Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

# МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна система керування контентом та користувацькими комунікаціями»

Виконав: студент 2-го курсу, групи 1ПІ-22м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Івасьов О.С.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ:
Коваленко О.О.
(прізвище та ініціали)
«15» грудня 2023 р.

Опонент: к.т.н., доц. каф. ЗІ:
Лукічов В.В.
(прізвище та ініціали)
«15» грудня 2023 р.

**Допущено до захисту**
Завідувач кафедри ПЗ
д.т.н. проф. Романюк О.Н.
(прізвище та ініціали)
«15» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О.Н.

«19» вересня 2023 р.

**З А В Д А Н Н Я**
**НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙУ РОБОТУ СТУДЕНТУ**

Івасьову Олексію Станіславовичу

1. Тема роботи – Інформаційна система керування контентом та користувацькими комунікаціями

Керівник роботи: Коваленко Олена Олексіївна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023 року №247

2. Строк подання студентом роботи

5 грудня 2023 року

3. Вихідні дані до роботи: середовища розробки Visual Studio 2019 та Visual Studio Code, мови розробки JavaScript, операційна система – Windows 10, правила інформаційної екосистеми, методи автоматизації управлінських процесів, інформаційні системи управління персоналом.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; аналіз платформ управління персоналом; методи та моделі автоматизації

управлінських процесів; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: блок-схеми алгоритмів роботи додатку; структура нейронної мережі; тестування додатку.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада Консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1-4 | Коваленко О.О.., к.т.н., доцент. кафедри ПЗ | 19.09.2023 | 05.12.2023 |
| 5 | Причепа І.В. к.е.н., доцент кафедри ЕПВМ | 25.11.2023 | 01.12.2023 |

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

| з/п | Назва етапів магістерської кваліфікаційної Роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі | 17.09.2023 – 30.09.2023 | вик |
| 2 | Розробка структури та методів програмного продукту | 01.10.2023 – 10.10.2023 | вик |
| 3 | Реалізація програми інформаційної системи керування контентом | 11.10.2023 – 25.10.2023 | вик |
| 4 | Тестування програмного застосунку | 26.10.2023 – 20.11.2023 | вик |
| 5 | Економічна частина | 21.11.2023 – 08.12.2023 | вик |

Студент _____ Івасьов О.С.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Коваленко О.О.
(підпис) (прізвище та ініціали)

## Анотація

УДК 004. 005:96

Івасьов О.С. Інформаційна система керування контентом та користувацькими комунікаціями. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 118 с.

На укр. мові. Бібліогр.: 35 назв; рисунків: 34; таблиць 11.

У магістерській кваліфікаційній роботі проведено детальний аналіз існуючих інформаційних систем. Проведено аналіз аналогів, визначено їх переваги та недоліки. На основі досліджень було прийняте рішення розробити власний застосунок з врахуванням недоліків аналогів. Сформульовано мету досліджень – підвищення ефективності використання інформаційної системи керування контентом та користувацькими комунікаціями в організації.

Розроблено методи створення колективного інформаційного середовища та програмна реалізація, яка дозволяє організувати процес розробки програмного продукту що значно полегшує роботу адміністраторів та користувачів. Застосовані сучасні технології для забезпечення ефективного збору, обробки та відображення різноманітного контенту на веб-сайті чи в інших інтерактивних середовищах.

Створений програмний продукт написаний на мові програмування JavaScript з використанням фреймворку Angular. Характеризується швидкістю та надійністю роботи, а також широким спектром можливих застосувань.

Отримані в магістерській кваліфікаційній роботі результати можна використати для створення інформаційної системи керуванням контенту.

Ключові слова: інформаційна система керування контентом, редагування контенту, методи розробки інформаційної системи.

## Abstract

Ivasos O.S. Information system for managing content and user communications. Master's thesis on the specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. 118 p.

Ukrainian language. Bibliography: 35 titles; drawings: 34; Tables 11.

The master’s thesis contains a detailed analysis of existing information systems. An analysis of analogues was carried out, their advantages and disadvantages were determined. Based on research, a decision was made to develop our own application, taking into account the shortcomings of analogues. The purpose of the research is formulated - to increase the efficiency of the use of the information system for managing content and user communications in the organization.

Methods of creating a collective information environment and software implementation have been developed, which allows you to organize the process of developing a software product, which greatly facilitates the work of administrators and users. Modern technologies are used to ensure efficient collection, processing and display of various content on the website or in other interactive environments.

The created software product is written in the JavaScript programming language using the Angular framework. It is characterized by speed and reliability of operation, as well as a wide range of possible applications.

The results obtained in the master's qualification work can be used to create an information system for content management.



Keywords: content management information system, content editing, information system development methods.

# ЗМІСТ

# ВСТУП

**Обґрунтування вибору теми дослідження.** Сучасний інформаційний вік вимагає від підприємств та організацій високофункціональних інформаційних систем, спрямованих на ефективне управління контентом та користувацькими комунікаціями. Швидкий розвиток технологій та зростання обсягу інформації ставлять перед сучасними організаціями завдання забезпечення не лише надійного зберігання даних, але й активного управління ними для досягнення стратегічних цілей.

Інформаційні системи керування контентом та користувацькими комунікаціями відіграють ключову роль у формуванні іміджу організації, взаємодії зі зацікавленими сторонами та забезпеченні ефективної внутрішньої комунікації. Застосування сучасних технологій у цих системах може значно поліпшити якість комунікації та сприяти підвищенню продуктивності діяльності організації.

Враховуючи постійні зміни у сфері інформаційних технологій, важливо проводити наукові дослідження для розробки та впровадження новітніх рішень в області керування контентом та комунікаціями. Дана дипломна робота спрямована на аналіз сучасних тенденцій, визначення найефективніших підходів та розробку інформаційної системи, що відповідає вимогам сучасного бізнесу та сприяє підвищенню конкурентоспроможності організації.

Результати даного дослідження передбачається використовувати для розробки та впровадження інформаційної системи керування контентом та користувацькими комунікаціями в конкретній організації. Практична реалізація розробленої системи покликана покращити ефективність управління інформацією, сприяти взаємодії з клієнтами та підвищувати рівень задоволеності користувачів.

Таким чином, розробка інформаційної системи керування контентом та користувацькими комунікаціями обумовлена актуальністю проблеми,

необхідністю впровадження сучасних технологій у сферу управління інформацією та має велике практичне значення для подальшого розвитку організаційного інформаційного простору.

**Мета та завдання дослідження.** Метою роботи є підвищення ефективності використання інформаційної системи керування контентом та користувацькими комунікаціями в організації. Для досягнення цієї мети передбачається вирішення наступних завдань:

- розгляд сучасних тенденцій у сфері керування контентом та користувацькими комунікаціями.

- запропонувати методи покращення функціоналу інформаційної системи. Це включає в себе розробку нових можливостей, які сприятимуть зручності користування системою, підвищенню її продуктивності та забезпеченню більш ефективної взаємодії з користувачами.

- розробка інформаційної системи керування контентом та користувацькими комунікаціями.

**Об'єкт дослідження** – процеси керуванням контентом.

**Предмет дослідження** – методи та програмні засоби керуванням контенту.

**Методи дослідження.** У процесі дослідження використовувались: теорія систем для здійснення системного аналізу; метод аналізу для визначення вимог користувачів, моделювання для формування моделей за методологією DFD (діаграм потоків даних), моделей взаємодії за допомогою UML, моделі прийняття управлінських рішень на основі аналізу вартості-ефективності; моделі життєвого циклу розробки програмного забезпечення Agile, моделей тестування та верифікації; теорія алгоритмів – для формування алгоритмів створення програмних модулей системи керування контентом.

**Наукова новизна одержаних результатів.**

- отримав подальшого розвитку метод інтеграції інформаційної системи з соцмережами, який, на відміну від існуючих, дозволяє не лише авторизуватись через соціальні мережі, а також обмінюватись контентом між

системою та соціальними профілями користувачів, що сприяє покращенню показників залучення аудиторії.

- отримав подальшого розвитку метод аналізу й відстеження користувацької активності, який, на відміну від існуючих, об’єднує в собі метод рекомендацій на основі вмісту та метод спільної фільтрації, що дозволяє оптимізувати персоналізацію системи за визначеними параметрами для кращої взаємодії з користувачами за цільовими показниками відповідності контенту.

**Практичне значення одержаних результатів.** Практична цінність одержаних результатів полягає у можливості практичного використання розробленої спеціалізованої інформаційної системи.

**Особистий внесок.** Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, автору належать: розробка інформаційної системи керування контентом та користувацькими комунікаціями та пропозиції та способи покращення інформаційних систем керування контентом.

**Апробація матеріалів роботи.** Матеріали роботи доповідалися та обговорювалися на науково-технічній міжнародній онлайн-конференції «МІСЦЕ УКРАЇНИ У СВІТОВОМУ РОЗВИТКУ НАУКИ ТА ТЕХНІКИ» м.Хмельницький та на всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету.

**Публікації.** Основні результати дослідження опубліковані в науковій роботі – тезах-доповіді на міжнародній онлайн-конференції «МІСЦЕ УКРАЇНИ У СВІТОВОМУ РОЗВИТКУ НАУКИ ТА ТЕХНІКИ» м.Хмельницький та на всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету.

# 1 . АНАЛІЗ ПРОБЛЕМИ, ОБҐРУНТУВАННЯ АКТУАЛЬНОСТІ РОЗРОБКИ СИСТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Обґрунтування доцільності створення інформаційної системи керування контентом та користувацькими комунікаціями

У сучасному цифровому віці, де інтернет та інформаційні технології перетворилися на невід'ємну частину практично всіх сфер життя, інформаційні системи керування контентом та користувацькими комунікаціями стали важливим інструментом для бізнесу та організацій. Обґрунтування доцільності створення і використання системи базується на декількох ключових аспектах.

По-перше, інформаційна система керування контентом дозволяє ефективно та організовано збирати, зберігати та оновлювати контент. Завдяки цьому, користувачі можуть легко створювати, редагувати та публікувати контент без необхідності глибоких знань в галузі програмування. Це зменшує залежність від IT-спеціалістів і сприяє швидкому впровадженню змін в онлайн-середовищі [1].

По-друге, CMS дозволяє забезпечити єдність та стандартизацію контенту. Це особливо важливо для компаній, які мають різноманітні канали розповсюдження інформації (веб-сайт, соціальні мережі, електронна пошта тощо). Система гарантує, що інформація виглядає однаково та має однорідний стиль, підвищуючи впізнаваність бренду та покращуючи користувацький досвід.

По-третє, інформаційні системи керування контентом допомагають у вирішенні проблеми безпеки та контролю доступу до інформації. Забезпечуючи права доступу для користувачів на різних рівнях, CMS дозволяє обмежити доступ до конфіденційної інформації, збільшуючи рівень безпеки.

По-четверте, створення інформаційної системи керування контентом є ключовим елементом стратегії цифрової трансформації. У сучасному світі, де

швидкість змін в інформаційній сфері надзвичайно велика, CMS дозволяє організаціям бути більш гнучкими та адаптивними до змін у вимогах ринку та споживачів. Ця гнучкість дозволяє швидше реагувати на нові можливості та ефективно впроваджувати стратегічні ініціативи [2].

По-п'яте, інформаційна система керування контентом спрощує процес співпраці та комунікації в межах організації. Вона надає можливість розподіленого редагування, обговорення та затвердження контенту, що полегшує спільну роботу різних відділів та команд. Такий підхід збільшує продуктивність та допомагає уникнути зайвих помилок чи неоднозначностей у контенті.

Зрештою, важливим фактором є те, що споживачі використовують соціальні медіа для обміну інформацією та взаємодії з брендами. Інформаційні системи  надають можливість інтегрувати контент з соціальних медіа, що спрощує спілкування з аудиторією та відстеження реакції користувачів на контент [3].

