEF0 було створено контекстну діаграму основного процесу функціонування системи, яка в подальшому була декомпонована до другого рівня з деталізацією основних підпроцесів. Це дозволило більш ретельно структурувати функціональність системи та створити чітку основу для її подальшого моделювання та впровадження.

РОЗДІЛ 3

Програмні засоби розв'язання задачі

3.1 Вибір та обґрунтування засобів розв'язання задач

Для інформаційної системи контролю сервісів, що надаються користувачу в мережі Інтернет, надзвичайно важливо вибрати технології, які забезпечать швидкий відгук під час фільтрації сервісів, надійне зберігання персональних даних користувачів, а також можливість масштабування при розширенні функціоналу чи інтеграції нових сервісів [32]. З точки зору розробників, важливо мати єдиний технологічний стек, щоб невелика команда могла зручно підтримувати проєкт.

У попередніх розділах було розглянуто аналоги на ринку та визначили ключові функції системи. Тепер, щоб реалізувати цей продукт, потрібно створити повноцінний веб-застосунок, який включатиме такі основні компоненти:

- бекенд (серверна частина),
- базу даних,
- фронтенд (користувацький інтерфейс),
- API-інструменти для інтеграції,
- програмне середовище для розробки та підтримки.

3.1.1. Бекенд технології системи

Для створення серверної частини інформаційної системи контролю сервісів, які надаються користувачу в мережі Інтернет, проаналізовано кілька мов програмування та фреймворків, здатних забезпечити необхідний рівень продуктивності, масштабованості та безпеки. У процесі бекенд-розробки було розглянуто такі технології: Node.js, Spring Boot (Java) та Django (Python). Кожна з цих технологій має свої унікальні особливості, недоліки та переваги, які були детально вивчені в контексті вимог до системи.

Node.js – це асинхронна подієво-орієнтована платформа, що використовує JavaScript на сервері [14]. Вона дає змогу опрацьовувати величезні масиви запитів миттєво, що робить її бездоганним рішенням для веб-програм, які вимагають

надшвидкого оброблення запитів та здатності витримувати значну кількість одночасних з’єднань. Серед основних переваг Node.js можна виділити:

- Високу продуктивність: підтримка асинхронних операцій дозволяє обробляти значний обсяг запитів без блокувань, забезпечуючи низьку затримку;
- Єдиний стек для фронт- та бекенду: використання JavaScript як на клієнтській, так і на серверній частинах спрощує розробку та робить її більш узгодженою;
- Багатий екосистема npm: доступ до великої кількості бібліотек та інструментів, що значно полегшує процес розробки.

Spring Boot (Java) – один із найпопулярніших фреймворків для серверної розробки на Java, що пропонує готові рішення для побудови мікросервісних архітектур. Ось його ключові переваги:

- Масштабованість для обробки великих обсягів даних та високих навантажень;
- Безпека забезпечується інтегрованими механізмами аутентифікації та авторизації;
- Надійність, підкріплена зрілою екосистемою та широким використанням у великих компаніях.

Django (Python) – це високорівневий фреймворк, який дозволяє легко та швидко створювати веб-додатки з добре продуманою архітектурою. Ось деякі з його ключових переваг:

- Швидка розробка завдяки вбудованим компонентам.
- Підвищена безпека із захистом від поширених атак.
- Масштабованість, хоча вона вимагає більше серверних ресурсів порівняно з Node.js.

Після порівняння особливостей кожної технології для розробки бекенду нашої інформаційної системи обрано Node.js та фреймворк Express [8]. Його невимушеність, здатність до адаптивного розширення та дивовижна працездатність

при обробці великої кількості звернень, разом із уніфікованим стеком як для передньої, так і для задньої частин, суттєво полегшують процес творення та допомагають нам сконструювати бекенд, який задовольнить усі наші потреби.

3.1.2 База даних

Під час визначення бази даних для інформаційної системи управління сервісами, доступними онлайн, критично важливо брати до уваги низку визначальних аспектів. До них належать: адаптивність структури даних, оперативність обробки запитів та перспективи масштабування. Враховуючи необхідність системи оперувати великим масивом даних щодо сервісів, користувачів, тегів та порад, було проаналізовано три базові варіанти: MongoDB, PostgreSQL та MySQL.

MongoDB – це документно-орієнтована база даних, яка використовує гнучку модель зберігання даних у форматі JSON-подібних документів. Вона здобула популярність завдяки своїй масштабованості, гнучкості та легкості інтеграції з іншими технологіями. Ось кілька основних характеристик:

- Продуктивність: асинхронне виконання запитів дозволяє працювати в реальному часі з великими обсягами даних;
- Гнучкість: можливість змінювати структуру даних без необхідності в міграціях;
- Індексація: підтримка індексування на багатьох полях, що пришвидшує запити та покращує користувацький досвід.

PostgreSQL – це об'єктно-реляційна система керування базами даних (СКБД), яка має підтримку SQL та демонструє відмінну продуктивність при роботі з великими обсягами даних. Ось кілька основних характеристик:

- Реляційна модель: використовує класичні таблиці з зв’язками між ними;
- Складні запити: підтримує агрегати, вкладені запити та складну логіку;
- Менш гнучка структура: зміни в схемі таблиць вимагають міграцій.

MySQL – це відома реляційна система керування базами даних (СКБД), яку часто застосовують для великомасштабних веб-застосунків [16]. Основні характеристики:

- Продуктивність: добре підходить для базових операцій читання/запису при помірному навантаженні;
- Масштабування: потребує додаткових налаштувань для обробки високих навантажень;
- Складні запити: має обмежену підтримку в порівнянні зі складними SQL-операціями.

Після ретельного аналізу баз даних було обрано MySQL, оскільки це найкраще рішення для нашої системи. MySQL пропонує надійність, достатню продуктивність для обробки запитів користувачів, легкість інтеграції з Node.js та можливість масштабування, якщо функціонал розшириться. Для системи, де головними пріоритетами є стабільність, швидкість запитів і робота з відносно стабільною структурою даних, MySQL стає оптимальним вибором, що дозволяє без зусиль впроваджувати та підтримувати систему.

3.1.3 Фронтенд технології

Перед тим, як обрати остаточний фронтенд-стек, було проведено детальне порівняння двох основних підходів: класичного, що включає HTML5, CSS3 та чистий JavaScript (Vanilla JS), і сучасних JavaScript-фреймворків, таких як React, Angular та Vue.

Класичний підхід передбачає створення інтерфейсу «з нуля», де HTML5 відповідає за структуру та зміст веб-застосунку, CSS3 – за стилізацію, а JavaScript – за логіку, інтерактивність і обробку подій [15]. Ось кілька його ключових особливостей:

- Простота: легко почати, адже базових знань HTML, CSS та JS цілком достатньо;
- Гнучкість: стандартний JavaScript дозволяє безпосередньо взаємодіяти з HTML і CSS;
- Складність при масштабуванні: з ростом проєкту зростає й обсяг коду, з'являється дублювання логіки, а підтримка стає дедалі складнішою.

Сучасні фреймворки, такі як React , Angular та Vue, хоч і мають свої унікальні синтаксиси та історії розвитку, виконують подібні завдання і пропонують спільний набір переваг [24]:

- Компонентна модель: інтерфейс складається з окремих, багаторазово використовуваних компонентів, що робить масштабування простішим;
- Віртуальний DOM: зменшує кількість операцій з реальним DOM, що підвищує продуктивність;
- Декларативний підхід: фреймворк самостійно визначає, які мінімальні зміни потрібно внести в DOM, щоб синхронізувати його з поточним станом;
- Керування станом та маршрутизація (Routing): забезпечує навігацію без перезавантаження сторінок і централізоване управління даними [25].

Зважаючи на вимоги до функціональності, масштабованості та подальшого розвитку інформаційної системи контролю сервісів, найкращим вибором виявився React. Використання React дозволяє швидко розробляти повторно використовувані компоненти, централізовано управляти станами та отримувати доступ до розвиненої екосистеми UI-рішень. Це значно скорочує час розробки системи, знижує довгострокові витрати на підтримку і мінімізує технічний борг у порівнянні з використанням чистого HTML, CSS та JavaScript, де багато процесів довелося б реалізовувати вручну [5].

3.1.4 API-інструменти

У процесі розробки REST API для інформаційної системи, виникає необхідність у інструменті, який дає змогу тестувати запити, верифікувати відповіді від сервера та генерувати звіти про це [27]. Серед найвідоміших рішень, варто звернути увагу на Insomnia та Postman.

Insomnia – це потужний інструмент для тестування REST API, який пропонує зручний інтерфейс для створення, надсилання запитів і перевірки відповідей. Він підтримує роботу з багатьма протоколами, такими як REST, GraphQL та gRPC. Ось кілька основних особливостей Insomnia:

- Підтримка протоколів: Окрім REST, Insomnia дозволяє працювати з GraphQL та іншими популярними технологіями;
- Можливості тестування: Надсилання запитів, перевірка відповідей, управління змінними середовища;
- Імпорт/експорт: Легкий обмін колекціями запитів між членами команди, що пришвидшує роботу над проєктом.

Postman – це універсальний інструмент, який не потребує вбудованої документації в коді і дозволяє вам самостійно створювати колекції запитів, тестувати автоматизацію та структуру відповідей у форматі JSON [22]. Він ідеально підходить для тестування API. Ось кілька основних переваг Postman:

- Легкість у використанні: Postman готовий до роботи відразу після встановлення, без жодних додаткових налаштувань;
- Візуалізація: Пропонує зручний графічний інтерфейс для перегляду запитів і відповідей, включаючи формат JSON;
- Тестування: Дозволяє надсилати запити, аналізувати відповіді та досліджувати функціональність API.

Для нашої системи Postman – найкращий вибір. Його гнучкість, можливість тестування будь-яких запитів без попередньої інтеграції, простота використання та легкість у роботі роблять його ідеальним інструментом для API-тестування на етапі розробки. Insomnia, хоча й є сучасною альтернативою, у нашому випадку може залишитися інструментом на перспективу – для роботи над специфічними запитами або в командній взаємодії.

3.1.5 Програмне середовище розробки

Вибір програмного середовища для реалізації проєкту – це важливий і суттєвий етап, адже правильно підібраний інструмент може значно скоротити час розробки, а також забезпечити зручність і ефективність роботи з кодом. Для нашої інформаційної системи було проаналізовано два середовища: Visual Studio Code та WebStorm.

Visual Studio Code – один з лідерів серед редакторів коду, що заслужено здобув популярність, вирізняється значною функціональністю. Надає все необхідне для зручного кодування JavaScript, HTML, CSS та підтримки багатьох інших мов програмування [40]. Ось кілька його основних особливостей:

- Інтеграція з Git, що суттєво полегшує процес керування версіями коду та фіксації змін.;
- Висока швидкість роботи, завдяки якій він ефективно справляється як з малими, так і з великими проєктами;
- Наявність величезного вибору плагінів і розширень, які допомагають пришвидшити написання коду та налаштувати середовище під потреби проєкту.

WebStorm – це середовище розробки, яке спеціалізується на JavaScript та фронтенд-технологіях. Воно забезпечує глибоку інтеграцію з Node.js, React, Angular та іншими популярними фреймворками. Ось кілька основних переваг:

- Розширене автодоповнення та рефакторинг: WebStorm пропонує інтелектуальне автодоповнення, перевірку коду в реальному часі та потужні можливості рефакторингу.
- Інтеграція з інструментами розробки: підтримка npm, Yarn, Webpack та інших інструментів для управління залежностями, запуску скриптів і побудови проєкту.

Враховуючи простоту, блискавичну продуктивність і гнучкі налаштування, було обрано Visual Studio Code. Це комфортне та функціональне середовище повністю відповідає вимогам розробки як клієнтської, так і серверної частин системи, пропонуючи всі необхідні засоби для роботи з обраним набором технологій.

3.2 Проектування структури даних

Проектування структури даних – це надзвичайно важливий етап у створенні інформаційної системи. Саме на цьому етапі окреслюються ключові сутності, їхні характеристики та взаємозв’язки. У процесі розробки інформаційної системи для контролю сервісів, які надаються користувачам в Інтернеті, була створена логічна

модель даних. Вона забезпечує ефективне зберігання та обробку інформації про сервіси, користувачів, теги та інше.

ER-діаграма (Рис. 3.1) демонструє основні об’єкти та їхні взаємозв’язки [7], що лежать в основі розроблюваної системи. У нашій моделі виділено кілька ключових сутностей: Users (користувачі), Services (сервіси), Tags (теги), а також допоміжні таблиці, які забезпечують збереження переваг, збережених сервісів, тегів користувачів та тегів сервісів.

Сутність Users містить базову інформацію про користувачів системи: унікальний ідентифікатор, email, пароль, ім’я, роль, дату створення та оновлення облікового запису. Кожен користувач може мати свої теги (UserTags) і переваги (Preferences), а також зберігати улюблені сервіси (SavedServices).

Сутність Services зберігає детальну інформацію про сервіси, а саме: унікальний ідентифікатор, назву, опис, URL, статус верифікації, дату створення, оновлення та ID користувача, який додав сервіс. Сервіси пов’язані з тегами через таблицю ServiceTags, що дозволяє гнучко класифікувати та фільтрувати їх.

Сутність Tags описує категорії та теги, які можна застосовувати як до сервісів, так і до переваг користувачів. Це дає змогу налаштувати систему рекомендацій та персоналізації.

Сутність SavedServices дозволяє зберігати сервіси, що зацікавили користувача, разом із пов’язаним тегом. Сутність Preferences зберігає індивідуальні вподобання користувача щодо тегів, що допомагає будувати рекомендації. UserTags відображає персональні теги користувача, що дозволяє ще точніше налаштувати роботу системи під конкретного користувача.