Незважаючи на багато позитивних аспектів, інформаційні системи CMS також стикаються з рядом викликів та проблем:

З великою кількістю контенту, створюваного користувачами, можуть виникнути проблеми з якістю та достовірністю інформації. Споживачі мають справедливі очікування щодо якості та достовірності контенту, а отже, CMS повинні мати системи перевірки та модерації контенту.

Зростаючі загрози безпеці вимагають постійного оновлення заходів захисту інформаційних систем. Атаки на веб-сайти, включаючи хакерські атаки та розповсюдження шкідливих програм, можуть завдати серйозної шкоди як веб-сайту, так і його користувачам.

Швидкість завантаження веб-сторінок грає важливу роль у задоволенні користувачів. Завантаження повільної сторінки може вплинути на зниження конверсії та задоволення користувачів. Оптимізація швидкості завантаження стає ключовим завданням для розробників CMS.

Для глобальних організацій важливо мати систему керування

контентом, яка підтримує різні мови та культурні особливості. Це включає в себе можливість перекладу контенту, використання різних форматів дат та часу, а також підтримку міжнародних стандартів.

З огляду на ті проблеми та виклики, з якими стикаються інформаційні системи керування контентом, існують різні можливості для їх покращення та інновацій:

Використання штучного інтелекту для персоналізації контенту, прогнозування потреб користувачів та аналізу даних може покращити ефективність CMS. Штучний інтелект може допомогти автоматизувати процеси розподілу контенту та рекомендацій для користувачів.

Подальша розробка мобільних версій CMS та підтримка адаптивного дизайну допоможе впровадити контент на мобільних пристроях. Це включає в себе оптимізацію для різних типів пристроїв, розмірів екранів та орієнтацій.

Використання аналітики великих даних для розуміння користувачів та їхніх потреб може покращити здатність інформаційної системи до персоналізації контенту та пропозицій.

Розширення набору плагінів та додатків для CMS надасть користувачам більше можливостей для розширення функціональності. Зростання кількості доступних плагінів може значно спростити інтеграцію різноманітних інструментів та сервісів у веб-проекти [4].

Інформаційна система керування контентом та користувацькими комунікаціями включає в себе ряд завдань та елементів, які необхідно розробити для забезпечення її ефективної функціональності. Завдання перед інформаційною системою включають:

- створення та редагування контенту: система повинна надавати можливість створення, редагування та форматування текстового, графічного та мультимедійного контенту;
- категоризація та індексація контенту: система повинна дозволяти організувати контент за категоріями, мітками та індексувати його для полегшення пошуку та навігації;

- управління версіями контенту: має забезпечуватись можливість ведення історії версій та відновлення попередніх версій контенту;
- забезпечення інтерактивності: CMS повинна дозволяти взаємодію користувачів з контентом, включаючи можливість коментування, обговорення, лайків, рейтингів тощо;
- персоналізація контенту: Система повинна надавати засоби персоналізації контенту для різних користувачів з урахуванням їхніх інтересів та попередньої активності;
- ведення статистики та аналітика: CMS повинна забезпечувати збір та аналіз даних про користувачів, їхню взаємодію з контентом та ефективність комунікаційних стратегій;
- автентифікація та авторизація: Система повинна забезпечувати безпеку користувачів шляхом вимоги аутентифікації та обмеження доступу до функціональності в залежності від рівня авторизації;
- резервне копіювання та відновлення: Забезпечення можливості резервного копіювання контенту та системи для захисту від можливих втрат даних;
- модульність: CMS повинна мати архітектуру, що дозволяє додавати та розширювати функціональність за допомогою модулів та плагінів;
- налаштовуваність: Система повинна надавати можливість налаштовувати вигляд та функціональність відповідно до потреб конкретного користувача чи організації;
- моніторинг та логування: CMS повинна забезпечувати інструменти для моніторингу та ведення журналів подій для виявлення помилок та аномалій [5].

Отже, загальною тенденцією є постійний розвиток інформаційних систем та їх адаптація до зростаючих потреб користувачів і вимог безпеки. Аналіз стану питання виявив, що існує багато можливостей для покращення та інновацій у цій сфері, що важливо враховувати під час розробки інформаційної системи керування контентом та користувацькими комунікаціями.

## 1.2 Порівняльний аналіз аналогів

Аналіз інформаційних систем керування контентом та користувацькими комунікаціями є ключовим етапом при визначенні оптимального рішення для створення чи удосконалення інформаційної системи. На ринку існує безліч різних інструментів та платформ, які пропонують різний функціонал та можливості. Розглянемо чотири з найбільш відомих та використовуваних інформаційних систем керування контентом та користувацькими комунікаціями, проведемо їхній порівняльний аналіз, висвітлимо їхні переваги та недоліки.

WordPress — це одна з найпопулярніших та найвикористовуваніших платформ для управління контентом та створення веб-сайтів [6]. Вона вперше була випущена у 2003 році і з того часу здобула широкий попит серед користувачів, починаючи від блогерів і закінчуючи великими корпораціями. WordPress є вільним і відкритим програмним забезпеченням, що означає, що систему можна використовувати безкоштовно і редагувати вихідний код під свої потреби. На рисунку 1.1 подано головне вікно системи.

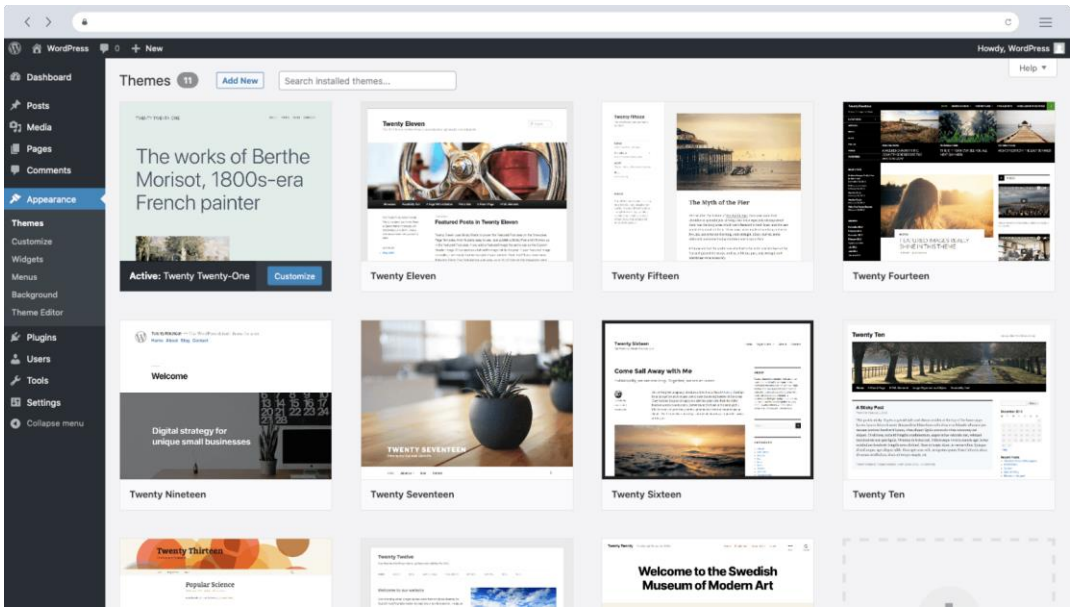


Рисунок 1.1 – Вікно екрану системи Wordpress

WordPress приваблює користувачів своєю простотою використання,

багатофункціональністю та безкоштовністю, що дозволяє створювати різноманітні типи веб-сайтів, від особистих блогів і невеликих підприємств до корпоративних веб-порталів та електронних магазинів. Основною мовою розробки WordPress є PHP, а база даних використовує MySQL.

Перевагами системи є простота використання, можливість розширити функціонал веб-сайту за допомогою безкоштовних та платних розширень та велика кількість користувачів та розробників, які готові надавати підтримку та допомогу. WordPress підходить навіть для не-технічних користувачів, завдяки зручному інтерфейсу.

WordPress визначається своєрідним підходом до керування контентом, де використовується система плагінів для розширення функціональності. Це дозволяє користувачам адаптувати платформу до своїх конкретних потреб, використовуючи різноманітні розширення для SEO, аналітики, соціальних мереж та інших аспектів.

Однією з основних переваг WordPress є його висока ступінь доступності, навіть для тих, хто не має глибоких технічних знань. Інтерфейс користувача дружелюбний та інтуїтивно зрозумілий, що дозволяє широкому колу користувачів створювати та редагувати вміст без необхідності програмування.

Важливою рисою є широкий вибір тем та дизайнів, що полегшує персоналізацію веб-сайту. Це дозволяє власникам підприємств створювати унікальний образ бренду та привертати увагу цільової аудиторії.

Однак, з усією своєю популярністю, WordPress також має свої обмеження. Зокрема, підвищена вразливість до кібератак та потреба регулярного оновлення з метою забезпечення безпеки можуть стати проблемними для менеджерів веб-сайту.

Порівнюючи із створенням інформаційної системи "з нуля", WordPress забезпечує швидше розгортання веб-сайтів, але водночас вносить обмеження щодо гнучкості та управління великим обсягом інформації. Багато компаній виявляють необхідність у великих витратах на додаткові плагіни або в подальшому переходять до інших інструментів.

У підсумку, WordPress відзначається легкістю використання та доступністю, але обмежується у гнучкості та можливостях вирішення більш складних завдань. Його застосування може бути ефективним для невеликих та середніх підприємств, але при високих вимогах до безпеки та гнучкості може виникнути необхідність розглядати більш розширені рішення.

Drupal – це інформаційна система, спеціалізована на розробці складних веб-проектів та корпоративних порталів [7]. Drupal – це потужна та відкрита система управління контентом, яка здатна відповісти на різноманітні потреби веб-розробників і власників веб-сайтів. Вона була вперше випущена в 2000 році і вже пройшла шлях розвитку, ставши однією з найпопулярніших CMS у світі. Drupal підтримує спільні мови програмування, такі як PHP, і використовує MySQL або інші системи управління базами даних. Однією з визначальних рис Drupal є його модульна архітектура.

Велика кількість модулів дозволяє користувачам налаштовувати свої веб-сайти під конкретні потреби. Модулі можуть додавати функціональність, таку як аналітика, соціальні мережі, електронна комерція і багато іншого. Це робить Drupal дуже гнучкою для різних видів проєктів.

Drupal призначений для роботи як для невеликих веб-сайтів, так і для великих та складних проектів. Його гнучкість дозволяє розробникам та адміністраторам вибирати та налаштовувати функції залежно від конкретних вимог.

Drupal надає широкий набір можливостей для управління вмістом, створення блогів, форумів, анкет, галерей зображень і багато іншого. Він також підтримує створення власних типів контенту, що дає змогу адаптувати платформу під конкретні потреби.

Як і у випадку з WordPress, Drupal користується активною спільнотою розробників та користувачів. Це означає, що ви можете знайти велику кількість допомоги, ресурсів та плагінів, щоб поліпшити та налаштувати свій веб-сайт.

Drupal приділяє значну увагу безпеці, і регулярні оновлення та патчі

допомагають у забезпеченні найвищого рівня безпеки для користувачів. Це особливо важливо для бізнес-сайтів та інших проектів, де захист особистих даних є критичним аспектом.

Drupal надає інструменти для легкої інтернаціоналізації (створення мультиязикових сайтів) та локалізації (адаптація до різних мов та регіонів), що робить його привабливим для глобальних проектів.

Також Drupal надає потужну систему керування правами доступу, що дозволяє налаштовувати рівні доступу до контенту і функціональності для різних користувачів і груп. Хоча Drupal є потужним інструментом, його навчання та конфігурація може виявитися більш складними порівняно з іншими CMS, такими як WordPress. Він також може бути більш важким для менеджерів веб-сайтів, які не мають технічного досвіду. У той же час, для складних та великих проектів, де вимагається велика гнучкість, масштабованість та безпека, Drupal може бути відмінним вибором. На рисунку 1.2 подано головне вікно системи.

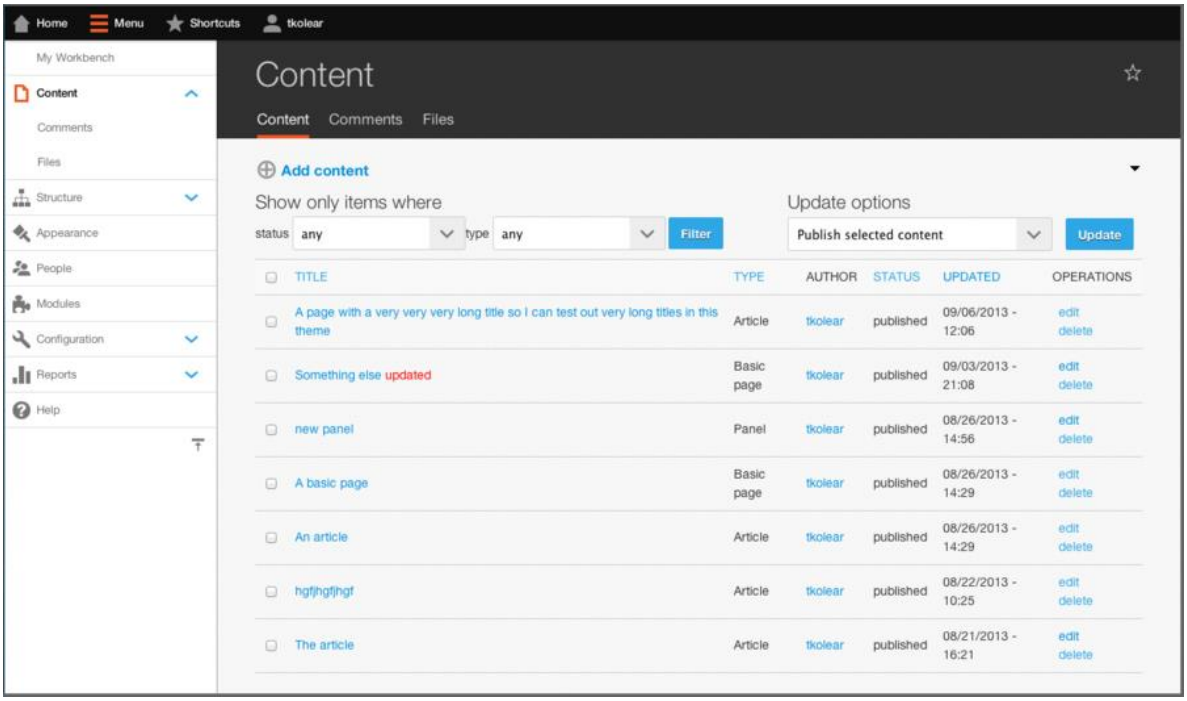


Рисунок 1.2 – Вікно екрану системи Drupal

Основними перевагами системи є гнучкість, високий рівень безпеки та

широкі можливості для налаштування доступу та можливість додавати функціонал через модулі.

Серед недоліків системи можна виділити: Обмежена кількість готових тем і плагінів порівняно з WordPress та вимоги глибоких знань програмування та налаштування.

Joomla – це відкрита система управління контентом, яка використовується для створення і управління веб-сайтами [8]. Вона була вперше випущена в 2005 році і стала популярною завдяки своїй простоті в використанні і розширюваності. Joomla використовує мову програмування PHP і систему управління базами даних MySQL.

Joomla володіє привабливим інтерфейсом та різноманітними темами, що дозволяють вам швидко налаштовувати зовнішній вигляд веб-сайту. Його адміністративний інтерфейс інтуїтивно зрозумілий, що полегшує роботу з платформою навіть для новачків.

Як і в інших відкритих системах, Joomla має активну та допоміжну спільноту. Це означає, що ви можете отримати допомогу, обговорювати питання та знаходити рішення через форуми та інші ресурси.

Однією з сильних сторін Joomla є її можливість працювати в режимі багатьох мов, що дозволяє створювати мультиязикові веб-сайти без зайвих зусиль.

Joomla приділяє велику увагу безпеці, і регулярні оновлення та патчі допомагають забезпечити найвищий рівень безпеки для користувачів та їх даних. Joomla відзначається високою продуктивністю та відмінною роботою на сервері. Це може бути важливим для великих та високонавантажених веб-сайтів, де ефективна робота системи грає ключову роль.

Основною особливістю Joomla є її висока розширюваність. Вона пропонує велику кількість розширень, таких як модулі, компоненти та шаблони, які дозволяють розширити функціональність веб-сайту за ваших потреб. Це робить Joomla відмінним вибором для різних видів веб-проєктів, від особистих блогів до корпоративних портфоліо та інтернет-магазинів. На

рисунку 1.3 подано головне вікно системи.

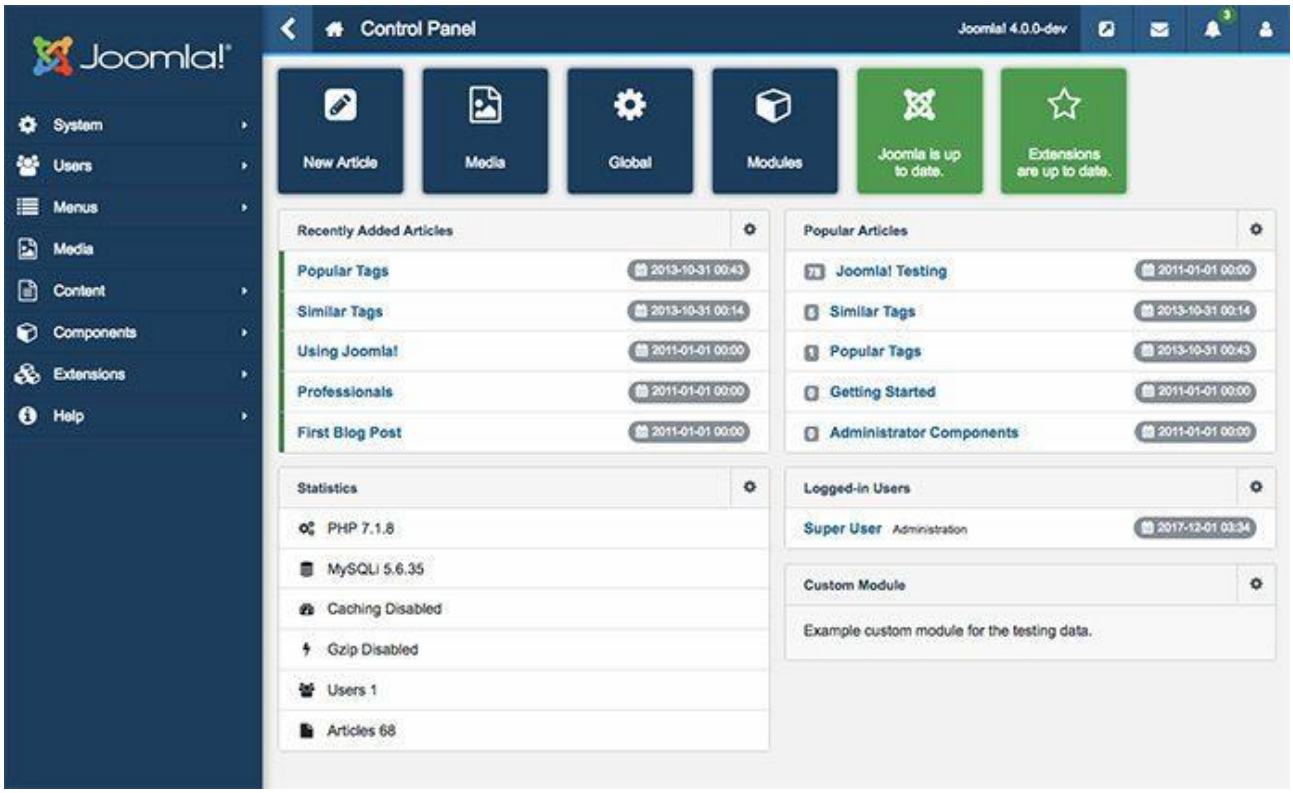


Рисунок 1.3 – Вікно екрану системи Joomla

Joomla також має потужну систему керування правами доступу, що дозволяє налаштовувати рівні доступу для різних користувачів і груп. Це особливо важливо для сайтів, де необхідно обмежувати доступ до певного контенту.

Однією з переваг Joomla є активна спільнота користувачів і розробників, яка надає підтримку та розвиток платформи. Ви можете знайти велику кількість документації, форумів та розширень, щоб полегшити процес розробки і підтримки веб-сайту на Joomla.

Недоліками Joomla є те, що вона може бути менш простою для новачків порівняно з іншими CMS, і вона може вимагати більше часу на налагодження і налаштування. Також, оскільки Joomla не має вбудованої системи оптимізації для пошукових систем, ви повинні додатково налаштовувати SEO-плагіни.

Хоча Joomla виявляється дуже привабливою для користувачів, які шукають легку у використанні та гнучку CMS, вона може бути менш гнучкою та менш адаптованою для деяких завдань, порівняно з іншими CMS. Однак, з урахуванням її дизайну та легкості використання, Joomla залишається важливим гравцем на полі систем управління контентом.

Magento — це потужна відкрита система управління електронною комерцією (e-commerce CMS), яка розроблена спеціально для створення і управління інтернет-магазинами та іншими онлайн-торговими платформами [9]. Magento визнана однією з провідних платформ для електронної комерції завдяки своїй функціональності та розширюваності.

Magento відомий своєю гнучкістю та розширюваністю. Він дозволяє створювати високофункціональні магазини, пристосовані до різних видів бізнесу. Розширення та модулі Magento дозволяють легко налаштовувати та розширювати функціональність відповідно до унікальних потреб користувача.

CMS призначена для великих магазинів з великою кількістю товарів та транзакцій. Він володіє вражаючою продуктивністю, що дозволяє обробляти значні обсяги даних та забезпечувати стабільну роботу сайту під час пікових навантажень.

Система надає ряд інструментів для SEO-оптимізації, що допомагає покращити видимість вашого магазину в пошукових системах. Його структура URL, мета-теги та інші SEO-елементи можуть легко налаштовуватися для оптимізації позицій в пошукових результатів.

Також вона підтримує створення маркетплейсів та багато-вендорних магазинів. Це дозволяє різним продавцям продавати свої товари через один і той самий магазин, розширюючи асортимент та привертаючи різноманіття виробників.

Magento надає можливості для мобільної оптимізації, що важливо в умовах зростання мобільних покупок. Ви можете створювати мобільні версії сайту або використовувати респонсивний дизайн для забезпечення оптимального відображення на різних пристроях.

CMS має велику та активну спільноту розробників та користувачів. Це забезпечує доступність різних ресурсів, форумів та документації для отримання допомоги та вирішення питань.

Magento включає вбудовані інструменти аналітики та звітності, які допомагають вам відстежувати різні аспекти продажів, поведінки покупців та інші ключові показники.

На рисунку 1.4 подано головне вікно системи.