Завдяки ретельно продуманій структурі даних система здатна обробляти запити користувачів, пропонувати релевантні сервіси, зберігати уподобання та формувати персоналізовані рекомендації, що суттєво підвищує ефективність її роботи та задоволення користувачів.

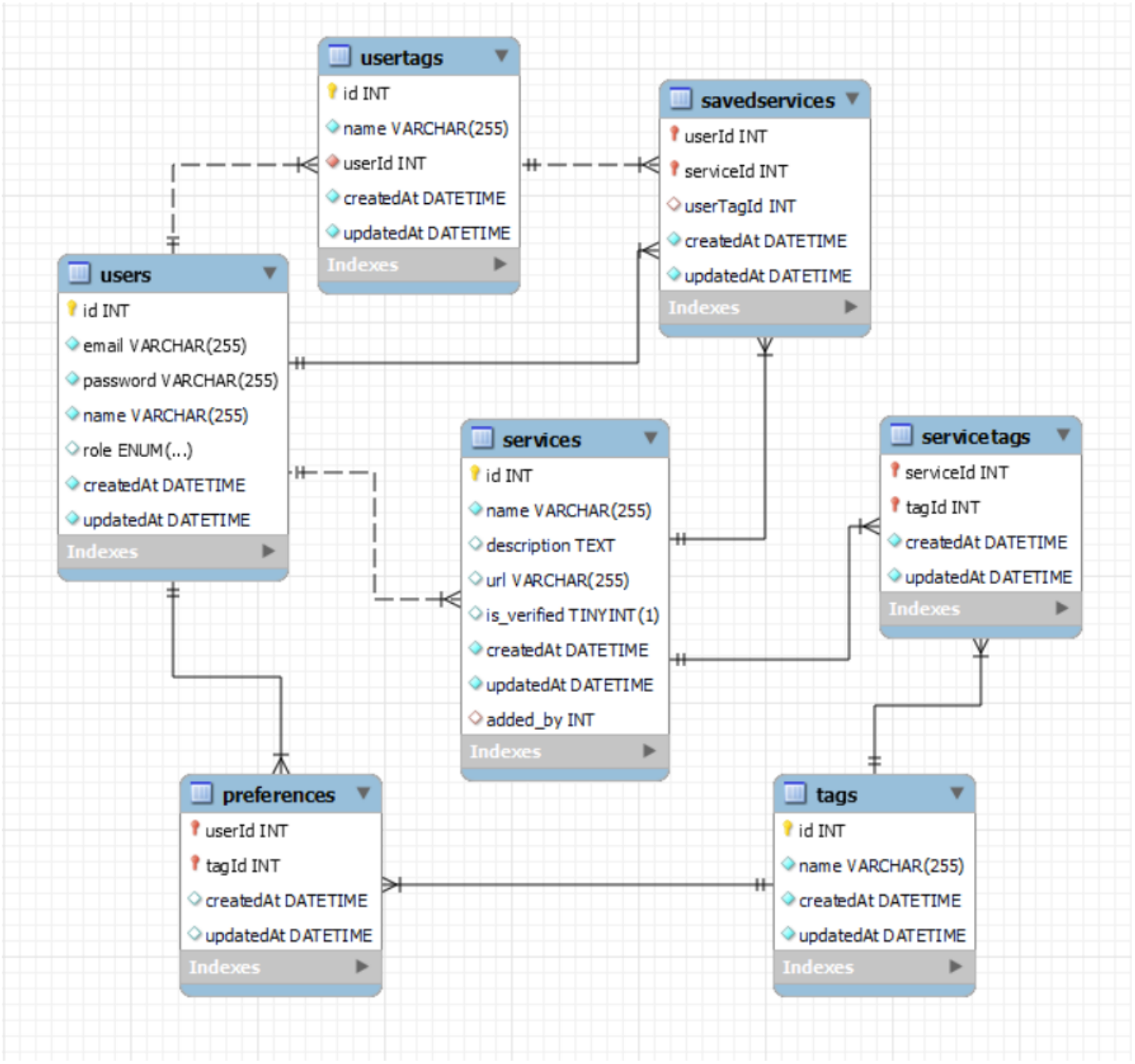

Рис. 3.1. ER-діаграма інформаційної системи контролю сервісів, що надаються користувачу в мережі Інтернет.

3.3 Технічні характеристики обраних програмних засобів

Інформаційна система контролю сервісів, які надаються користувачу в мережі Інтернет, є складним і ретельно спроєктованим рішенням, що базується на потужних інструментах та сучасних технологіях розробки. Вона об'єднує передові рішення для забезпечення високої продуктивності, масштабованості та зручності у використанні.

Основою серверної частини проекту є Node.js – популярна платформа для створення масштабованих веб-додатків, яка використовує JavaScript для обробки запитів. Завдяки своїй асинхронній природі, Node.js дозволяє обробляти велику кількість одночасних запитів, що є ключовою перевагою для інтерактивних систем, таких як наша. Бекенд працює за допомогою фреймворку Express.js, який забезпечує просте налаштування маршрутизації, обробки запитів та інтеграцію з базою даних MySQL. Такий вибір дозволяє створити гнучку, швидку та зручну у підтримці серверну архітектуру, яка відповідає всім вимогам проекту.

Коли мова йде про базу даних, обрано MySQL. Це реляційна база даних, яка дозволяє ефективно працювати зі структурованими даними, забезпечуючи надійність, швидкість виконання запитів і підтримку складних зв’язків між таблицями. MySQL ідеально підходить для нашої системи, оскільки вона дозволяє зберігати та обробляти інформацію про користувачів, сервіси, теги, вподобання та їх взаємозв’язки.

Для клієнтського інтерфейсу використовуватиметься фреймворк React. Його компонентний підхід значно спрощує створення динамічних інтерфейсів, дозволяючи повторно використовувати компоненти та ефективно управляти станами програми. Для полегшення створення клієнтської архітектури додатку були використані додаткові бібліотеки, такі як Vite, що значно спрощує створення фронтенду [30].

Система працюватиме локально, без потреби у зовнішньому хостингу, що дає розробникам можливість контролювати всі процеси без зайвих налаштувань серверів чи веб-хостингу. Це також робить тестування та демонстрацію роботи системи на локальних машинах значно простішими.

Для тестування REST API обрано Postman. Цей інструмент дозволяє зручно формувати запити, перевіряти відповіді сервера та швидко виявляти й виправляти помилки.

Основним середовищем розробки є Visual Studio Code, яке підтримує всі зазначені технології. Завдяки розширенням, воно забезпечує швидку та ефективну роботу з кодом, підсвічування синтаксису, інтеграцію з Git та роботу з базами даних.

Висновок до розділу 3

У цьому розділі було розглянуто обрання та обґрунтування технологій для створення інформаційної системи контролю сервісів, які надаються користувачам в Інтернеті. Добре підібраний стек технологій забезпечує стабільну та злагоджену роботу всіх компонентів системи.

Серверна частина реалізована на Node.js у поєднанні з Express.js, що дозволяє швидко створювати RESTful API та обробляти запити клієнтів з мінімальними затримками. React разом з Vite забезпечують високу продуктивність клієнтської частини, а також пришвидшує процес розробки завдяки підтримці сучасних модулів і миттєвому оновленню інтерфейсу.

Використання MySQL як системи управління базами даних дозволило ефективно структурувати та зберігати інформацію про сервіси, користувачів, теги тощо. Завдяки ORM-інструментам інтеграція між базою даних та логікою додатку стала значно простішою. Для перевірки роботи API ми використовували Postman, що допомогло забезпечити надійність передачі даних між фронтендом і бекендом.

Обраним середовищем розробки став Visual Studio Code, що надає безліч можливостей для роботи з кодом.

РОЗДІЛ 4

Практична реалізація

4.1 Опис створеного програмного засобу

Реалізована інформаційна система контролю сервісів, що надаються користувачу в мережі Інтернет, є відтворенням основного функціоналу сучасного веб-застосунку для управління та вибору онлайн-сервісів. Користувачам надається можливість переглядати доступні сервіси, ознайомлюватися з їх описами, за допомогою фільтрації обирати сервіси за тегами, додавати сервіси до обраного та, у кінцевому результаті, – зберігати їх для подальшого використання.

Веб-застосунок створено у вигляді «вітрини», де на головній сторінці представлена загальна інформація про систему та переваги користування системою. Окрема сторінка каталогу сервісів дозволяє користувачам шукати та переглядати сервіси за різними характеристиками, такими як теги, щоб звузити вибір до бажаних категорій.

Усі дані завантажуються динамічно через запити до бекенд-сервера, який реалізовано на базі Node.js та Express.js. Серверна частина виконує основну бізнес-логіку: обробку запитів, перевірку даних, управління збереженими сервісами, а також авторизацію та реєстрацію користувачів.

Усі дані завантажуються динамічно через запити до бекенд-серверу, реалізованого на базі Node.js та Express.js. Серверна частина виконує основну бізнес-логіку: обробку запитів, перевірку даних, управління збереженими сервісами, авторизацію та реєстрацію користувачів.

Уся система базується на двох основних компонентах, а саме:

- Фронтенд – реалізований за допомогою React із застосуванням Vite для спрощення створення архітектури додатку [39];
- Бекенд – реалізований на Node.js з використанням Express.js та бази даних MySQL.

Детальніший опис функціоналу кожного елемента системи буде представлений у наступних підрозділах.

4.1.1 Функціонал бекенд частини (Node.js та Express.js)

Бекенд-частина інформаційної системи контролю сервісів, які надаються користувачеві в мережі Інтернет, створена з використанням Node.js у поєднанні з серверним фреймворком Express. На даному етапі розробки основне завдання бекенду полягає в обробці запитів, пов’язаних із реєстрацією користувачів, авторизацією, управлінням сервісами та тегами, а також у збереженні обраних сервісів користувачів.

У системі реалізована базова REST API архітектура, яка включає:

- обробку запитів на створення облікових записів користувачів;
- авторизацію користувачів за допомогою електронної пошти та паролю;
- роботу з сервісами (додавання, оновлення, видалення, перегляд);
- управління тегами та вподобаннями користувачів.

Архітектура бекенду побудована на принципах REST API, де кожен маршрут відповідає певній операції. Сервер отримує запити від фронтенду, обробляє їх, виконує необхідні дії з базою даних MySQL і повертає результат у форматі JSON.

Щоб спростити підтримку та масштабування проекту, код розділено на окремі модулі. Такий підхід допомагає уникнути безладу в коді, адже кожен модуль виконує свою конкретну функцію та має чітке призначення. Розробка проекту проходила в середовищі Visual Studio Code, а основною мовою програмування була JavaScript на платформі Node.js. Далі буде представлено опис основних папок-модулів бекенд-частини системи (Рис. 4.1).

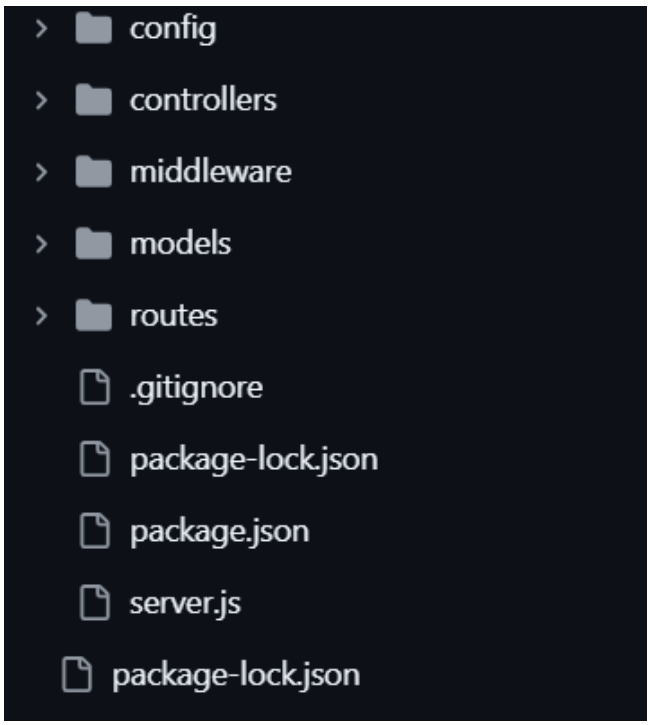

Рис. 4.1. Серверна структура системи.

Давайте розпочнемо з найважливішої частини бекенд-частини – моделі користувача. Її головна мета полягає в створенні схеми для бази даних, яка описує користувача. Модель визначає, які поля міститиме кожен запис користувача, а також встановлює правила для цих полів. Завдяки цій моделі можна ефективно працювати з колекцією користувачів у базі даних, використовуючи бібліотеку Sequelize, яка забезпечує з'єднання між MySQL та Node.js [31]. Модель також визначає об'єкти та їх структуру, що відповідає даним у базі, дозволяючи нам створювати, оновлювати та взаємодіяти з користувачами в системі.

Модель користувача, яку було описано в нашій системі, складається з п'ятьох основних полів:

- id – ID користувача, яке є первинним ключем таблиці, створюється автоматично.
- name – ім'я користувача, яке є обов'язковим для заповнення та має тип String (рядок). Це поле містить інформацію про ім'я користувача, і його заповнення є необхідним для створення облікового запису.

- email – електронна пошта, також обов’язкова для заповнення та має тип String. Було приділено особливу увагу валідації цього поля, щоб забезпечити правильність введених поштових адрес. Це важливий атрибут, оскільки він використовується для авторизації користувача в системі.
- password – пароль користувача, який повинен мати мінімум 6 символів, є обов’язковим для заповнення та має тип String. Щоб підвищити рівень безпеки, пароль хешується перед збереженням у базі даних, що захищає конфіденційну інформацію користувача від несанкціонованого доступу.
- role – роль користувача, автоматично ставиться роль user. Це поле потрібно системі для розуміння повноважень авторизованого користувача.

У нашій системі також реалізовано асоціації між моделями за допомогою Sequelize. Наприклад, кожен користувач може мати кілька тегів через асоціацію з моделлю Tag. Крім того, користувач може зберігати та обробляти інформацію про обрані послуги через зв’язок з моделлю Service.

Ці зв’язки забезпечують збереження структурованої інформації та полегшують управління даними в системі.

```
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define('User', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    email: { type: DataTypes.STRING, unique: true, allowNull: false },
    password: { type: DataTypes.STRING, allowNull: false },
    name: { type: DataTypes.STRING, allowNull: false },
    role: { type: DataTypes.ENUM('user', 'admin'), defaultValue: 'user' }
  });

  User.associate = (models) => {
    User.belongsToMany(models.Tag, {
      through: models.Preference,
      foreignKey: 'userId',
      as: 'preferences'
    });

    User.belongsToMany(models.Service, {
      through: models.SavedService,
      foreignKey: 'userId',
      as: 'savedServices'
    });
  };

  return User;
};
```

Рис. 4.2. Модель таблиці user.

Файл authController.js у нашому проекті виконує ключову бізнес-логіку для обробки реєстрації та авторизації користувачів. Цей файл є частиною контролерів, які відповідають за обробку запитів, що надходять на відповідні маршрути користувачів. Він реалізує такі функції:

- registerUser – для реєстрації нового користувача.
- loginUser – для входу користувача до системи.
- getMe – для отримання даних поточного користувача.

Функція registerUser:

Ця функція обробляє запит на реєстрацію нового користувача і працює наступним чином:

- Зчитуються дані з запиту, які містять name, email та password.
- Перевіряється, чи є користувач із вказаною електронною поштою.
- Якщо користувач існує, повертається помилка. Якщо ні, пароль хешується за допомогою бібліотеки bcryptjs для безпечного зберігання.
- Створюється новий документ користувача в базі даних використовуючи модель User.
- У відповідь надсилається повідомлення про успішну реєстрацію користувача.

Функція loginUser:

У функції входу перевірка відбувається на основі електронної пошти та зашифрованого пароля. Алгоритм роботи:

- Вводяться email та password.
- Перевіряється, чи є користувач із вказаною електронною поштою.
- Порівнюється введений пароль з тим, що зберігається у базі за допомогою bcryptjs.
- Якщо все вірно, генерується JWT токен, який містить ідентифікатор користувача та його роль, та повертається на клієнтську сторону.

Функція getMe дозволяє отримати дані поточного користувача, якщо він увійшов до системи, шляхом пошуку в базі даних за ID користувача.

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const { User } = require('../models');

exports.register = async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = await User.create({ name, email, password: hashedPassword });
    res.status(201).json({ message: 'User registered' });
  } catch (err) {
    res.status(500).json({ error: 'Registration failed' });
  }
};

exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ where: { email } });
    if (!user || !(await bcrypt.compare(password, user.password))) {
      return res.status(401).json({ error: 'Invalid credentials' });
    }
    const token = jwt.sign({ id: user.id, role: user.role }, process.env.JWT_SECRET);
    res.json({ token });
  } catch (err) {
    res.status(500).json({ error: 'Login failed' });
  }
};

exports.getMe = async (req, res) => {
  const user = await User.findByPk(req.user.id);
  res.json({ id: user.id, name: user.name,email: user.email, role: user.role });
};
```

Рис. 4.3. Контролер authController.js.

Файл authMiddleware.js у нашому проекті відповідає за перевірку авторизації користувача перед тим, як він зможе отримати доступ до захищених маршрутів. Цей файл перевіряє, чи є JWT-токен у заголовку запиту, і чи він коректний.

Ось основні етапи роботи middleware:

- Перевірка наявності токена: спочатку зчитується заголовок Authorization з запиту. Якщо заголовка немає або він не починається з Bearer <token>, то система

повертає помилку з HTTP статусом 401 і повідомленням про необхідність авторизації.

- Розбір токена: якщо заголовок містить токен, він розділяється на дві частини за допомогою методу split(' '). Одна частина повинна бути Bearer, а інша – сам токен.
- Верифікація токена: за допомогою бібліотеки jsonwebtoken токен перевіряється з використанням секретного ключа з process.env.JWT_SECRET. Якщо токен недійсний або термін його дії закінчився, повертається помилка з відповідним повідомленням.
- Передача контролю: якщо токен успішно верифіковано, корисне навантаження токена (payload), яке містить ідентифікатор користувача, передається в об'єкт req.user. Після цього викликається функція next(), що дозволяє передати управління наступному middleware або обробнику маршруту.

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Authorization token missing' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid or expired token' });
  }
};
```

Рис. 4.4. Middleware/authMiddleware.js елемент системи.

Файл roleMiddleware.js у нашому проекті відповідає за перевірку прав доступу користувача на основі його ролі перед тим, як він зможе отримати доступ до певних

маршрутів. Цей middleware використовує роль користувача, щоб визначити, чи має він необхідні привілеї для виконання конкретних дій.

Ось основні етапи роботи цього middleware:

- Перевірка наявності користувача: Спочатку перевіряємо, чи є в запиті об'єкт користувача (req.user). Якщо користувача немає, система повертає помилку з кодом 401 і повідомленням "Unauthorized".
- Перевірка ролі користувача: Далі перевіряється роль користувача. Якщо роль не відповідає тій, що вказана в параметрі requiredRole, повертається помилка з кодом 403 і повідомленням "Forbidden: insufficient rights".
- Продовження обробки: Якщо обидві перевірки пройдені успішно, контроль передається до наступного middleware або обробника маршруту за допомогою функції next().

```
module.exports = (requiredRole) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ message: 'Unauthorized' });
    }
    if (req.user.role !== requiredRole) {
      return res.status(403).json({ message: 'Forbidden: insufficient rights' });
    }
    next();
  };
};
```

Рис. 4.5. Middleware/roleMiddleware.js елемент системи.

Файл auth.js у нашому проекті відповідає за маршрутизацію запитів, які стосуються реєстрації, авторизації користувачів та отримання інформації про поточного користувача. У коді описані 3 основних API-шляхи для авторизації, що в подальшому використовуватимуться фронтенд-частиною додатку.

```
const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');

router.post('/register', authController.register);
router.post('/login', authController.login);
router.get('/me', require('../middleware/authMiddleware'), authController.getMe);

module.exports = router;
```

Рис. 4.6. Routes/auth.js елемент системи.

Файл server.js відповідає за налаштування серверної частини нашого проекту. Його головна мета – встановити з'єднання з базою даних за допомогою Sequelize (ORM для реляційних баз даних) та запустити HTTP-сервер, щоб забезпечити доступ до API для обробки запитів з фронтенду.

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const app = express();
const db = require('./models');

app.use(cors());
app.use(express.json());

app.use('/api/auth', require('./routes/auth.js'));
app.use('/api/user', require('./routes/user.js'));
app.use('/api/services', require('./routes/service.js'));
app.use('/api/tags', require('./routes/tag.js'));
app.use('/api/user/tags', require('./routes/userTag'));

const PORT = process.env.PORT || 3001;
db.sequelize.sync().then(() => {
  app.listen(PORT, () => {
    console.log(`Server running on http://localhost:${PORT}`);
  });
});
```

Рис. 4.7. server.js файл.

4.1.2 Структура веб-застосунку (React)

Реалізація повноцінної системи контролю сервісів, що надаються користувачу в мережі Інтернет, просто неможлива без створення вебдодатку. Це ключовий елемент проєкту, який надає користувачам інтуїтивно зрозумілий інтерфейс для взаємодії з розробленою інформаційною системою. Вебдодаток створено на базі React, що дозволяє забезпечити динамічний та зручний користувацький досвід.

Хоча весь інтерфейс рендериться за допомогою React, важливо розглядати структуру вебсайту окремо та детально. Як і на серверній частині, клієнтська частина також розбита на модулі. Такий підхід забезпечує гнучкість, масштабованість і спрощує підтримку, оновлення та розвиток системи. Завдяки модульності вебдодаток стає більш зручним для подальшого розширення функціоналу та адаптації під потреби користувачів.

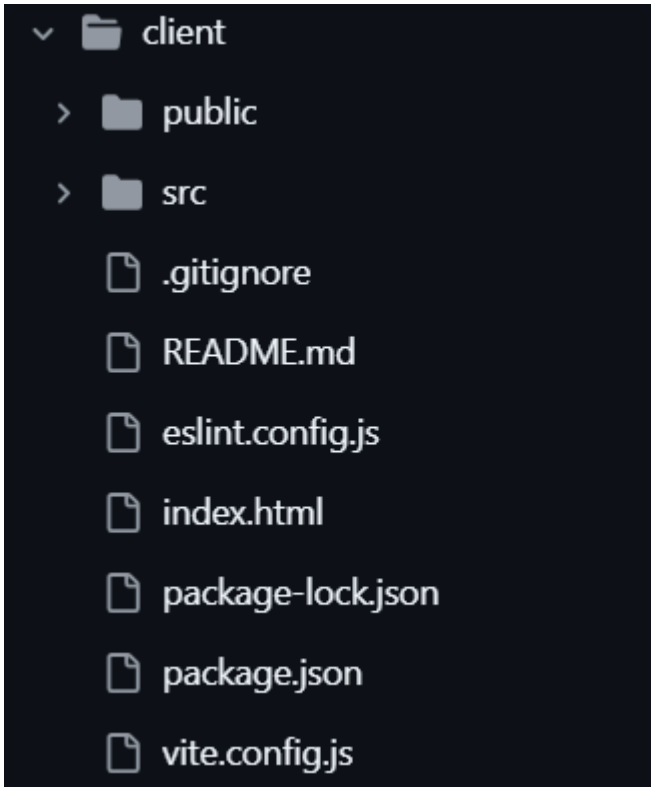

Рис. 4.8. Клієнтська структура сайту.

Структура вебдодатку для системи контролю сервісів, що надаються користувачу в мережі Інтернет, складається з кількох основних каталогів і файлів, які

допомагають зручно розподілити завдання та організувати код. Точка входу, де починається основний компонент додатку, – це файл src/main.jsx. Оскільки клієнтська частина розроблена на React, цей підхід дозволяє використовувати JavaScript для створення розмітки, замість традиційного HTML. У цьому файлі також підключаються глобальні стилі та налаштовуються маршрути для навігації між сторінками додатку.

```jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { BrowserRouter } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <BrowserRouter>
      <AuthProvider>
        <App />
      </AuthProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

Рис. 4.9. React структура main.jsx.

Компоненти інтерфейсу, які можна використовувати повторно, такі як картки товарів, кнопки, форми та інші елементи, розміщені в каталозі src/components. Усі ці компоненти створюються з урахуванням принципу їх можливого повторного використання в інших частинах додатку, що забезпечує гнучкість і зручність. Щоб відрізнити компоненти від звичайних JavaScript файлів, використовуються різні розширення: файли компонентів мають розширення .jsx, що вказує на використання HTML-подібного коду в React, тоді як файли з логікою або звичайним JavaScript кодом використовують розширення .js.

```jsx
import { Link } from 'react-router-dom';
import { useAuth } from '../hooks/useAuth';

const Navbar = () => {
  const { user, logout } = useAuth();

  if (!user) {
    return null;
  }

  return (
    <nav style={navStyle}>
      <div style={leftSideStyle}>
        <Link to="/home" style={linkStyle}>Головна</Link>
        <Link to="/catalog" style={linkStyle}>Каталог</Link>
        <Link to="/saved" style={linkStyle}>Збережені</Link>
      </div>
      <div style={rightSideStyle}>
        {user.role === 'admin' && (
          <span style={adminTextStyle}>Вітаю, адміністраторе!</span>
        )}
        <Link to="/profile" style={linkStyle}>Профіль</Link>
        <button onClick={logout} style={logoutButtonStyle}>Вийти</button>
      </div>
    </nav>
  );
};
```

Рис. 4.10. Код панелі навігації сайту.

У вебдодатку для системи контролю сервісів, що надаються користувачу в мережі Інтернет, кожен компонент має свої власні стилі, які реалізуються за допомогою стандартних CSS файлів або інших технологій для стилізації. Для зручності та ефективності ці стилі зазвичай зберігаються в окремих файлах, що відповідають компонентам. Використання бібліотеки react-router-dom дозволяє налаштовувати маршрути та керувати навігацією між сторінками без перезавантаження, що створює динамічний користувацький досвід.

```jsx
import { Routes, Route, Navigate } from 'react-router-dom';

import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import CatalogPage from './pages/CatalogPage';
import SavedPage from './pages/SavedPage';
import TagsPage from './pages/TagsPage';
import ProfilePage from './pages/ProfilePage';
import NotFoundPage from './pages/NotFoundPage';
import HomePage from './pages/HomePage';
import AuthorizedHomePage from './pages/AuthorizedHomePage';

import PrivateRoute from './components/PrivateRoute';
import Navbar from './components/Navbar';
function App() {
  return (
    <>
      <Navbar />
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/login" element={<LoginPage />} />
        <Route path="/register" element={<RegisterPage />} />

        <Route path="/home" element={
          <PrivateRoute>
            <AuthorizedHomePage />
          </PrivateRoute>
        } />
        <Route path="/catalog" element={
          <PrivateRoute>
            <CatalogPage />
          </PrivateRoute>
        } />
```

Рис. 4.11. Частина коду файлу App.jsx.