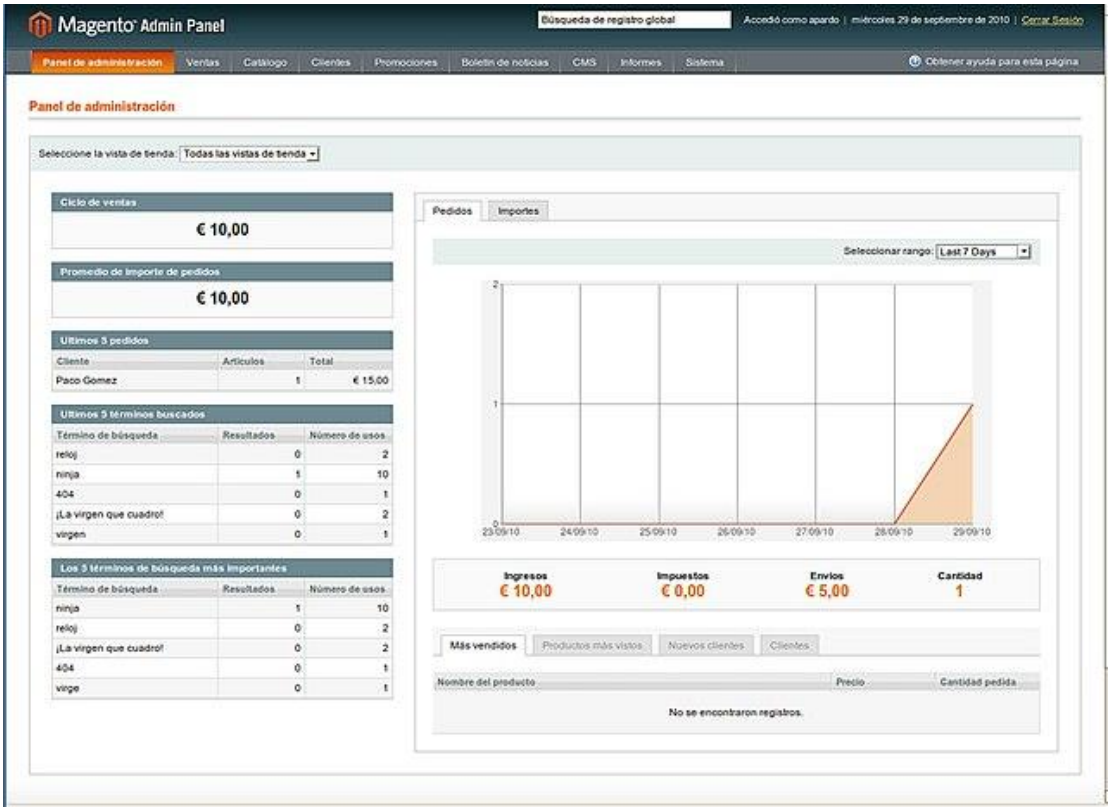


Рисунок 1.4 – Вікно екрану системи Magento

Основною перевагою Magento є її здатність надавати розширений набір функцій для створення інтернет-магазинів практично будь-якого розміру і складності. Платформа має велику кількість розширень, тем оформлення і модулів, які дозволяють налаштовувати інтернет-магазин згідно з унікальними потребами бізнесу. Magento підтримує багато мов і валют, що дозволяє розширити аудиторію інтернет-магазину на глобальному рівні.

Ще однією важливою перевагою Magento є її потужна система

управління продуктами і інвентарем, що дозволяє легко додавати, оновлювати та видаляти товари, а також вести облік запасів. Також Magento надає аналітичні засоби, які допомагають власникам магазинів аналізувати та вдосконалювати їхні торгові стратегії.

Недоліками системи є те, що вона може бути більш складною для налаштування та використання в порівнянні з іншими платформами для електронної комерції, і вона може вимагати додаткових зусиль для розгортання та підтримки. Також, Magento - це важка система, і вона може потребувати потужного хостингу та ресурсів сервера.

Хоча Magento володіє вражаючою функціональністю, варто зазначити, що він може бути складним у використанні та вимагає певних технічних знань для ефективної настройки та управління. Також, у порівнянні з іншими CMS, розгортання Magento може вимагати більше ресурсів. Однак, для великих та складних проектів у сфері електронної комерції, Magento залишається однією з найпотужніших та найефективніших платформ.

Проаналізувавши усі аналоги, було визначено їх сильні та слабкі місця та створено порівняльну таблицю зі власним додатком (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

| Критерій | WordPress | Drupal | Joomla | Magento | Власний додаток |
|---|---|---|---|---|---|
| Створення та редагування контенту | + | + | + | - | + |
| Можливості електронної комерції | - | - | - | + | + |
| Управління користувачами і ролями | + | + | + | - | + |
| Розширені можливості SEO | + | + | + | - | + |

Продовження таблиці 1.1

| Критерій | WordPress | Drupal | Joomla | Magento | Власний додаток |
|---|---|---|---|---|---|
| Модульність та розширюваність | + | + | + | + | + |
| Аналітика та звітність | + | + | + | - | + |
| Можливості роботи з великим обсягом даних | - | + | - | + | + |
| Наявність користувацьких комунікацій | + | + | + | - | + |
| Підсумковий результат | 6 | 7 | 6 | 3 | 8 |

Таблиця порівняльних характеристик показала, що розробка програмного продукту є доцільною. В результаті отримаємо продукт, що покриває недоліки існуючих рішень та забезпечує кращу ефективність.

## 1.4 Моделі персоналізації контенту для покращення ефективності інформаційної системи керування контентом

Що таке персоналізація? Персоналізація, згідно із Кембриджським словником, – це процес створення індивідуалізованого вмісту для задоволення конкретних потреб особи.

У цьому визначенні ми виокремлюємо три ключові аспекти:

- Процес (вірний час і канал);
- Підходить для потреб (правильне повідомлення)
- Конкретна особа (правильна людина)

Отже, персоналізація – це постійні зусилля з оптимізації взаємодії з клієнтом. Мета – передати потрібне повідомлення через відповідний канал до конкретної людини в потрібний час.

Правильне повідомлення: Персоналізація – це не загальні листи чи

повідомлення для всіх, а адаптація до кожної людини. Тут мова йде про здатність адаптувати повідомлення до конкретної особи.

Правильний канал: Персоналізація означає вибір відповідного каналу для кожного клієнта. Це спонукає до взаємодії з компанією і максимізує ефективність маркетингових інвестицій.

Правильна особа: Персоналізація - це доставка повідомлення конкретній особі. Вимагає розуміння їхніх потреб та адаптації продукту та маркетингу відповідно.

Відповідний час: Загалом, персоналізація полягає в визначенні оптимального часу для взаємодії з клієнтами. Це може бути на будь-якому етапі маркетингового процесу – від залучення нових клієнтів до відправлення знижкового коду [10].

Не секрет, що людей приваблює релевантний для них контент. Ідея веб-персоналізації полягає в тому, що якщо ваш сайт може представити інформацію про свої продукти чи послуги в індивідуальній манері на основі того, до чого користувач уже виявляв інтерес, то він, швидше за все, залишиться й подивиться, що ще у вас є для нього . Персоналізація застосунку дає клієнтам відчуття, що їх цінують, і показує їм, що про їхні потреби дбають.

Персоналізацію веб-вмісту можна здійснити, адаптувавши текстові повідомлення, електронні листи, сторінки веб-сайтів, публікації в соціальних мережах і рекламу відповідно до вподобань клієнта. Це також означає збір даних від відвідувачів, щоб знати, хто вони і коли їм потрібен певний вміст.

Персоналізація – це головний фактор, який виділяє веб-сайт або бізнес серед інших. У наш час, недостатньо мати хороший продукт за справедливою ціною. Ви повинні дати клієнтам те, що вони шукають, те, що відповідає їхнім потребам, те, що змушує їх відчувати себе особливими. Персоналізований маркетинг допомагає продавати більше продуктів, зменшувати витрати та збільшувати прибуток. Він надає вам глибокі дані про вподобання ваших клієнтів, щоб допомогти вам покращити свої послуги чи продукти. І це допоможе вам побудувати міцні стосунки зі своїми клієнтами, дізнавшись їх краще. Ось

чому 60% споживачів кажуть, що вони стануть постійними покупцями після персоналізованого досвіду покупок.

Кілька головних переваг персоналізації:

– Відповідні рекомендації щодо продуктів;
– Краще розуміння клієнта;
– Вищі коефіцієнти конверсії та вища цінність клієнта;
– Покращена лояльність клієнтів.

Персоналізацію можна поділити на декілька етапів, які використовують різні моделі взаємодії з користувачем: one to all, one to many, one to some, one to few, one to one. Чим ширша аудиторія, тим загальнішим і менш складним буде спосіб пропозиції, і навпаки. Модель персоналізації графічно зображено на рисунку 1.5.


Рисунок 1.5 – Модель персоналізації

Розберемо кожен з етапів. Один до всіх, початковий етап персоналізації. Початковим етапом персоналізації є збір базових даних. Чим більше інформації ви маєте про своїх користувачів, тим більша ймовірність, що ви зможете запропонувати їм щось доречне. Основна мета цього етапу — зібрати дані про ваших клієнтів, щоб потім ви могли націлити на них вміст, який вони вважатимуть релевантним і привабливим.

Другим етапом персоналізації є етап сегментації або один до багатьох. На цьому етапі компанії починають створювати сегменти клієнтів на основі даних, які вони зібрали раніше. Якщо ви зібрали достатньо даних про своїх клієнтів, швидше за все, ви зможете визначити їхні потреби на основі географічних і демографічних факторів.

Наступний етап — один до деяких, це етап персоніфікації, на якому ви створюєте глибші сегменти клієнтів із поведінковими моделями. До цього етапу ви визначили своїх клієнтів. Наступний крок — розділити їх на більше груп. Це допоможе вам зрозуміти, чим вони схожі, а також чим відрізняються. Дізнавшись це, ви зможете надати клієнтам більш персоналізований досвід.

Четвертий етап у моделі зрілості персоналізації – один до кількох, це етап мікросегментації. На цьому етапі сегментація клієнтів йде ще глибше.. Крім того, до цього етапу це була в основному персоналізація на основі правил. З цього етапу компанії починають використовувати прогнозну персоналізацію разом із персоналізацією на основі правил.

Останній етап персоналізації – один до одного. На цьому етапі бренди намагаються створити індивідуальний досвід спілкування з клієнтами. На цьому етапі компанії отримують вигоду від усіх точок даних по всіх каналах і створюють єдиний профіль для кожного клієнта. Етапи персоналізації та їх опис зображено на рисунку 1.6

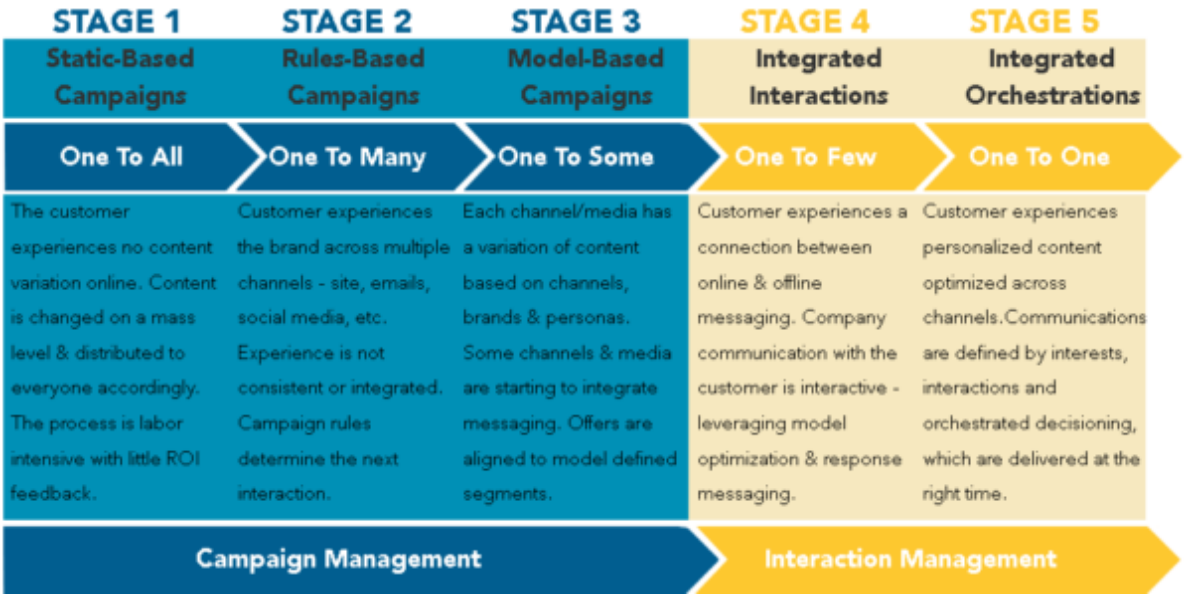


Рисунок 1.6 – Етапи персоналізації

Також додатково була створена піраміда персоналізованого досвіду (Рис. 1.7), згідно з якою спочатку ми проводимо збір даних про користувача, виконуємо сегментацію по інтересам і далі робимо йому найбільш влучну пропозицію.

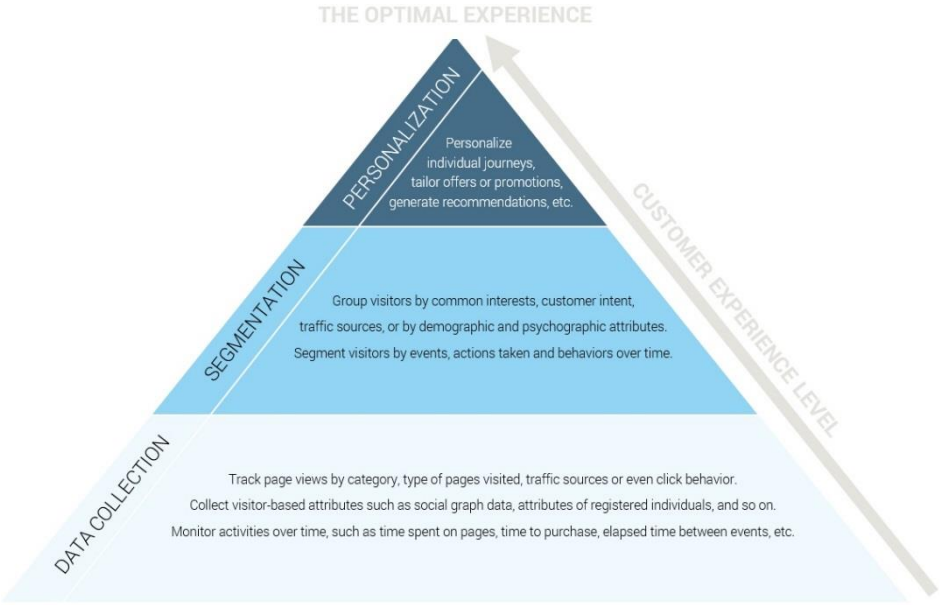


Рисунок 1.7 – Піраміда персоналізованого досвіду

Отже, реалізація всіх етапів та використання піраміди дозволяє створити найкращу прогнозну персоналізацію для своїх клієнтів.

## 1.4 Постановка задач дослідження

Проаналізувавши питання розробки інформаційного середовища керування контентом та користувацькими комунікаціями, було визначено завдання, які необхідно виконати для розробки програмного додатку:

- аналіз існуючих рішень;
- виконати порівняльний аналіз аналогів;
- визначити основні завдання які необхідно вирішити для розробки колективного інформаційного середовища;
- виконати проектування колективного інформаційного середовища;
- вибір програмних засобів для вирішення поставлених завдань;
- розробити фронтенд частину для системи керування контентом;
- інтегрувати користувацькі комунікації у систему керування контентом;
- виконати тестування розробленої системи.

## 1.5 Висновки

У другому розділі дослідження було приділено увагу детальному аналізу основних етапів розробки інформаційної системи керування контентом з користувацькими комунікаціями. Починаючи з визначення вихідних вимог до системи, було розглянуто можливості їхнього деталізованого визначення та уточнення.

Окрему увагу приділено вибору технологій, які будуть використовуватися під час розробки програмного додатку. Проведений аналіз вибору мов програмування, фреймворків та інших інструментів дозволив обґрунтувати оптимальний набір технологій для досягнення поставлених цілей.

У зв'язку з визначенням технічних аспектів розробки, важливим етапом стало визначення функціональності майбутнього програмного додатку. Здійснено докладний аналіз вимог користувачів та бізнес-потреб, що визначило основні функціональні можливості програмного продукту.

# 2. РОЗРОБКА СТРУКТУРИ ТА МЕТОДІВ ПРОГРАМНОГО ПРОДУКТУ

## 2.1 Метод спільної фільтрації

Системи рекомендацій на основі спільного фільтрування покладаються виключно на минулі взаємодії між користувачами та елементами, щоб пропонувати нові продукти. Особливості кожного окремого предмета не враховуються.

У спільній фільтрації дані користувача, який взаємодіє з елементами, записуються та зберігаються. Зазвичай це представлено матрицею, відомою як матриця взаємодії між користувачем і елементом, де рядки представляють користувачів, а стовпці — елементи. Схожі користувачі групуються, і всі їхні взаємодії враховуються під час надання рекомендацій цільовому користувачеві [11]. Реакція може бути явною (оцінка за шкалою від 1 до 5, подобається чи не подобається) або прихованою (перегляд товару, додавання його до списку бажань, час, витрачений на статтю).

Ці дані представлені у вигляді матриці, що складається з реакцій набору користувачів на деякі елементи з набору елементів. Кожен рядок міститиме оцінки, надані користувачем, а кожен стовпець міститиме оцінки, отримані елементом. Матриця показує п’ять користувачів, які оцінили деякі елементи за шкалою від 1 до 5. Наприклад, перший користувач оцінив 4 для третього елемента.  Приклад матриці взаємодії зображено на рисунку 2.1.

| | $i_1$ | $i_2$ | $i_3$ | $i_4$ | $i_5$ |
|---|---|---|---|---|---|
| $u_1$ | 5 | | 4 | 1 | |
| $u_2$ | | 3 | | 3 | |
| $u_3$ | | 2 | 4 | 4 | 1 |
| $u_4$ | 4 | 4 | 5 | | |
| $u_5$ | 2 | 4 | | 5 | 2 |

Рисунок 2.1 – Матриця взаємодії

У більшості випадків клітинки в матриці порожні, оскільки користувачі оцінюють лише кілька елементів. Малоймовірно, що кожен користувач буде оцінювати або реагувати на кожен доступний елемент. Матриця з переважно порожніми клітинками називається розрідженою, а протилежна до неї (здебільшого заповнена матриця) називається щільною.

Щоб побудувати систему, яка може автоматично рекомендувати елементи користувачам на основі вподобань інших користувачів, першим кроком є пошук схожих користувачів або предметів. Другим кроком є прогнозування рейтингів елементів, які ще не оцінені користувачем.

Одна важлива річ, про яку слід пам’ятати, полягає в тому, що в підході, заснованому виключно на спільній фільтрації, подібність не обчислюється за такими факторами, як вік користувачів, категорія статті або будь-які інші дані про користувачів або елементи. Він обчислюється лише на основі оцінки (явної чи неявної), яку користувач дає елементу. Наприклад, двох користувачів можна вважати схожими, якщо вони дають однакові оцінки десяти статтям, незважаючи на велику різницю у віці.

Показником для вимірювання точності є середня абсолютна похибка (MAE), за якою ви визначаєте величину похибки, знаходячи її абсолютне значення, а потім беручи середнє значення всіх похибок.

Як зображено на рисунку 2.2 метод спільної фільтрації поділяється на метод на основі пам’яті та метод на основі моделі.

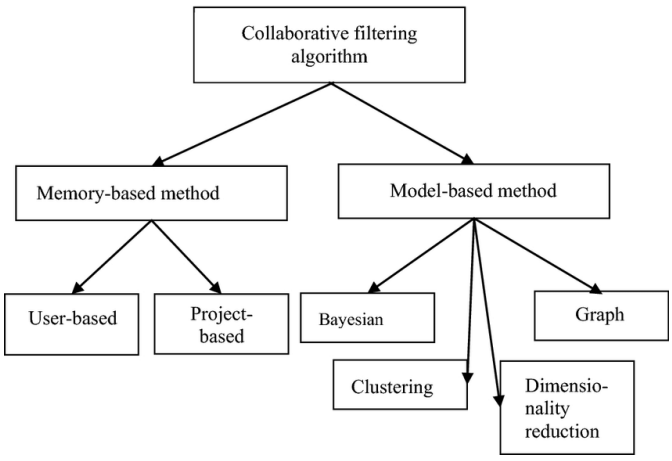


Рисунок 2.2 – Метод спільної фільтрації

Memory-based collaborative filtering (CF) — це метод рекомендаційних систем, який базується на спільному використанні відгуків або оцінок користувачів для рекомендації контенту. Цей метод використовує інформацію про подібність між користувачами або елементами (товари, фільми, тощо) для здійснення рекомендацій. Цей метод використовує такі кроки:

Розрахунок подібності між користувачами або елементами. Для цього часто використовуються метрики, такі як косинусна схожість, кореляція Пірсона або Євклідова відстань. Схематичне зображення косинусної схожості на прикладі 4 користувачів, які подивились 2 фільми і поставили оцінки від 1 до 5 представлене на рисунку 2.3

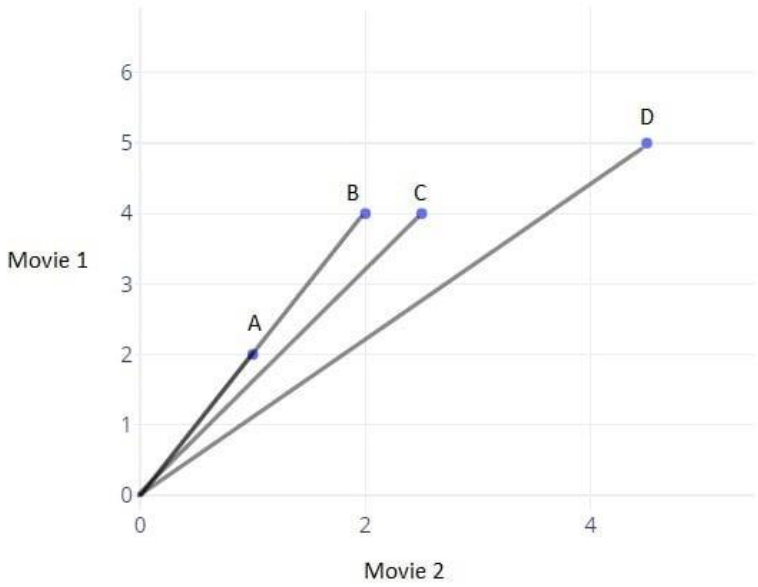


Рисунок 2.3 – Приклад обчислення схожості смаків користувачів

На графіку показано чотири лінії, що з’єднують кожну точку з початком координат. Прямі A і B збігаються, тому кут між ними дорівнює нулю. Можна вважати, що якщо кут між лініями збільшується, то схожість зменшується, а якщо кут дорівнює нулю, то користувачі дуже схожі. Щоб обчислити подібність за допомогою кута,  потрібна функція, яка повертає вищу подібність або меншу відстань для меншого кута та меншу подібність або більшу відстань для більшого кута.

На основі подібності генеруються рекомендації для користувача. Основні

переваги memory-based collaborative filtering включають простоту інтерпретації та ефективність в реальному часі. Однак вони також мають деякі недоліки, такі як проблема холодного старту (не може рекомендувати новим користувачам або елементам) і обмежена масштабованість до великих наборів даних.

Також memory-based collaborative filtering поділяється на user-based та item-based. У user-based підході визначається подібність між користувачами на основі їхніх відгуків або оцінок. Коли користувач запитує рекомендації, система знаходить інших користувачів, які схожі на нього, і рекомендує те, що подобається цим схожим користувачам. Роботу user-based моделі зображено на рисунку 2.4 [12].