Компоненти інтерфейсу, такі як картки сервісів, кнопки, форми та інші елементи, створюються за допомогою React. Це дає можливість додавати до стандартних HTML елементів необхідну стилізацію та функціональність, що робить їх зручними для повторного використання в різних частинах додатку. Завдяки такому підходу кожен компонент може мати свій унікальний вигляд і поведінку, а також бути адаптованим до різних вимог.

```jsx
import axiosInstance from '../api/axiosInstance';
import { useState } from 'react';
import styles from './ServiceCard.module.css';

const ServiceCard = ({ service }) => {
  const [message, setMessage] = useState('');

  const handleSave = async () => {
    try {
      await axiosInstance.post('/user/saved', { serviceId: service.id });
      showMessage('Сервіс збережено!');
    } catch (error) {
      console.error('Помилка при збереженні сервісу:', error);
    }
  };

  const showMessage = (text) => {
    setMessage(text);
    setTimeout(() => {
      setMessage('');
    }, 2000);
  };

  const verifiedBadgeStyle = {
    backgroundColor: '#28a745',
    color: 'white',
    padding: '4px 10px',
    borderRadius: '20px',
    fontSize: '12px',
    display: 'inline-block',
    fontWeight: 'bold',
  };
```

Рис. 4.12. Компонент системи ServiceCard.jsx.

Важливим аспектом є використання глобальних стилів і змінних. Глобальні стилі допомагають скинути стандартну стилізацію елементів, що дозволяє уникнути проблем з накладенням стилів під час розробки. Змінні, наприклад, для кольорів у RGB-форматі, допомагають уникнути дублювання коду, покращуючи його читабельність і спрощуючи підтримку та оновлення стилів у майбутньому.

```css
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
```

Рис. 4.13. Файл глобальних стилів App.css.

Веб-застосунок був створений з акцентом на надання інформаційних сервісів користувачам в Інтернеті та забезпечення зручного інтерфейсу. Це багатосторінковий застосунок, де всі основні блоки розташовані вертикально, що робить навігацію логічною та взаємодію простою. Хоча наразі застосунок не має повної адаптивності для різних пристроїв, він розроблений на основі сучасних аналогів інформаційних систем з мінімалістичним дизайном і зрозумілими функціональними блоками.

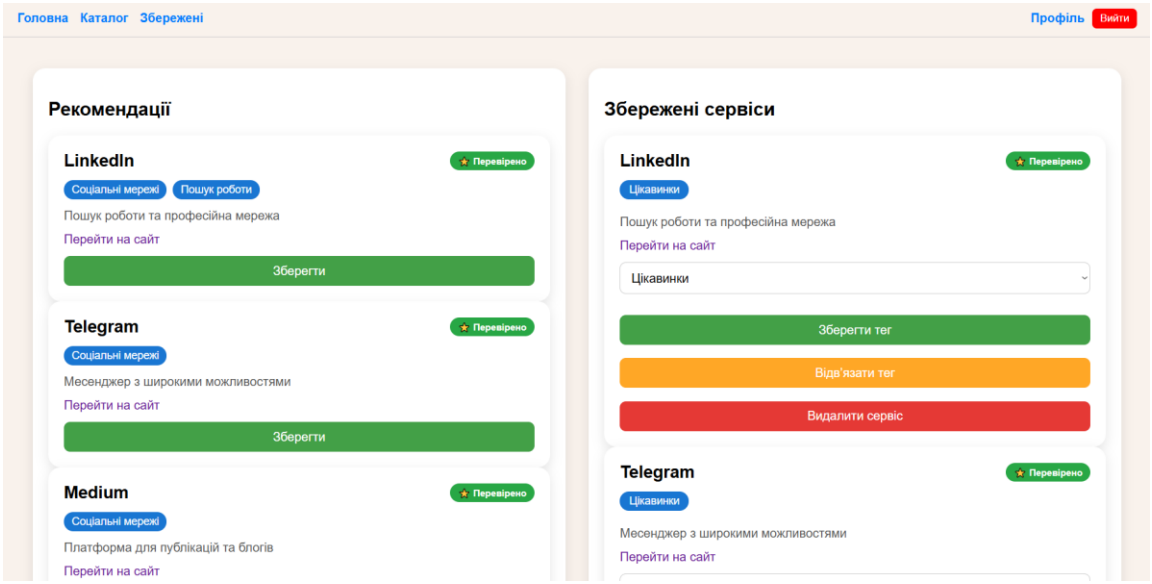

Рис. 4.14. Головна сторінка авторизованого користувача.

Більшість навігаційних посилань на вебсайті ведуть до відповідних секцій системи. Наприклад, якщо авторизований користувач натисне кнопку «Каталог», він буде переадресований на окрему сторінку з переліком доступних сервісів.

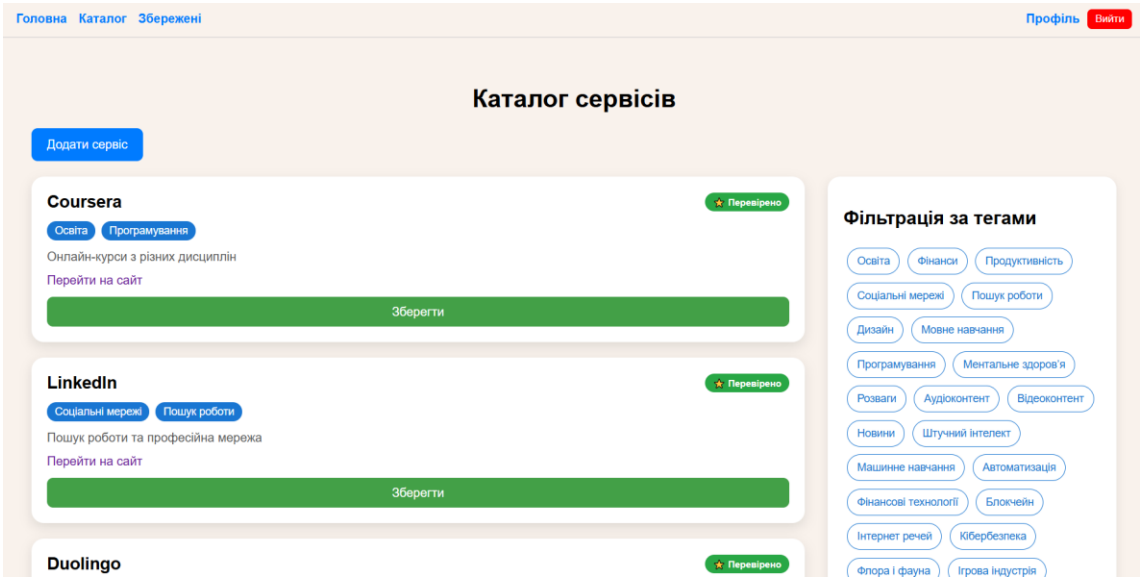

Рис. 4.15. Сторінка каталогу.

У цьому веб-додатку реалізовано 6 окремих сторінок, серед яких:

- Головна сторінка неавторизованого користувача, яка знайомить відвідувачів з основними функціями сервісу, демонструє переваги використання системи та мотивує зареєструватися.
- Сторінки реєстрації та авторизації, які дозволяють новим користувачам створити обліковий запис, а вже зареєстрованим – увійти до системи. Обидві сторінки мають подібний дизайн та логічно об’єднані в один пункт опису.
- Головна сторінка авторизованого користувача, що відображає персоналізовану інформацію, наприклад рекомендації та збережені сервіси [19].
- Каталог, у якому представлено перелік доступних сервісів, з можливістю фільтрації та пошуку за різними параметрами.
- Профіль користувача, де можна редагувати рекомендації, додавати та видаляти користувацькі теги для збережених сервісів.
- Сторінка збережених, яка дозволяє користувачеві переглядати та керувати збереженими сервісами.

4.1.3 JavaScript-скрипти та API

В рамках проекту, що налагоджує комунікацію між клієнтською та серверною сторонами, застосовано бібліотеку Axios – відомий інструмент для здійснення HTTP-запитів [3]. Axios робить роботу з асинхронними запитами дуже зручною і підтримує всі необхідні HTTP-методи, такі як GET, POST, PUT і DELETE [27]. Усі запити до серверної частини інформаційної системи, яка контролює сервіси, що надаються користувачам в Інтернеті, виконуються саме через Axios. Це дозволяє ефективно обмінюватися даними без потреби перезавантажувати сторінку.

```
const RegisterPage = () => {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [message, setMessage] = useState('');

  const handleRegister = async (e) => {
    e.preventDefault();

    if (password !== confirmPassword) {
      setMessage('Паролі не збігаються!');
      return;
    }

    try {
      await axiosInstance.post('/auth/register', { name, email, password });
      window.location.href = '/login';
    } catch (error) {
      console.error('Помилка реєстрації:', error);
      setMessage('Помилка реєстрації');
    }
  };
```

Рис. 4.16. Фрагмент коду для реєстрації.

Окрім роботи з клієнтською частиною, для ручного тестування API під час розробки використовувався Postman. Завдяки йому було перевірено, як працюють усі серверні маршрути, такі як реєстрація користувачів, отримання даних про сервіси, управління профілем і так далі. У Postman вручну налаштовувались запити, вказуючи HTTP-метод, необхідні заголовки та тіло запиту у форматі JSON.

Наприклад, спочатку інформаційна система не має зареєстрованих користувачів, окрім тих, хто був доданий до бази даних вручну. Коли користувач вперше відвідує сайт, його статус – Гість. Щоб отримати доступ до всіх можливостей системи, потрібно пройти процедуру реєстрації. Якщо ж у користувача вже є обліковий запис, він може авторизуватися, ввівши свої реєстраційні дані.

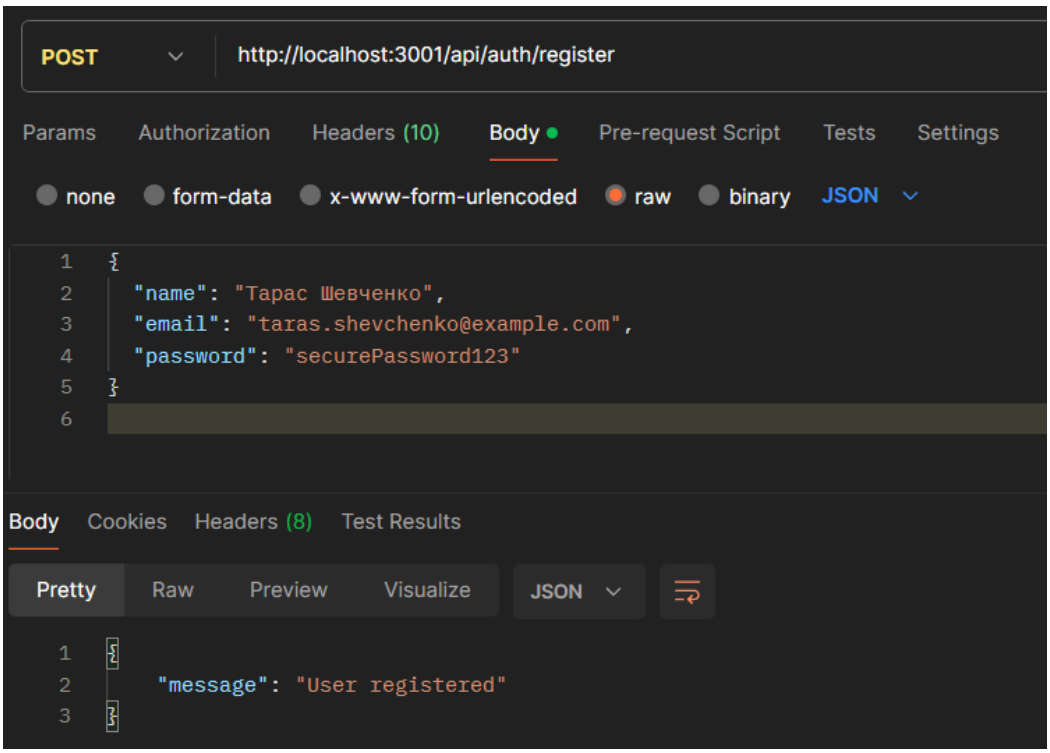

Рис. 4.17. POST метод реєстрації користувача.

Повертаючись до використання JavaScript у моєму проєкті, варто зазначити, що основна логіка взаємодії користувача з інформаційною системою реалізована через скрипти. Зокрема, розроблено функції для обробки кліків на кнопках: додавання сервісів до списку обраних, перегляд детальної інформації про вибраний сервіс і так далі.

Наприклад, коли ви переходите на сторінку каталогу через меню навігації, завантажується повний список доступних онлайн-сервісів із сервера. На цій сторінці реалізовано логіку фільтрації сервісів за певними параметрами, що значно спрощує вибір для користувача. Крім того, JavaScript забезпечує динамічне оновлення контенту без необхідності перезавантаження сторінки, що покращує загальний користувацький досвід.

Особливу увагу виділено перевірці даних на боці клієнта, що дозволяє уникнути помилок при введенні інформації та підвищує надійність роботи інформаційної системи. Також були розроблені анімаційні ефекти для покращення візуального сприйняття інтерфейсу.

```
const CatalogPage = () => {
  const [services, setServices] = useState([]);
  const [tags, setTags] = useState([]);
  const [selectedTags, setSelectedTags] = useState([]);

  const [showModal, setShowModal] = useState(false);
  const [name, setName] = useState('');
  const [description, setDescription] = useState('');
  const [url, setUrl] = useState('');
  const [selectedTagId, setSelectedTagId] = useState('');
  const [addedTagIds, setAddedTagIds] = useState([]);

  useEffect(() => {
    fetchTags();
    fetchServices();
  }, []);

  const fetchTags = async () => {
    const response = await axiosInstance.get('/tags');
    setTags(response.data);
  };

  const fetchServices = async (tagIds = []) => {
    if (tagIds.length > 0) {
      const response = await axiosInstance.post('/services/search-by-tags', { tagIds });
      setServices(response.data);
    } else {
      const response = await axiosInstance.get('/services');
      setServices(response.data);
    }
  };
```

Рис. 4.18. Фрагмент коду сторінки каталогу.