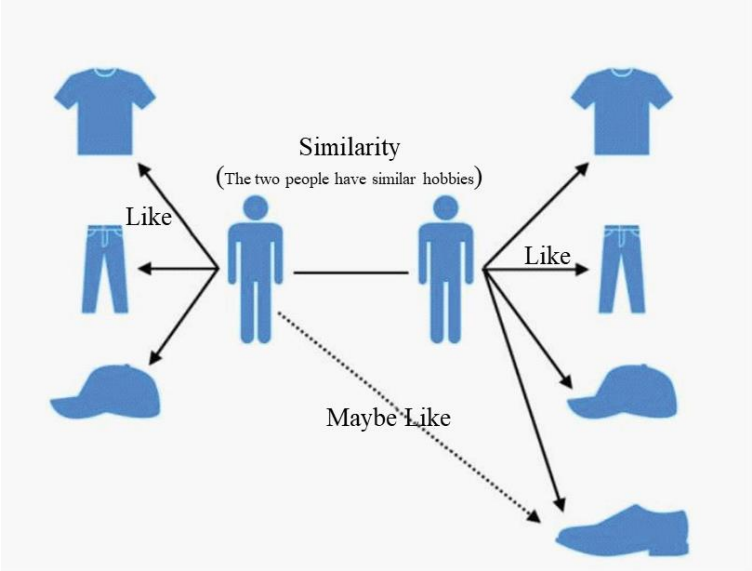


Рисунок 2.4 – User-based модель

У item-based моделі визначається подібність між елементами (товарами, фільмами і т.д.). Коли користувач взаємодіє з певним елементом, система рекомендує інші елементи, які схожі на той, з яким взаємодіє користувач. Спільна фільтрація на основі елементів була розроблена Amazon. У системі, де більше користувачів, ніж елементів, фільтрація на основі елементів є швидшою та стабільнішою, ніж на основі користувачів. Це ефективно, оскільки зазвичай середня оцінка, яку отримує товар, не змінюється так швидко, як середня оцінка, яку користувач дає різним елементам. Також відомо, що він ефективніший, ніж модель, орієнтована на користувача, коли матриця оцінок розріджена. Приклад

такої моделі наведено на рисунку 2.5 [12].

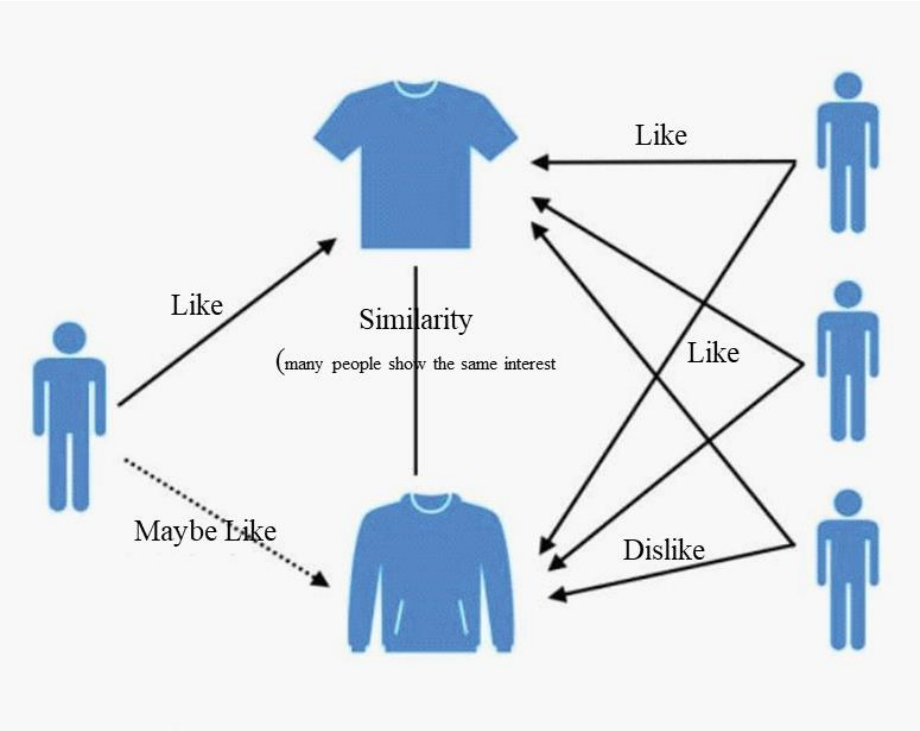


Рисунок 2.5 – Item-based модель

Модель спільної фільтрації також може використовувати модельно-орієнтований підхід (Model-Based Approach), що використовує побудовану модель, яка репрезентує внутрішні залежності або характеристики користувачів та елементів. Цей підхід дозволяє рекомендаційній системі зрозуміти та передбачити відповіді користувачів на новий контент без прямого порівняння їхніх відгуків або оцінок з іншими користувачами. Основні етапи model-based підходу включають:

- Розроблення математичної моделі, яка описує відносини між користувачами та елементами. Це може бути, наприклад, матричний факторизаційний метод або байєсівська мережа.
- Використання тренувальних даних для навчання параметрів моделі. Навчання моделі означає налаштування її параметрів так, щоб вона найкращим чином передбачала реальні оцінки користувачів.
- Використання навченої моделі для передбачення рейтингів, які

користувачі нададуть елементам, які їм ще не були представлені.

- Використання прогнозованих рейтингів для генерації персоналізованих рекомендацій для користувачів.

Переваги model-based підходу включають можливість роботи з розрідженими даними (коли користувачі мають невелику кількість оцінок) і можливість врахування складних взаємодій між користувачами та елементами. Однак ці підходи можуть вимагати більше обчислювальних ресурсів та експертизи для налаштування моделей.

### 2.3 Метод фільтрації на основі вмісту

Метод фільтрації на основу вміст ґрунтується на описі елемента та виборі користувача. У системі рекомендацій, що базується на вмісті, ключові слова використовуються для опису елементів. Крім того, створюється профіль користувача, який допомагає визначити тип елемента, що сподобався користувачеві. Іншими словами, алгоритми намагаються рекомендувати продукти, схожі на ті, які вже сподобалися користувачеві. Головна ідея фільтрації на основі вмісту полягає в тому, що, якщо вам подобається певний елемент, вам сподобається і "схожий" елемент [13].

Наприклад, якщо рекомендується фільм або пісня, схожа на той, який вам вже сподобався. Основна проблема фільтрації на основі вмісту полягає в тому, чи може система вивчити вподобання користувача відносно одного джерела вмісту та успішно відтворювати їх для інших типів вмісту. Якщо система обмежується рекомендаціями лише вмісту того самого типу, який вже використовується користувачем, користь від системи рекомендацій суттєво зменшується, коли вона рекомендує різні типи вмісту з інших служб.

Також під час цього методу створюється профіль користувача. Це набір векторів, які визначають уподобання користувача. Профіль базується на діяльності та смаках користувача, наприклад, оцінки користувача, кількість кліків на різних елементах. Ця інформація допомагає системі рекомендацій

найкраще оцінити нові пропозиції.

Для фільтрації на основі вмісту ми вимагаємо, щоб різні характеристики кожного окремого елемента відображали його основні якості. На прикладі інформаційної системи та статей це можуть бути деякі необхідні атрибути, які допоможуть системі рекомендацій розрізняти їх, це назва, категорія, теги, час перегляду статті, тощо. Принцип роботи метода графічно зображено на рисунку 2.6.

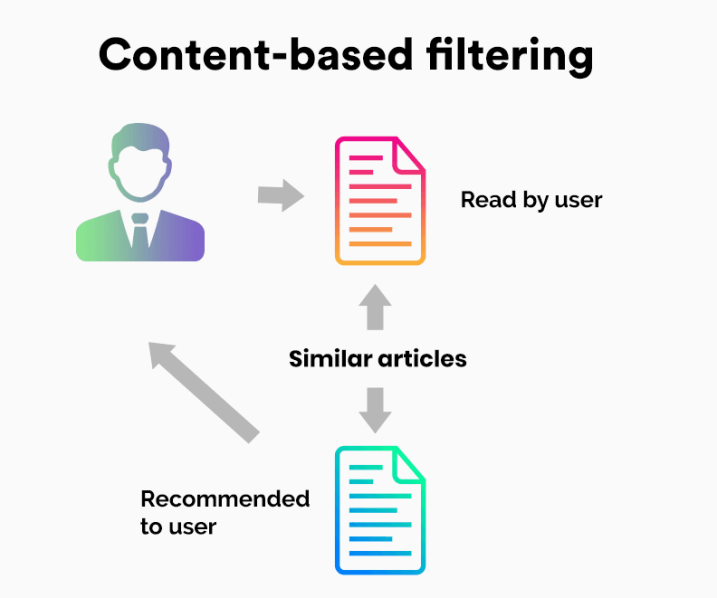


Рисунок 2.6 – Content-based filtering

Не дивлячись на переваги цього методу, він також має певні недоліки, а саме:

- Створення механізму рекомендацій на основі вмісту потребує багато знань предметної області, оскільки вибір функцій елементів здебільшого жорстко закодований у системі;
- Модель може рекомендувати нові елементи на основі поточного інтересу користувача. Отже, виявити та розширити нові шляхи, які можуть зацікавити користувача, неможливо;
- Проблема холодного запуску є значним недоліком, оскільки система рекомендацій не має достатньо інформації про нового користувача, щоб

почати робити пропозиції;

- Важко давати нові рекомендації не дуже активним користувачам.

### 2.4 Гібридний метод рекомендацій

Хоча алгоритми спільної фільтрації та фільтрації на основі вмісту широко використовувався, все ще існують деякі проблеми, такі як розрідженість даних, холодний старт і закінчення терміну дії інформації тощо.

Для вирішення вищезазначених проблем було внесено серію вдосконалень на основі традиційного алгоритму спільної фільтрації, і вони досягли певного успіху. Наприклад, Пірасте та ін. [14] пом’якшили проблеми розрідженості та холодного старту матриці, проводячи попереднє опитування користувача. Кумар та ін. [15] використовували технологію декомпозиції матриці, щоб зменшити розмірність матриці та підвищити точність результату рекомендації. Сан і Донг [16] запропонували динамічну модель дрейфу часу, враховуючи вплив змін інтересу користувачів на подібність у різні періоди часу. Вонг та ін. [17] запропонував алгоритм спільної фільтрації, що поєднує модель KNN і модель XGBoost. Зарзур та ін. [18] представили нову ефективну спільну фільтрацію на основі моделі для покращення якості рекомендацій. Крім того, існують деякі алгоритми спільної фільтрації, засновані на кластеризації [19], нейронних мережах [20] і різних імовірнісних моделях [21]. Наведені вище дослідження певною мірою оптимізували модель рекомендацій і підвищили точність результатів рекомендацій, але все ще є деякі проблеми, які потребують подальшого вивчення. Наприклад, більшість існуючих алгоритмів спільної фільтрації враховують лише рейтингову інформацію серед користувачів, але ігнорують характеристики користувача та вплив популярних елементів на схожість користувачів, що призводить до поганих результатів рекомендацій.

Щоб ще більше підвищити точність рекомендацій та зменшити вплив проблеми з холодним стартом, пропонується гібридна модель рекомендацій на основі методу спільної фільтрації та фільтрації за вмістом. Даний метод

використовує окремі рекомендаційні списки для методу спільної фільтрації та фільтрації за вмістом. Після цього можна використовувати комбінацію цих списків для отримання остаточного списку рекомендацій для користувача. Ваги можуть бути динамічно або статично налаштовані.

Отже, гібридний метод рекомендацій на основі методу спільної фільтрації та фільтрації за вмістом може виявитися дуже ефективним у покращенні точності рекомендацій та подоланні проблеми холодного старту. В цій моделі поєднуються переваги колаборативного підходу, який враховує взаємодії між користувачами, та контентного підходу, який використовує характеристики об'єктів вмісту. Метод схематично зображений на рисунку 2.7.