У репозиторії GitHub за посиланням можна повністю ознайомитись з кодом програмної реалізації: https://github.com/SheamBest/Up2You-Service

4.2 Інструкція користувача

Вступ

Інформаційна система контролю сервісів, що надаються користувачу в мережі Інтернет. представлена у вигляді веб-сайту, який дозволяє користувачам переглядати каталог доступних онлайн-сервісів і обирати ті, що їм потрібні для підключення чи використання. Відвідувач може отримати детальну інформацію про кожен сервіс, додати обрані варіанти до списку збережених, а також додавати власні теги до них.

Крім основного функціоналу, сайт також має розділи з контактною інформацією та посиланнями на профілі в соціальних мережах, що покращує комунікацію з користувачами і робить платформу більш зручною для використання.

Загальні відомості про програму

Розроблена система контролю сервісів, що надаються користувачу в мережі Інтернет, функціонує на основі двох ключових складових: клієнтської частини, збудованої за допомогою React з інтеграцією Vite для прискореної збірки та розробки, та серверної частини, реалізованої на Node.js із застосуванням фреймворку Express.js. Взаємодія між клієнтом та сервером забезпечується через Axios, який ефективно управляє HTTP-запитами. Серверна частина реалізує взаємодію з базою даних MySQL, використовуючи спеціалізовані драйвери для зберігання та опрацювання інформації про сервіси.

Для тестування та налагодження API активно використовувався Postman, що дозволяв перевіряти правильність обробки запитів на всіх етапах розробки.

Ця інформаційна система реалізована у вигляді веб-застосунку, який працює локально, без необхідності розгортання на зовнішніх платформах. Основна мета створення цієї системи – автоматизація процесу контролю та моніторингу онлайн-сервісів: користувачі можуть переглядати каталог сервісів, додавати їх до списку збережених, відстежувати їх та додавати власні теги.

Класи вирішуваних завдань

Основними завданнями, які були реалізовані в рамках проєкту, стали: надання користувачам повного доступу до каталогу онлайн-сервісів, впровадження механізму фільтрації сервісів за рекомендаціями, а також створення можливостей для збереження сервісів і їх класифікації за власними тегами.

Крім того, у системі передбачена головна сторінка, яка містить загальну інформацію про призначення платформи, а також контактні дані для зв’язку. Щодо адаптивності, веб-застосунок обмежено підтримує різні типи пристроїв, але дозволяє зручно переглядати інтерфейс як на настільних, так і на мобільних екранах.

Відомості про функціональні обмеження на застосування

Через специфічні технічні рішення та деякі обмеження в реалізації інтерфейсу, наша інформаційна система контролю сервісів має певні недоліки в зручності використання. Хоч використано сучасні фреймворки для фронтенду (React + Vite) та бекенду (Node.js + Express), веб-застосунок не зовсім адаптивний. Він працює добре на екранах ноутбуків і стаціонарних ПК, але не забезпечує належної підтримки для мобільних пристроїв або планшетів, що може ускладнити його використання на екранах з обмеженим простором.

4.3 Аналіз контрольного прикладу

Цей розділ надає опис функціоналу інформаційної системи контролю сервісів, які доступні користувачам в Інтернеті під назвою «Up2You Service». Аналіз проводиться з точки зору клієнта і охоплює, як користувач взаємодіє з основними модулями, функціональними блоками та сторінками веб-застосунку.

Веб-застосунок Up2You Service – це система, що дозволяє переглядати, обирати та контролювати онлайн-сервіси. Інтерфейс виконано в мінімалістичному стилі, з акцентом на простоту та зручність навігації. Коли користувач вперше заходить на сайт, він одразу бачить головну сторінку, де розповідається про систему та переваги нею користуватись. Під текстом одразу розміщена кнопка «Зареєструватися», щоб користувач міг створити собі обліковий запис. Зверху справа розташована кнопка «Увійти», де користувач з обліковим записом може увійти у систему.

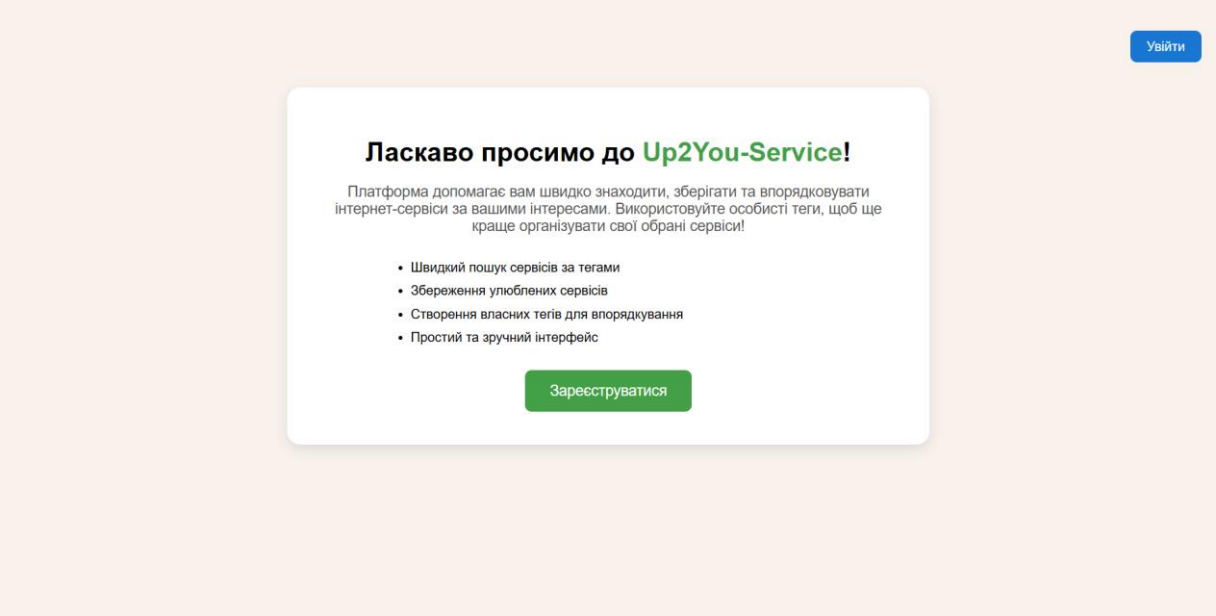

Рис. 4.19. Головна сторінка неавторизованого користувача.

Умовно користувач створив новий акаунт та увійшов до системи, він переходить до головної сторінки авторизованого користувача. Зверху можна побачити панель навігації з кнопками «Головна», «Каталог», «Збережені», «Профіль» та «Вийти». Перші три кнопки на панелі розташовані ліворуч, інші дві – праворуч. На сторінці також виведені списки «Рекомендації» та «Збережені сервіси». Так як акаунт новий, ці списки пусті [26].

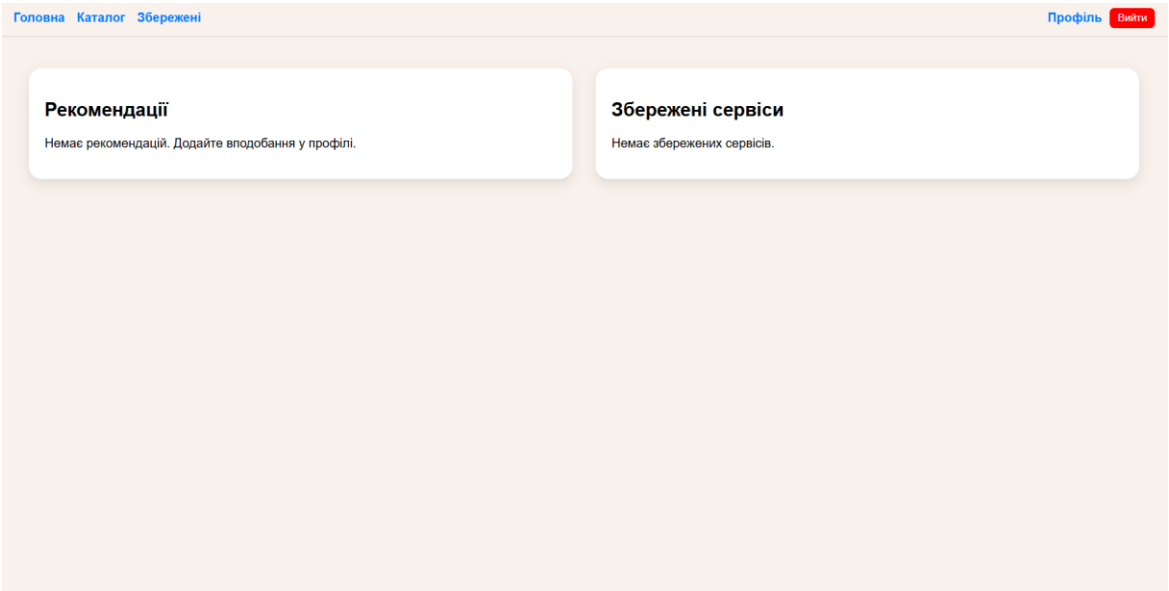

Рис. 4.20. Головна сторінка авторизованого користувача.

Переходячи до вкладки «Каталог», користувач може бачити перелік усіх сервісів та перелік тегів. Без фільтрації сервіси виводяться у порядку розташування в базі даних, з фільтрацією показуються сервіси з обраними тегами.

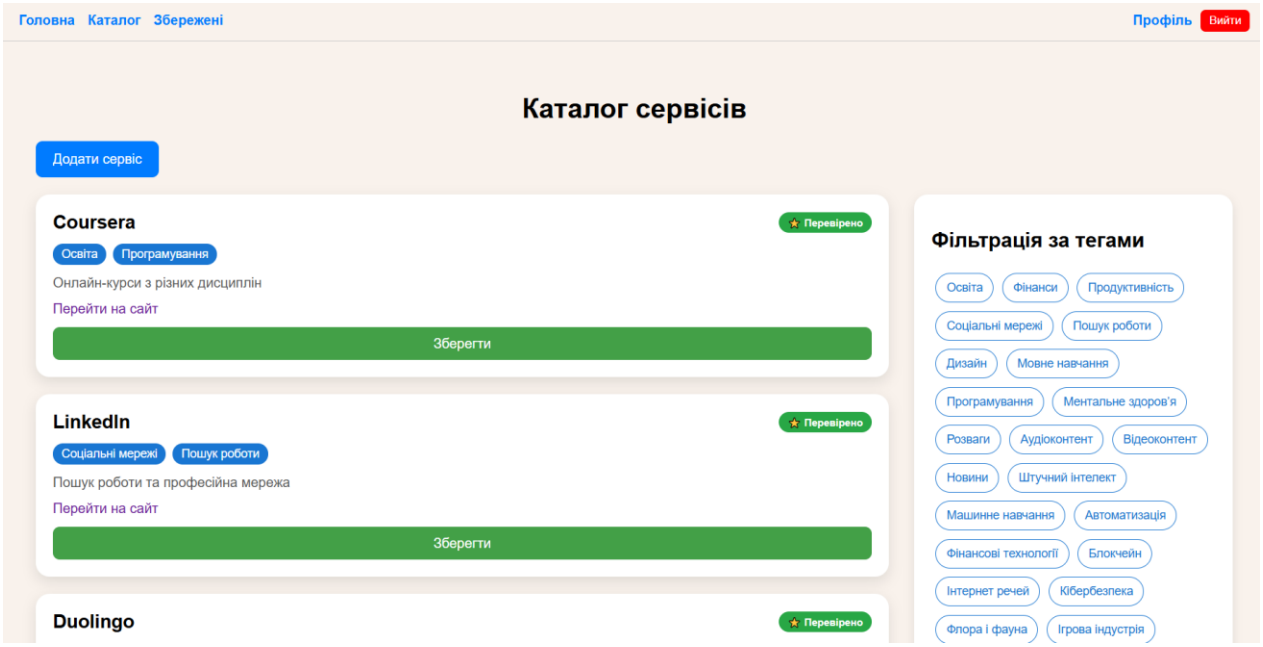

Рис. 4.21. Сторінка каталогу системи.

Також на сторінці розташована кнопка «Додати сервіс», якою користувач може додати власний сервіс до бази даних. Натиснувши її, відкривається модальне вікно з полями для введення даних. Після усіх операцій сервіс додається до загального переліку сервісів з однією відмінністю: додані звичайними користувачами сервіси не матимуть відмітки «Перевірено».

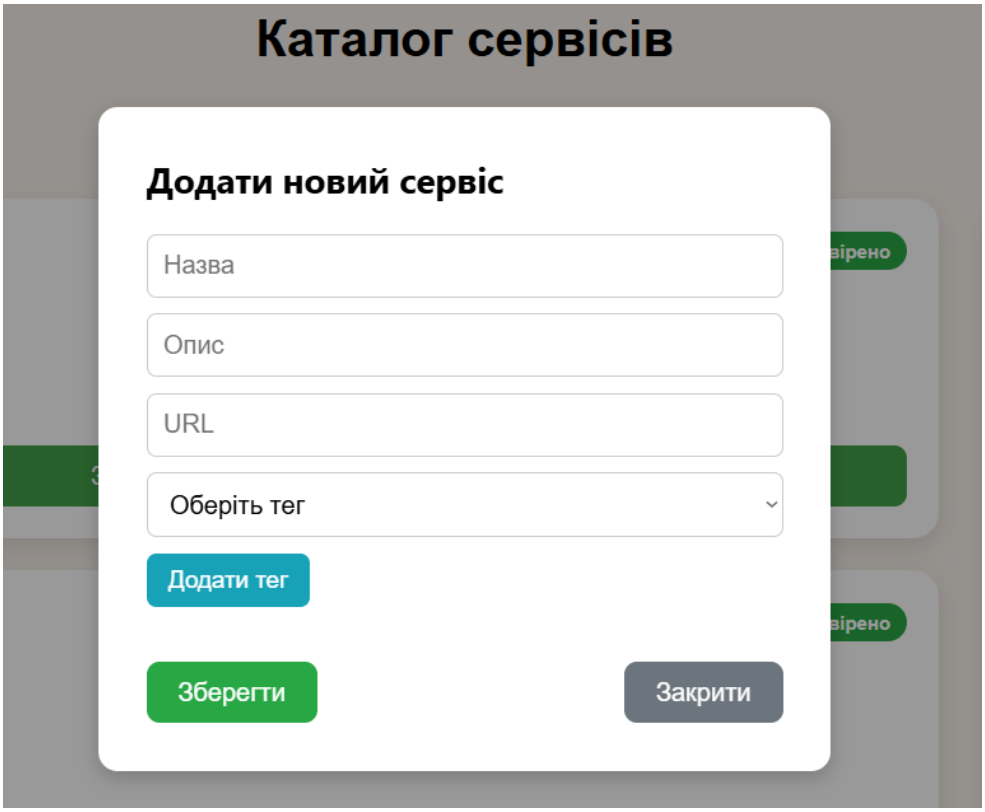

Рис. 4.22. Модальне вікно для додавання сервісу.

У вкладці «Збережені» розташовані збережені сервіси. Умовно користувач додав один сервіс до збережених. З збережними сервісами можна проводити виконувати такі дії: додавати та видаляти користувацькі теги, видаляти сервіс з збережених.

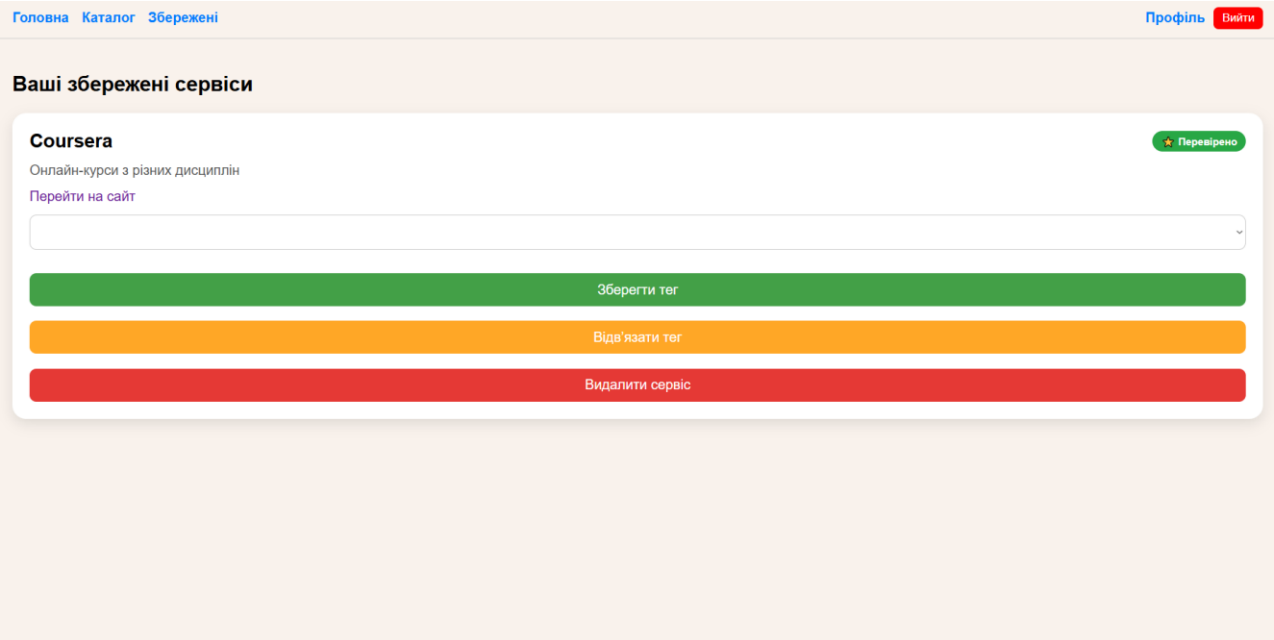

Рис. 4.23. Сторінка «Збережені».

Перейдемо до вкладки «Профіль». У ній користувач може обрати та видаляти теги для вподобань, зокрема було обрано тег «Програмування». Також на цій сторінці можна створювати та видаляти власні теги для прив’язування до збережених сервісів. Унизу сторінки є стилізована кнопка виходу з облікового запису.

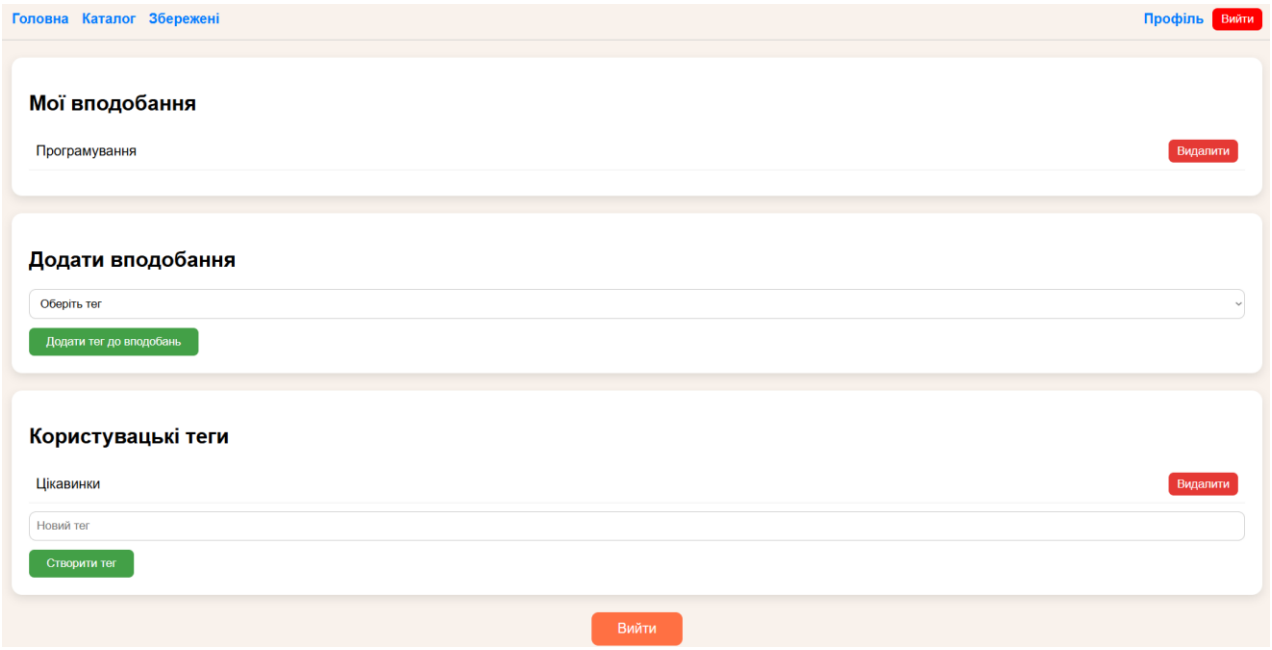

Рис. 4.24. Сторінка «Профіль».

Проходячи всі сторінки, користувач поволі обрав теги для вподобань, зберіг сервіс до збережених, створив власний тег та прикріпив його до збереженого сервісу. В такому випадку змінився вигляд головної сторінки, де у списку «Рекомендації» показані сервіси з відповідним тегом, а у списку «Збережені сервіси» – вподобаний сервіс з прикріпленим користувацьким тегом [20].

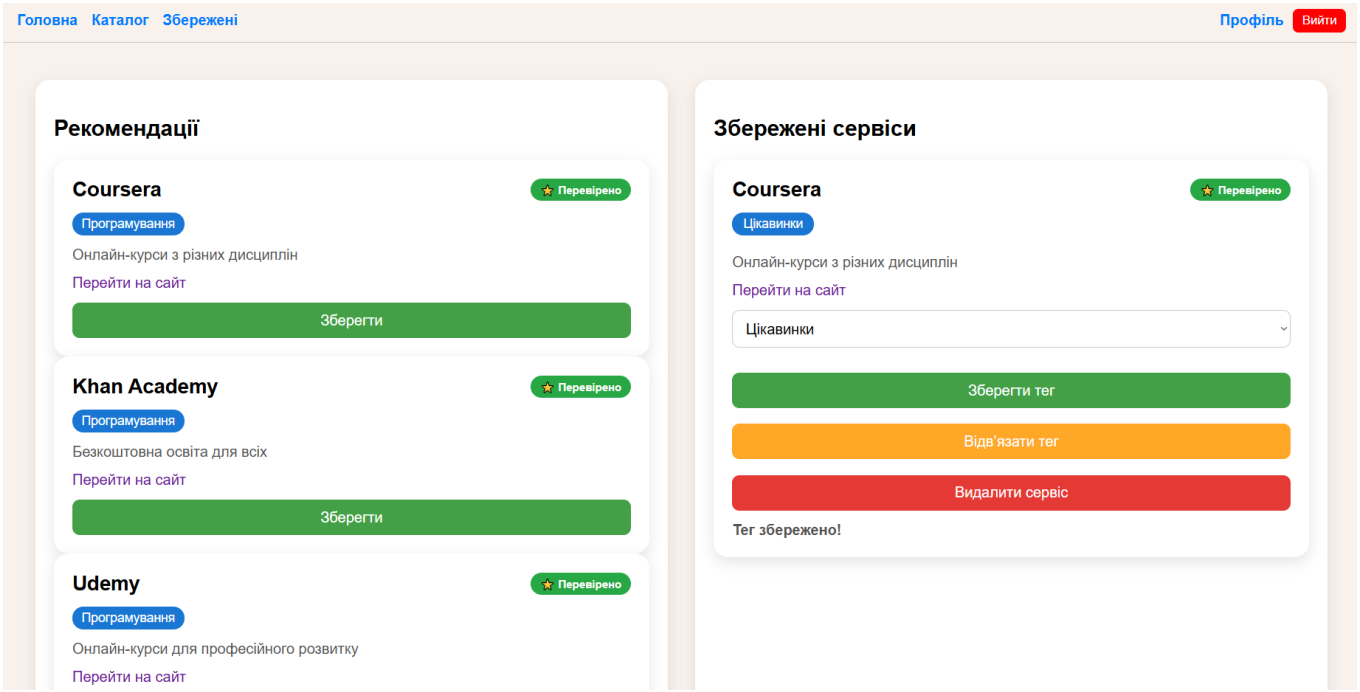

Рис. 4.25. Змінена головна сторінка авторизованого користувача.

Висновок до розділу 4

На завершення цього розділу, який присвячений розробці інформаційної системи для контролю сервісів, що надаються користувачам в мережі Інтернет у формі веб-застосунку, проведено детальний аналіз дизайну та функціональних можливостей програмної реалізації. Система надає користувачам доступ до каталогу онлайн-сервісів, дозволяє переглядати детальну інформацію, формувати списки рекомендованих та збережених сервісів, додавати сервіси та власні теги.

У цьому розділі також розглянуті технічні аспекти реалізації, зокрема інструменти та технології, які ми використали, такі як React з Vite, Express.js і база даних MySQL. Для зручної взаємодії між клієнтом і сервером ми застосували бібліотеку Axios.

Особливу увагу приділено створенню інструкції користувача, яка допоможе спростити навігацію по сайту та ефективно використовувати його функціонал. У ній описані основні можливості системи, ключові функції та поточні обмеження в її використанні. Також проведено аналіз контрольного прикладу, який підтверджує працездатність системи, її зручність і відповідність поставленим завданням.

ВИСНОВКИ

Створення інформаційної системи контролю сервісів, що надаються користувачу в мережі Інтернет, стало наслідком поетапного, ретельно вибудованого процесу, який об'єднав аналітику, системне моделювання та практичне втілення. Головною метою було забезпечення простого доступу до цифрових сервісів, їх індивідуалізованого представлення та подальшого ефективного керування з боку споживача.

В першому розділі було проведено огляд сучасних цифрових рішень у сфері агрегаторів сервісів. Аналіз конкуруючих платформ допоміг виявити типові недоліки (обмежена персоналізація, перевантажений інтерфейс, складність навігації) та сформулювати вимоги до функціональності і зручності майбутньої системи.

Другий розділ присвячений системному аналізу. Побудоване дерево цілей дало змогу виокремити ключові напрями: персоналізацію, актуальність даних, аналітику та рекомендації. Використання методів IDEF0 та AHP (методу аналізу ієрархій) сприяло формуванню цілісної логічної структури, визначенню пріоритетності критеріїв якості та обґрунтуванню вибору найбільш ефективної архітектури системи.

У третьому розділі обґрунтовано вибір технологічного стеку: Node.js + Express.js для серверної частини, MySQL як система управління базами даних, React + Vite для фронтенду. Також використано Axios для HTTP-запитів та Postman для тестування API. Обрані інструменти забезпечили надійність, масштабованість та простоту підтримки проекту.

Четвертий розділ демонструє реалізацію веб-застосунку та описує повний функціонал: авторизацію, керування вподобаннями, перегляд каталогу сервісів, фільтрацію, додавання сервісів до збережених, генерацію рекомендацій, а також можливість користувача додавати власні сервіси й теги. Створено інструкцію користувача для полегшення взаємодії з платформою.

У підсумку реалізовано повноцінну інформаційну систему, що забезпечує:

– зручну взаємодію з каталогом сервісів;

– персоналізацію та збереження вподобань;

– надання рекомендацій на основі інтересів користувача;

– прозору архітектуру з можливістю подальшого розширення функціоналу.