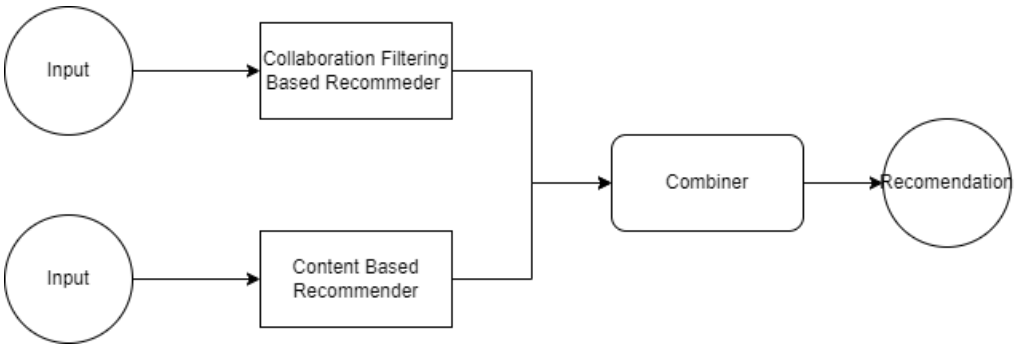


Рисунок 2.7 – Гібридний метод рекомендацій

## 2.5 Розробка архітектури системи керуванням контенту з користувацькими комунікаціями

Розробка архітектури системи керування контентом з користувацькими комунікаціями є однією з найважливіших та складних частин процесу створення такої інформаційної системи. Впродовж останніх десятиліть розробка архітектури систем керування контентом з користувацькими комунікаціями зазнала значного розвитку та змін. Цей процес був обумовлений швидким технологічним розвитком, зростанням обсягу та різноманітності інформації, а також зміною потреб користувачів. Серед шаблонів проектування було обрано MVVM, адже він задовільняє усім вимогам для розробки системи керування

контентом, а саме: MVVM дозволяє ефективно розділити логіку додатку на три основні компоненти: Model, View та ViewModel. Model представляє дані та бізнес-логіку, View відповідає за відображення інтерфейсу, а ViewModel служить посередником між ними. Це розділення полегшує розробку, тестування та управління кодом. Також MVVM використовує механізм data binding для автоматичного синхронізування даних між ViewModel та View. Це дозволяє вам легко відображати та оновлювати дані без прямого втручання в код інтерфейсу. У MVVM зв'язок між View і ViewModel часто буває більш зручним для роботи зі станом, ніж пряма інтеракція, яка часто властива MVC та MVP.

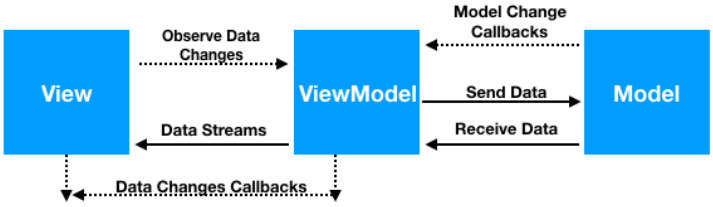


Рисунок 2.8 – Шаблон проектування MVVM

Для веб додатків зазвичай розглядають монолітну, модульно-монолітну, та модульно-розподілену архітектури (рисунок 2.9) .

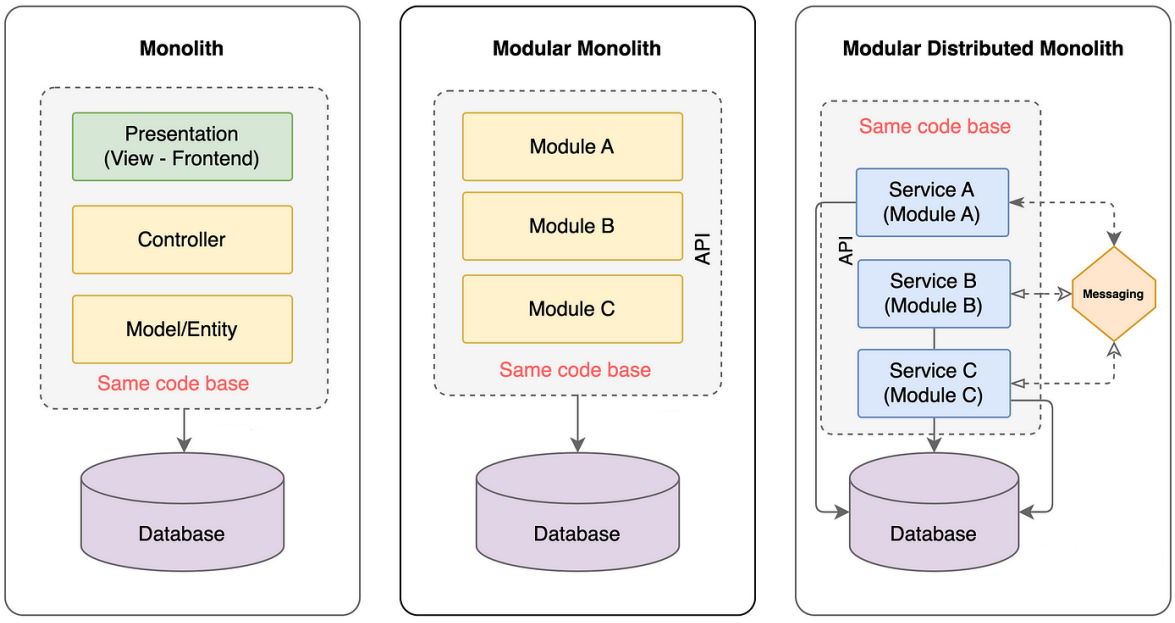


Рисунок 2.9 – Графічне зображення 3 типів архітектур

Монолітна архітектура є типовим підходом до розробки програмного забезпечення, де весь функціонал додатку об'єднується в один блок, відомий як моноліт. Основна ідея полягає в тому, що весь код, включаючи серверну логіку та базу даних працює як єдиний юніт. Ось деякі ключові аспекти монолітної архітектури:

- У моноліті весь код програми розміщується в єдиному застосунку, тобто весь функціонал, включаючи сервер, клієнтську логіку та базу даних, працює як одне ціле.
- Моноліт використовує спільний код та ресурси для всіх його компонентів. Це дозволяє легко обмінювати інформацією та даними між різними частинами додатку.
- Один цілісний застосунок розгортається як єдиний блок, що полегшує розгортання і управління.
- Масштабування може бути обмеженим, оскільки зазвичай весь додаток повинен бути масштабованим разом.
- У монолітній архітектурі розробка та тестування можуть бути простіше, оскільки все розташовано в одному місці.
- Зазвичай моноліти використовують одну централізовану базу даних для всіх їх компонентів.

Хоча монолітна архітектура може бути простою для розробки та розгортання, вона може стати обмеженням при рості проекту, збільшенні команди розробників та вимогах до масштабованості. У таких випадках розглядають альтернативи, такі як мікросервісна архітектура, яка дозволяє розділити додаток на невеликі, незалежні мікросервіси. Модульний розподілений моноліт, часто відомий як Micro Frontends, представляє собою підхід до розробки веб-додатків, де фронтенд розбивається на невеликі, незалежні частинии. Кожен мікрофронтенд розглядається як окремий юніт розробки та розгортання. Ось деякі ключові аспекти цієї архітектури:

- Кожен мікрофронтенд розглядається як незалежний модуль з власним

фронтендом, бізнес-логікою та стеком технологій. Це дає можливість розробляти, тестувати та розгортати кожен мікрофронтенд незалежно.

- Мікрофронтенди дозволяють ефективно масштабувати окремі частини системи, що полегшує підтримку та розвиток.
- Введення меж між мікрофронтендами може забезпечити ізоляцію помилок та підвищити безпеку системи.

Мікрофронтенди дозволяють працювати над окремими функціональними частинами додатку, що робить розробку, тестування та розгортання більш гнучкими та ефективними, особливо у великих та складних проектах.

Модульно-монолітна архітектура є комбінацією ідеї модульності та монолітної архітектури. У цьому підході застосовуються модульні принципи для організації коду внутрішньо монолітного додатку. Ось деякі ключові аспекти модульно-монолітної архітектури:

- Код розбивається на невеликі, самостійні модулі, кожен з яких відповідає за конкретний функціонал.
- Кожен модуль може бути відокремлений ізольовано від інших, що дозволяє зберігати велику кількість функціоналу в окремих логічних блоках.
- Модулі можуть спільно використовувати загальні ресурси, такі як сервіси чи бібліотеки, що полегшує повторне використання коду.
- Кожен модуль може бути протестований незалежно від інших, що полегшує тестування та забезпечує надійність коду.

Цей підхід дозволяє вам зберігати переваги модульності, такі як легше тестування і розширюваність, у межах єдиної монолітної програми. Також на відміну від мікрофронтендів вони взаємодіють не через стандартизовані API, а через спільні ресурси і внутрішні інтерфейсу. Розібравши особливості кожної архітектури, було обрано модульно-монолітну, як таку, що найбільш точно відповідає вимогам з розробки.

Модульність – це ключова характеристика будь-якої сучасної CMS, яка дає можливість додавати, оновлювати та видаляти функціональні модулі без зміни

основної системи. Модульність грає важливу роль у розширенні функціоналу та адаптації CMS до конкретних потреб користувачів. Для досягнення максимальної модульності, CMS може використовувати розгалужену структуру, де кожен модуль має власний набір функцій та API [22]. Це дозволяє зберігати інтерфейси чіткими та стабільними, незважаючи на зміни внутрішньої реалізації.

Модульність сприяє покращенню масштабованості та підтримці системи. Кожен модуль може бути розроблений та тестований окремо, що полегшує внесення змін або додавання нового функціоналу без великих ризиків порушення роботи інших частин системи.

Розподіл системи на модулі також дозволяє використовувати різні технології та мови програмування для реалізації кожного модуля відповідно до його особливостей та вимог. Це дає можливість використовувати найкращі та найбільш підходящі інструменти для кожного конкретного завдання.

Гнучкість архітектури забезпечує зручність внесення змін в систему, адаптацію до нових вимог бізнесу та швидку реакцію на зміни у технологічному оточенні. Це особливо важливо в умовах постійної зміни вимог користувачів та конкурентного середовища.

У розгалуженій структурі модулів переважно існують основні модулі, які відповідають за базовий функціонал системи. Наприклад, це може бути модуль для управління контентом, модуль для управління користувачами, модуль для SEO оптимізації тощо. Оскільки модулі в розгалуженій структурі можуть бути взаємозамінними, важливо мати систему взаємодії між ними. Це досягається завдяки стандартизованим інтерфейсам, які дозволяють модулям обмінюватися даними та викликами функцій. Розгалужена структура також дозволяє розподілити модулі за функціональністю. Наприклад, може бути окремий модуль для роботи з мультимедіа (зображення, відео), інший - для роботи зі структурованими даними (наприклад, розкладом університетських занять), і ще інший - для роботи з користувачами та їхніми правами. Розгалужена структура дозволяє легко розширювати функціональність системи. Наприклад, для додавання нової можливості, якщо вже існує модуль для роботи зі статтею, може

бути розроблений окремий модуль для роботи зі статистикою переглядів статей, і він може бути легко підключений та інтегрований.

Одним із важливих алгоритмів у системі є алгоритм підвантаження модулів – це важлива частина архітектури системи керування контентом, яка впливає на продуктивність та ефективність системи. Основна ідея полягає в тому, щоб завантажувати та виконувати модулі лише тоді, коли вони потрібні, замість того, щоб завантажувати всі модулі при запуску системи. Існують декілька механізмів підвантаження модулів: Механізм лінійного завантаження, також відомий як "lazy loading," [23] дозволяє системі завантажувати модулі лише при їх першому виклику або коли вони стають актуальними для обробки певного запиту. Це зменшує час завантаження системи при старті і дозволяє заощадити ресурси. Або ж система може мати плагін-систему, яка дозволяє додавати розширення та модулі до системи під час її роботи. Це означає, що адміністратор системи може легко додавати нові функціональність без перезавантаження або зупинки системи. Можливість додавати та оновлювати модулі без перезавантаження системи сприяє зменшенню витрат на обслуговування. Адміністраторам не потрібно зупиняти систему для внесення змін, що робить процес обслуговування більш ефективним.