Система може бути застосована як основа для запуску комерційного або навчального проекту з управління онлайн-сервісами в різних галузях діяльності. Її масштабованість та орієнтація на користувача відкривають перспективи для інтеграції в більші цифрові екосистеми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AlternativeTo – Crowdsourced software recommendations. URL: https://alternativeto.net/ (дата звернення: 01.04.2025)

2. Applying System Thinking and AHP in Software Projects. URL: https://www.researchgate.net/publication/327957845 (дата звернення: 01.04.2025)

3. Axios in React: A Guide for Beginners. URL: https://www.geeksforgeeks.org/axios-in-react-a-guide-for-beginners/ (дата звернення: 01.04.2025)

4. COVID-19 and the acceleration of digital transformation. URL: https://www.weforum.org/agenda/2020/11/digital-transformation-covid19/ (дата звернення: 01.04.2025)

5. CSS Tricks – Responsive Design Basics. URL: https://css-tricks.com/snippets/css/media-queries-for-standard-devices/ (дата звернення: 09.04.2025)

6. Digital public services in the COVID era. URL: https://ec.europa.eu/digital-strategy/news/digital-public-services-and-covid19_en (дата звернення: 09.04.2025)

7. ER Diagrams: A Complete Guide. URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 20.04.2025)

8. Express.js – The Fast, Unopinionated, Minimalist Web Framework for Node.js. URL: https://expressjs.com/ (дата звернення: 11.04.2025)

9. Future of Service Aggregator Platforms. URL: https://www.researchgate.net/publication/351954752 (дата звернення: 01.04.2025)

10. Hierarchy Models in System Analysis. URL: https://www.tandfonline.com/doi/full/10.1080/00207543.2020.1824080 (дата звернення: 01.04.2025)

11. How Search Engines Use Tags and Metadata. URL: https://moz.com/learn/seo/meta-description (дата звернення: 01.04.2025)

12. How the Pandemic Changed Consumer Behavior. URL: https://hbr.org/2020/05/how-covid-19-is-changing-consumer-behavior (дата звернення: 01.04.2025)

13. IDEF0 Function Modeling Methodology. URL: https://www.sciencedirect.com/topics/computer-science/idef0 (дата звернення: 05.04.2025)

14. Introduction to Node.js. URL: https://nodejs.dev/en/learn/ (дата звернення: 10.04.2025)

15. JavaScript Event Listeners Explained. URL: https://developer.mozilla.org/en-US/docs/Web/API/EventListener (дата звернення: 10.04.2025)

16. MySQL :: MySQL 8.0 Reference Manual. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 15.04.2025)

17. Online Platforms and User-Centered Access During Crisis. URL: https://link.springer.com/article/10.1007/s10209-021-00785-5 (дата звернення: 10.04.2025)

18. Online Services Aggregators: Challenges and Trends. URL: https://link.springer.com/article/10.1007/s00779-021-01591-4 (дата звернення: 10.04.2025)

19. Personalization and Context-Aware Recommender Systems. URL: https://www.sciencedirect.com/science/article/pii/S0306457321001702 (дата звернення: 10.04.2025)

20. Personalization in Web Applications: Techniques and Tools. URL: https://www.sciencedirect.com/science/article/abs/pii/S0747563220303547 (дата звернення: 15.04.2025)

21. Platforms Aggregating Digital Services: A Comparative Study. URL: https://www.sciencedirect.com/science/article/pii/S0160791X20303485 (дата звернення: 07.04.2025)

22. Postman API Platform. URL: https://www.postman.com/product/api-client/ (дата звернення: 20.04.2025)

23. Product Hunt – Discover your next favorite thing. URL: https://www.producthunt.com/ (дата звернення: 01.04.2025)

24. React – A JavaScript library for building user interfaces. URL: https://react.dev/ (дата звернення: 10.04.2025)

25. React Router – Declarative Routing for React.js. URL: https://reactrouter.com/ (дата звернення: 10.04.2025)

26. Recommendation Systems: A Survey. URL: https://dl.acm.org/doi/10.1145/1454008.1454009 (дата звернення: 10.04.2025)

27. REST API Tutorial – Learn REST with Examples. URL: https://restfulapi.net/ (дата звернення: 10.04.2025)

28. Role of Aggregator Platforms in Modern Society. URL: https://www.sciencedirect.com/science/article/pii/S0740624X22000599 (дата звернення: 10.04.2025)

29. Saaty T.L. – Decision making with the analytic hierarchy process. URL: https://link.springer.com/article/10.1007/s00146-008-0097-0 (дата звернення: 15.04.2025)

30. Scalable Web Architecture and Distributed Systems. URL: https://martinfowler.com/articles/web-scale.html (дата звернення: 11.04.2025)

31. Sequelize – Node.js ORM for Postgres, MySQL. URL: https://sequelize.org/ (дата звернення: 09.04.2025)

32. Security Best Practices for Web Applications. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 09.04.2025)

33. SimilarSites – Discover Similar Sites. URL: https://www.similarsites.com/ (дата звернення: 01.04.2025)

34. Tags and Faceted Navigation in Information Systems. URL: https://www.nngroup.com/articles/faceted-search/ (дата звернення: 11.04.2025)

35. The Importance of User Feedback in Software Design. URL: https://uxdesign.cc/why-user-feedback-is-vital-to-product-design-530c4562f7b0 (дата звернення: 12.04.2025)

36. The Role of Goal Trees in IS Design. URL: https://ieeexplore.ieee.org/document/8560196 (дата звернення: 11.04.2025)

37. UI/UX Principles for Web Applications. URL: https://uxdesign.cc/essential-ui-ux-principles-for-web-designers-6b0c0fbb84f2 (дата звернення: 10.04.2025)

38. UX Design for Web Platforms: Key Concepts. URL: https://uxplanet.org/ux-design-principles-for-web-applications-92d4b39a6d60 (дата звернення: 11.04.2025)

39. Vite – Next Generation Frontend Tooling. URL: https://vitejs.dev/ (дата звернення: 10.04.2025)

40. Visual Studio Code – Code Editing. Redefined. URL: https://code.visualstudio.com/ (дата звернення: 06.04.2025)

АНОТАЦІЯ

Шельвах М.А., Григорович В.Г. (керівник). Інформаційна система контролю сервісів, що надаються користувачу в мережі Інтернет. Бакалаврська кваліфікаційна робота – Національний університет «Львівська Політехніка», Львів, 2025.

Розширена анотація.

Проблема недостатньої цифровізації бізнес-процесів у сфері надання онлайн-сервісів стала одним із ключових бар'єрів на шляху до якісної та ефективної взаємодії з користувачами. Попри стрімке зростання попиту на зручні цифрові рішення, багато платформ досі не використовують сучасні IT-інструменти для інформування, підтримки й персоналізованого супроводу користувачів. Сучасний користувач має обмежену кількість часу та очікує, що сервіс буде доступний онлайн – через інтуїтивно зрозумілий веб-застосунок або сайт, де можна швидко знайти потрібну інформацію, додати сервіси до обраного, переглянути статуси або залишити заявку на використання.

У конкурентному середовищі цифрових платформ саме рівень автоматизації та якість інформаційної системи стають визначальними факторами успіху. Очікування користувачів щодо швидкого доступу, зручного інтерфейсу та безперебійної роботи системи вимагають сучасних підходів до архітектури та реалізації. Проблеми більшості існуючих рішень полягають у відсутності персоналізації, обмеженій функціональності, низькій швидкості роботи або незрозумілому інтерфейсі. Саме тому було обрано тему створення якісної інформаційної системи контролю сервісів, що надаються користувачу в Інтернеті, яка відповідатиме реальним потребам як самих користувачів, так і адміністраторів платформи.

Об'єкт дослідження. Об'єктом дослідження є процеси надання та контролю сервісів користувачам у мережі Інтернет.

Предмет дослідження. Предметом дослідження є методи та засоби розробки та програмного втілення інформаційної системи, що гарантує накопичення інформації,

відслідковування, вивчення і розподіл сервісів, а також надання цих сервісів користувачам відповідно до їх потреб.

Метою цієї роботи є створення інформаційної системи, яка забезпечить ефективний механізм доступу до цифрових сервісів, їх перегляду, фільтрації, додавання до збережених, отримання рекомендацій на основі вподобань, додавання сервісів та власних тегів. Система має надати зручний веб-інтерфейс для користувачів, які зможуть швидко орієнтуватися в каталозі сервісів, застосовувати фільтри за категоріями або тегами, переглядати описи та статуси, а також керувати власним профілем.

Кінцевим результатом є програмна реалізація багатосторінкового веб-застосунку з авторизацією користувачів, персональними акаунтами, динамічним відображенням сервісів, інтеграцією з базою даних MySQL та REST API, реалізованим на Node.js і Express.js. Клієнтська частина створена за допомогою React з використанням Vite для швидкої збірки. Система розроблена з урахуванням можливості зростання, а також планується розвивати її функціональність відповідно до потреб користувачів.

Ключові слова: цифрові сервіси, автоматизація, інформаційна система, веб-застосунок, взаємодія з користувачем.

Перелік використаних джерел:

1. Digital Transformation in Service Industries: A Systematic Review. URL: https://www.sciencedirect.com/science/article/pii/S014829632030570X (дата звернення: 27.04.2025)

2. Web Applications as Service Platforms: Architecture and Development Practices. URL: https://link.springer.com/chapter/10.1007/978-3-030-64861-9_5 (дата звернення: 27.04.2025)

ANNOTATION

Shelvakh M.A., Hryhorovych V.H. (Supervisor). Information System for Monitoring Services Provided to Users via the Internet. Bachelor's Qualification Thesis – Lviv Polytechnic National University, Lviv, 2025.

Extended Abstract

The issue of insufficient digitalization of business processes in the field of online services has become one of the main barriers to high-quality and efficient interaction with users. Despite the rapid growth in demand for convenient digital solutions, many platforms still fail to use modern IT tools for informing, supporting, and personalizing user interaction. Today's users have limited time and expect the service to be available online – via an intuitive web application or website where they can quickly find the necessary information, add services to favorites, view statuses, or submit usage requests.

In the competitive environment of digital platforms, the level of automation and the quality of the information system become critical success factors. Users' expectations for quick access, a user-friendly interface, and uninterrupted system operation require modern approaches to system architecture and implementation. The main problems with most existing solutions include the lack of personalization, limited functionality, low speed, or a confusing interface. Therefore, the topic of creating a high-quality information system for monitoring services provided to users via the Internet was chosen. Such a system should meet the real needs of both users and platform administrators.

Object of research: The object of research is the processes of providing and monitoring services to users over the Internet.

Subject of research: The subject is the methods and tools for developing and implementing an information system that ensures the accumulation, tracking, analysis, and distribution of services, as well as the provision of those services to users based on their needs.

The aim of this thesis is to create an information system that provides an efficient mechanism for accessing digital services, browsing and filtering them, saving to favorites,

receiving personalized recommendations, and managing tags and services. The system should offer a convenient web interface allowing users to quickly navigate the service catalog, apply filters by category or tag, view descriptions and statuses, and manage their personal profile.

The final result is a multi-page web application with user authentication, personal accounts, dynamic service rendering, integration with a MySQL database, and a REST API built with Node.js and Express.js. The frontend is developed using React with Vite for fast builds. The system is designed with scalability in mind and is intended to be further expanded according to user needs.

Keywords: digital services, automation, information system, web application, user interaction.

References:

Digital Transformation in Service Industries: A Systematic Review. URL: https://www.sciencedirect.com/science/article/pii/S014829632030570X (Accessed: April 27, 2025)

Web Applications as Service Platforms: Architecture and Development Practices. URL: https://link.springer.com/chapter/10.1007/978-3-030-64861-9_5 (Accessed: April 27, 2025)

ДОДАТКИ

models/user.js

```
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define('User', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    email: { type: DataTypes.STRING, unique: true, allowNull: false },
    password: { type: DataTypes.STRING, allowNull: false },
    name: { type: DataTypes.STRING, allowNull: false },
    role: { type: DataTypes.ENUM('user', 'admin'), defaultValue: 'user' }
  });

  User.associate = (models) => {
    User.belongsToMany(models.Tag, {
      through: models.Preference,
      foreignKey: 'userId',
      as: 'preferences'
    });

    User.belongsToMany(models.Service, {
      through: models.SavedService,
      foreignKey: 'userId',
      as: 'savedServices'
    });
  };

  return User;
};
```

models/user_tag.js

```
module.exports = (sequelize, DataTypes) => {
  const UserTag = sequelize.define('UserTag', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    name: { type: DataTypes.STRING, allowNull: false },
    userId: { type: DataTypes.INTEGER, allowNull: false }
  });

  UserTag.associate = (models) => {
    UserTag.belongsTo(models.User, { foreignKey: 'userId' });
    UserTag.belongsToMany(models.Service, {
      through: models.SavedService,
```

```
      foreignKey: 'userTagId',
      otherKey: 'serviceId',
      as: 'taggedServices'
    });
  };

  return UserTag;
};
```

models/tag.js

```
module.exports = (sequelize, DataTypes) => {
    const Tag = sequelize.define('Tag', {
      id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
      name: { type: DataTypes.STRING, unique: true, allowNull: false }
    });

    Tag.associate = (models) => {
      Tag.belongsToMany(models.User, {
        through: models.Preference,
        foreignKey: 'tagId',
        as: 'preferredByUsers'
      });

      Tag.belongsToMany(models.Service, {
        through: models.ServiceTag,
        foreignKey: 'tagId',
        as: 'services'
      });
    };

    return Tag;
  };
```

models/service.js

```
module.exports = (sequelize, DataTypes) => {
    const Service = sequelize.define('Service', {
      id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
      name: { type: DataTypes.STRING, allowNull: false },
      description: { type: DataTypes.TEXT },
      url: { type: DataTypes.STRING },
      is_verified: { type: DataTypes.BOOLEAN, defaultValue: false }
    });
```

```
  Service.associate = (models) => {
    Service.belongsTo(models.User, { foreignKey: 'added_by', as: 'author' });

    Service.belongsToMany(models.Tag, {
      through: models.ServiceTag,
      foreignKey: 'serviceId',
      as: 'tags'
    });

    Service.belongsToMany(models.User, {
      through: models.SavedService,
      foreignKey: 'serviceId',
      as: 'savedByUsers'
    });
  };

  return Service;
};
```

models/service_tag.js

```
module.exports = (sequelize, DataTypes) => {
  const ServiceTag = sequelize.define('ServiceTag', {
    serviceId: { type: DataTypes.INTEGER, allowNull: false },
    tagId: { type: DataTypes.INTEGER, allowNull: false }
  });

  return ServiceTag;
};
```

models/saved_service.js

```
module.exports = (sequelize, DataTypes) => {
 const SavedService = sequelize.define('SavedService', {
  userId: { type: DataTypes.INTEGER, allowNull: false },
  serviceId: { type: DataTypes.INTEGER, allowNull: false },
  userTagId: { type: DataTypes.INTEGER, allowNull: true }
 });

 SavedService.associate = (models) => {
  SavedService.belongsTo(models.User, { foreignKey: 'userId' });
  SavedService.belongsTo(models.Service, { foreignKey: 'serviceId' });
  SavedService.belongsTo(models.UserTag, { foreignKey: 'userTagId', as: 'userTag' });
```

```
  };

  return SavedService;
};
```

models/preference.js

```
module.exports = (sequelize, DataTypes) => {
    const Preference = sequelize.define('Preference', {
      userId: { type: DataTypes.INTEGER, allowNull: false },
      tagId: { type: DataTypes.INTEGER, allowNull: false }
    });

    return Preference;
  };
```

middleware/authMiddleware.js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Authorization token missing' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid or expired token' });
  }
};
```

middleware/roleMiddleware.js

```
module.exports = (requiredRole) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ message: 'Unauthorized' });
    }
```

```
    if (req.user.role !== requiredRole) {
      return res.status(403).json({ message: 'Forbidden: insufficient rights' });
    }
    next();
  };
};
```

routes/auth.js

```
const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');

router.post('/register', authController.register);
router.post('/login', authController.login);
router.get('/me', require('../middleware/authMiddleware'), authController.getMe);

module.exports = router;
```

routes/service.js

```
const express = require('express');
const router = express.Router();
const serviceController = require('../controllers/serviceController');
const authMiddleware = require('../middleware/authMiddleware');
const roleMiddleware = require('../middleware/roleMiddleware');
const adminController = require('../controllers/adminController');

router.get('/', authMiddleware, serviceController.getAll);
router.get('/recommended', authMiddleware, serviceController.getRecommended);
router.get('/:id', authMiddleware, serviceController.getOne);
router.post('/', authMiddleware, serviceController.addService);
router.put('/:id/verify', authMiddleware, roleMiddleware('admin'), adminController.verifyService);
router.delete('/:serviceId', authMiddleware, roleMiddleware('admin'), serviceController.deleteService);
router.post('/search-by-tags', authMiddleware, serviceController.findServicesByTags);

module.exports = router;
```

routes/tag.js

```
const express = require('express');
const router = express.Router();
const tagController = require('../controllers/tagController');
const authMiddleware = require('../middleware/authMiddleware');
const roleMiddleware = require('../middleware/roleMiddleware');
```

```
router.get('/', tagController.getAll);
router.post('/', authMiddleware, roleMiddleware('admin'), tagController.addTag);

module.exports = router;
```

routes/user.js

```
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');
const authMiddleware = require('../middleware/authMiddleware');

router.use(authMiddleware);

router.get('/preferences', userController.getPreferences);
router.post('/preferences/create', userController.createPreferences);
router.delete('/preferences/:tagId', userController.deletePreference);
router.get('/saved', userController.getSavedServices);
router.post('/saved', userController.saveService);
router.delete('/saved/:serviceId', userController.deleteSavedService);

module.exports = router;
```

routes/userTag.js

```
const express = require('express');
const router = express.Router();
const authMiddleware = require('../middleware/authMiddleware');
const userTagController = require('../controllers/userTagController');

router.use(authMiddleware);
router.get('/', userTagController.getUserTags);
router.post('/', userTagController.createUserTag);
router.post('/attach', userTagController.addTagToService);
router.post('/detach', userTagController.removeTagFromService);
router.delete('/:tagId', userTagController.deleteUserTag);

module.exports = router;
```

src/api/axiosInstance.js

```
import axios from 'axios';

const axiosInstance = axios.create({
```

```
  baseURL: 'http://localhost:3001/api',
  headers: {
    'Content-Type': 'application/json'
  }
});

axiosInstance.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers['Authorization'] = `Bearer ${token}`;
  }
  return config;
}, error => {
  return Promise.reject(error);
});

export default axiosInstance;
```

src/context/AuthContext.jsx

```
import { createContext, useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(undefined);
  const navigate = useNavigate();

  useEffect(() => {
    const token = localStorage.getItem('token');

    if (token && token !== 'undefined') {
      setUser({});
    } else {
      setUser(null);
    }
  }, []);

  const login = (token) => {
    localStorage.setItem('token', token);
    setUser({});
```

```
    navigate('/home');
  };

  const logout = () => {
    localStorage.removeItem('token');
    setUser(null);
    navigate('/');
  };

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};

export default AuthContext;
```

src/hooks/useAuth.js

```
import { useContext } from 'react';
import AuthContext from '../context/AuthContext';

export const useAuth = () => useContext(AuthContext);
```

src/components/Navbar.jsx

```
import { Link } from 'react-router-dom';
import { useAuth } from '../hooks/useAuth';

const Navbar = () => {
  const { user, logout } = useAuth();

  if (!user) {
    return null;
  }

  return (
    <nav style={navStyle}>
      <div style={leftSideStyle}>
        <Link to="/home" style={linkStyle}>Головна</Link>
        <Link to="/catalog" style={linkStyle}>Каталог</Link>
        <Link to="/saved" style={linkStyle}>Збережені</Link>
      </div>
```

```
      <div style={rightSideStyle}>
        {user.role === 'admin' && (
          <span style={adminTextStyle}>Вітаю, адміністраторе!</span>
        )}
        <Link to="/profile" style={linkStyle}>Профіль</Link>
        <button onClick={logout} style={logoutButtonStyle}>Вийти</button>
      </div>
    </nav>
  );
};

const navStyle = {
  display: 'flex',
  justifyContent: 'space-between',
  padding: '10px 20px',
  borderBottom: '1px solid #ccc',
  alignItems: 'center'
};

const leftSideStyle = {
  display: 'flex',
  gap: '15px',
  alignItems: 'center'
};

const rightSideStyle = {
  display: 'flex',
  gap: '10px',
  alignItems: 'center'
};

const linkStyle = {
  textDecoration: 'none',
  color: '#007bff',
  fontWeight: 'bold'
};

const logoutButtonStyle = {
  backgroundColor: 'red',
  color: 'white',
```

```
  border: 'none',
  padding: '5px 10px',
  cursor: 'pointer',
  borderRadius: '6px',
  transition: 'background-color 0.3s',
};

const adminTextStyle = {
  fontWeight: 'bold',
  color: '#28a745',
  marginRight: '10px'
};

export default Navbar;
```

src/components/PrivateRoute.jsx

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../hooks/useAuth';

const PrivateRoute = ({ children }) => {
  const { user } = useAuth();

  if (user === undefined) {
    return null;
  }

  if (!user) {
    return <Navigate to="/login" />;
  }

  return children;
};

export default PrivateRoute;
```

src/components/SavedServiceCard.jsx

```
import { useState, useEffect } from 'react';
import axiosInstance from '../api/axiosInstance';
import styles from './SavedServiceCard.module.css';

const SavedServiceCard = ({ service, onServiceDeleted }) => {
  const [message, setMessage] = useState('');
```

```
const [userTags, setUserTags] = useState([]);
const [selectedTagId, setSelectedTagId] = useState(service.userTagId || '');
const [currentTagName, setCurrentTagName] = useState('');

useEffect(() => {
  const fetchUserTags = async () => {
    try {
      const response = await axiosInstance.get('/user/tags');
      setUserTags(response.data);

      const tag = response.data.find(tag => tag.id === service.userTagId);
      if (tag) {
        setCurrentTagName(tag.name);
      }
    } catch (error) {
      console.error('Помилка при завантаженні тегів користувача:', error);
    }
  };

  fetchUserTags();
}, [service.userTagId]);



const handleDelete = async () => {
  try {
    await axiosInstance.delete(`/user/saved/${service.id}`);
    showMessage('Сервіс видалено!');
    if (onServiceDeleted) {
      onServiceDeleted(service.id);
    }
  } catch (error) {
    console.error('Помилка при видаленні сервісу:', error);
  }
};

const handleTagSave = async () => {
  try {
    await axiosInstance.post('/user/tags/attach', {
      serviceId: service.id,
      tagId: selectedTagId
```

```
    });
    showMessage('Тег збережено!');
    const selectedTag = userTags.find(tag => tag.id === Number(selectedTagId));
    if (selectedTag) {
      setCurrentTagName(selectedTag.name);
    }
  } catch (error) {
    console.error('Помилка при збереженні тегу:', error);
  }
};

const handleDetachTag = async () => {
  try {
    await axiosInstance.post('/user/tags/detach', {
      serviceId: service.id
    });
    setSelectedTagId('');
    setCurrentTagName('');
    showMessage('Тег відв’язано!');
  } catch (error) {
    console.error('Помилка при відв’язанні тегу:', error);
  }
};

const showMessage = (text) => {
  setMessage(text);
  setTimeout(() => {
    setMessage('');
  }, 2000);
};

const verifiedBadgeStyle = {
  backgroundColor: '#28a745',
  color: 'white',
  padding: '4px 10px',
  borderRadius: '20px',
  fontSize: '12px',
  display: 'inline-block',
  fontWeight: 'bold',
};
```

```
return (
  <div className={styles['saved-service-card']}>
    <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
      <h3 style={{ margin: 0 }}>{service.name}</h3>
      {service.is_verified && <div style={verifiedBadgeStyle}>☆ Перевірено</div>}
    </div>

    {currentTagName && (
      <span className={styles['tag-badge']}>
        {currentTagName}
      </span>
    )}

    <p>{service.description}</p>
    <a href={service.url} target="_blank" rel="noopener noreferrer">
      Перейти на сайт
    </a>

    <select
      className={styles['select-tag']}
      value={selectedTagId}
      onChange={(e) => setSelectedTagId(e.target.value)}
    >
      <option value=""></option>
      {userTags.map(tag => (
        <option key={tag.id} value={tag.id}>
          {tag.name}
        </option>
      ))}
    </select>

    <button onClick={handleTagSave} className={`${styles.button} ${styles['save-button']}`}>
      Зберегти тег
    </button>

    <button onClick={handleDetachTag} className={`${styles.button} ${styles['detach-button']}`}>
      Відв’язати тег
    </button>
```

```
      <button onClick={handleDelete} className={`${styles.button} ${styles['delete-button']}`}>
        Видалити сервіс
      </button>

      {message && <p className={styles.message}>{message}</p>}
    </div>
  );
};

export default SavedServiceCard;
```

src/components/ServiceCard.jsx

```
import axiosInstance from '../api/axiosInstance';
import { useState } from 'react';
import styles from './ServiceCard.module.css';

const ServiceCard = ({ service }) => {
  const [message, setMessage] = useState('');

  const handleSave = async () => {
    try {
      await axiosInstance.post('/user/saved', { serviceId: service.id });
      showMessage('Сервіс збережено!');
    } catch (error) {
      console.error('Помилка при збереженні сервісу:', error);
    }
  };

  const showMessage = (text) => {
    setMessage(text);
    setTimeout(() => {
      setMessage('');
    }, 2000);
  };

  const verifiedBadgeStyle = {
    backgroundColor: '#28a745',
    color: 'white',
    padding: '4px 10px',
    borderRadius: '20px',
    fontSize: '12px',
```

```
    display: 'inline-block',
    fontWeight: 'bold',
  };

  return (
    <div className={styles['service-card']}>
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
        <h3 style={{ margin: 0 }}>{service.name}</h3>
        {service.is_verified && <div style={verifiedBadgeStyle}>☆ Перевірено</div>}
      </div>

      {service.tags && service.tags.length > 0 && (
        <div className={styles['tags-container']}>
          {service.tags.map(tag => (
            <span key={tag.id} className={styles['tag-badge']}>
              {tag.name}
            </span>
          ))}
        </div>
      )}

      <p>{service.description}</p>

      <a href={service.url} target="_blank" rel="noopener noreferrer">
        Перейти на сайт
      </a>

      <button className={styles['save-button']} onClick={handleSave}>
        Зберегти
      </button>

      {message && <p className={styles['message']}>{message}</p>}
    </div>
  );
};

export default ServiceCard;
```

src/pages/NotFoundPage.jsx

```
import { Link } from 'react-router-dom';
```

```
const NotFoundPage = () => {
  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h2>404 - Сторінку не знайдено</h2>
      <p>На жаль, такої сторінки не існує.</p>
      <Link to="/">
        <button style={{
          marginTop: '20px',
          padding: '10px 20px',
          border: 'none',
          backgroundColor: '#007bff',
          color: 'white',
          cursor: 'pointer'
        }}>
          Повернутися на головну
        </button>
      </Link>
    </div>
  );
};

export default NotFoundPage;
```