Розробка системи управління контентом є однією з ключових фаз при створенні інформаційної платформи, яка має забезпечити ефективне створення, редагування, та розповсюдження різноманітного контенту. У цьому контексті, основною метою є створення модульної та гнучкої архітектури, яка забезпечить не тільки зручність управління контентом, але й надійність, швидкість реакції на зміни та здатність взаємодіяти з іншими системами.

База даних відіграє ключову роль у зберіганні та організації контенту. Це повинна бути оптимізована для швидкого доступу та здатна вміщувати різноманітні типи даних – текст, графіку, відео, аудіо тощо. Зручний інтерфейс управління контентом має бути інтуїтивно зрозумілим та включати засоби для введення, редагування та видалення різноманітних видів контенту.

Важливим аспектом є система керування версіями та історією. Можливість

відстежування змін та відновлення попередніх версій контенту забезпечить надійність та консистентність інформації. Інтеграція з іншими системами, такими як CRM або аналітичні інструменти, розширить функціональність та забезпечить цілісність даних.

CMS повинна бути гнучкою щодо обробки різноманітного контенту та здатною індивідуалізувати вигляд та доступ для кожного користувача. При цьому, врахування вимог SEO для оптимізації контенту під пошукові системи визначає ефективність просування матеріалів у мережі.

Система керування контентом повинна підтримувати різні типи контенту, включаючи текст, зображення, відео, аудіо, документи тощо. Для оптимізації зберігання різних типів контенту, можна використовувати спеціалізовані сховища даних або бази даних, які підтримують різні формати. Наприклад, зберігати тексти у структурованому вигляді, зображення та відео у вигляді файлів, аудіо у спеціальних форматах, а документи можна зберігати у PDF або інших форматах. Використання відповідних типів сховищ дозволить зменшити обсяг займаного місця та забезпечить швидкий доступ до контенту.

У контексті системи управління контентом (CMS), розробка інтерфейсу користувача є важливою частиною, оскільки вона визначає зручність та ефективність взаємодії користувачів з платформою. Інтуїтивно зрозумілий інтерфейс забезпечує зручність введення та редагування контенту, а також навігацію між різними розділами системи.

Розробка інтерфейсу повинна враховувати потреби різних категорій користувачів, включаючи адміністраторів, редакторів, та звичайних відвідувачів. Адміністративний інтерфейс повинен бути потужним і в той же час інтуїтивно зрозумілим, забезпечуючи можливість ефективного управління різними аспектами системи.

Механізми візуального редагування та попереднього перегляду контенту грають ключову роль у полегшенні процесу створення та редагування матеріалів. Це важливо не лише для редакторів з глибокими технічними знаннями, але і для звичайних користувачів, які можуть бути менш знайомі з технічними деталями

веб-розробки.

Окрім цього, система повинна підтримувати історію та керування версіями контенту. Це надає можливість відстежувати всі зміни, вносити корективи та повертатися до попередніх версій матеріалів. Ця функціональність є важливою для забезпечення цілісності та безпеки контенту.

Система керування користувацькими комунікаціями визначає ефективність взаємодії між платформою та її користувачами. Це охоплює різноманітні аспекти, від сповіщень та внутрішньої пошти до функціоналу соціальних мереж та інтерактивних можливостей.

Важливою частиною є налагодження ефективних сповіщень. Це включає в себе інформування користувачів про новий контент, зміни в їхніх профілях або важливі події на платформі. Сповіщення можуть бути як електронними листами, так і повідомленнями в мобільних додатках або через внутрішню систему повідомлень.

Інтерактивна пошта та система обміну повідомленнями грають ключову роль у взаємодії користувачів. Забезпечення зручного та функціонального інтерфейсу для листування, можливості прикріплювати файли та вставляти мультимедійний контент, робить комунікацію ефективною та приємною.

Система також повинна підтримувати функції соціальних мереж, такі як коментарі, вподобання та репости. Це розширює можливості взаємодії користувачів між собою та з контентом, зроблює взаємодію більш динамічною та залучаючою.

Окрім цього, система повинна надавати можливості для проведення опитувань, голосувань та інших форм взаємодії, які дозволяють залучити користувачів до активної участі. Це не лише збагачує взаємодію, але й надає можливість збору важливого фідбеку.

Успішна архітектура керування користувацькими комунікаціями визначається розумінням потреб аудиторії та забезпеченням ефективних, інтерактивних засобів взаємодії між користувачами та платформою.

Персоналізація в системі виграває ключову роль у створенні унікального

та приємного досвіду для кожного користувача. Це охоплює широкий спектр аспектів, починаючи від налаштувань інтерфейсу та завершуючи індивідуалізованим контентом.

Користувачі повинні мати можливість налаштовувати свої профілі, обирати теми, встановлювати сповіщення та керувати іншими параметрами для оптимального використання платформи. Персоналізація також може включати адаптивний вміст, який аналізує поведінку користувача та надає рекомендації, відповідно до їхніх інтересів та попереднього використання.

Індивідуалізований контент є важливим елементом персоналізації. Система повинна враховувати історію переглядів, вподобань та коментарів користувачів для надання наступних рекомендацій та контенту, який найбільше відповідає їхнім інтересам.

Аналітика визначається здатністю системи збирати, обробляти та інтерпретувати дані для прийняття обґрунтованих рішень. Це включає в себе аналіз використання платформи, поведінки користувачів, та інших параметрів для вдосконалення якості та ефективності системи.

Аналітичні інструменти повинні забезпечувати зручний доступ до ключової інформації, використовуючи графіки, діаграми та інші методи візуалізації. Вони дозволяють слідкувати за трендами, визначати популярність контенту, та виявляти можливості для покращення.

Збір та аналіз даних також важливий для розуміння ефективності рекламних кампаній, конверсій, та інших метрик, що дозволяє оптимізувати стратегії та ресурси. Аналітика допомагає не лише вдосконалити внутрішні процеси, але і адаптувати платформу до змін у вимогах аудиторії та ринкових умов.

Загалом, впровадження персоналізації та аналітики в архітектурі системи сприяє створенню індивідуалізованого та оптимізованого досвіду користувачів, а також дозволяє бізнесу приймати обґрунтовані рішення для покращення функціоналу та ефективності платформи.

Забезпечення безпеки є однією з найважливіших складових успішної

інформаційної системи. Оскільки платформа здатна містити конфіденційну та особисту інформацію користувачів, безпека повинна бути вбудована на всіх рівнях архітектури.

Система повинна використовувати надійні механізми аутентифікації для впізнавання користувачів та контролю доступу до різних ресурсів. Використання двофакторної аутентифікації, паролів високої складності та інших методів допомагає захистити облікові записи від несанкціонованого доступу.

Авторизація повинна бути гнучкою та базуватися на ролях, що дозволяє точно визначати права кожного користувача в системі. Використання принципів "найменших привілегій" та системи контролю доступу допомагає уникнути надмірних прав та зменшити ризик витоку конфіденційної інформації.

Конфіденційність та цілісність даних є важливими аспектами забезпечення безпеки. Система повинна використовувати сучасні методи шифрування для захисту даних під час транспорту та в спокійному стані. Шифрування баз даних, особливо для конфіденційної інформації, є обов'язковим.

Регулярне резервне копіювання та відновлення даних грають ключову роль у відновленні системи після можливого інциденту, такого як атака або випадкове видалення даних.

Захист від кібератак включає в себе використання систем антивірусів, брандмауерів, та систем виявлення вторгнень. Регулярне оновлення програмного забезпечення та моніторинг системи на предмет підозрілих активностей допомагають у попередженні та реагуванні на потенційні загрози.

При розробці програмного забезпечення важливо враховувати безпеку на рівні коду. Використання безпечних практик програмування, аудит коду на предмет можливих уразливостей, та регулярні тести на безпеку є необхідними етапами для забезпечення надійності системи.

Загалом, ефективне забезпечення безпеки вимагає комплексного підходу на всіх рівнях архітектури системи, починаючи від засобів аутентифікації та закінчуючи заходами забезпечення безпеки на рівні коду. Тільки цілісний підхід дозволить платформі впевнено стояти перед сучасними кіберзагрозами та

забезпечувати конфіденційність та безпеку даних користувачів.

Одним з ключових аспектів є оптимізація відображення різних типів контенту на веб-сайті або іншому інтерфейсі користувача. Для цього можна використовувати різні методи, такі як кешування, стиснення зображень та відео, асинхронну завантаження ресурсів, що дозволяє прискорити завантаження сторінок та зменшити навантаження на сервер.

Для забезпечення швидкого пошуку та індексації різних типів контенту бажано використовувати спеціальні індексаційні механізми. Наприклад, текстовий контент може бути індексований за допомогою повнотекстового пошуку, а відео та аудіо можуть бути індексовані за допомогою аналізу метаданих та автоматичного розпізнавання мови.

Ще одна функція без якої не обійтись при створені системи керування контентом це керування правами доступу. Це важлива функція для забезпечення безпеки та конфіденційності інформації в системі керування контентом. Ця функція дозволяє обмежувати доступ до різних ресурсів та функціональності в залежності від ролі та прав користувачів. Необхідно визначити різні ролі користувачів. Це можуть бути, наприклад, адміністратори, редактори, автори, гостьові користувачі тощо. Кожна роль має свої права доступу до ресурсів системи. Також для кожної ролі користувача потрібно визначити список прав доступу. Це може включати права на перегляд, редагування, видалення або додавання ресурсів. Наприклад, адміністратор може мати права на доступ до всіх функцій, тоді як гостьовий користувач може мати обмежений доступ. Адміністратор повинен мати зручні інструменти для управління користувачами, включаючи можливість блокувати, розблоковувати, видаляти або редагувати облікові записи користувачів. Також для системи керування контентом з користувацькими комунікаціями необхідно створити можливість інтеграції зовнішніх систем аутентифікації, таких як LDAP або SSO (одноразовий вхід) [24]. Це дозволяє користувачам використовувати свої існуючі облікові записи для доступу до системи.

Кешування є важливою частиною архітектури системи керування

контентом, оскільки воно дозволяє значно покращити швидкодію та продуктивність системи. Кешування — це процес зберігання попередньо підготовлених даних або сторінок в спеціальному місці, яке називається кешем. Кеш використовується для збереження даних, які імовірно будуть запитані знову у недалекому майбутньому, замість того, щоб витрачати час на їх повторне обчислення або генерацію.

Алгоритм роботи кешування в системі керуванням контентом:

- запит на контент: Коли користувач або система CMS надсилає запит на отримання певного контенту (наприклад, сторінки веб-сайту), система спочатку перевіряє, чи є ця сторінка або дані в кеші;
- перевірка кешу: Система перевіряє кеш на наявність відповідних даних. Якщо дані знаходяться в кеші і не застаріли (за часом життя кешу), система може негайно повернути їх як відповідь на запит;
- відсутність в кеші або застарілий кеш: Якщо дані відсутні в кеші або кеш вже застарів, система виконує потрібні обчислення або запит до бази даних для отримання актуальних даних;
- збереження у кеші: Після отримання актуальних даних система оновлює кеш, зберігаючи ці дані в ньому для подальших запитів. Кеш може бути налаштований з визначеним часом життя, після якого дані в ньому автоматично стають застарілими і поновлюються.

Алгоритм можна покращити додавши деякі зміни:

- стратегія кешування: система повинна мати налаштовану стратегію кешування, яка визначає, які дані кешувати, як тривалий час життя кешу та як кеш оновлювати. Для статичного контенту, наприклад, може використовуватися довгий час життя кешу, тоді як для динамічного – короткий;
- інвалідація кешу: система повинна вміти інвалідувати (очищати) кеш, коли дані стають застарілими або змінилися. Це може відбуватися автоматично або відповідно до певного розкладу оновлень;
- робота з великим обсягом даних: Для великих систем, де обсяг даних

великий, може бути важливо використовувати розподілене кешування або CDN (мережу доставки контенту), щоб забезпечити швидкий доступ до контенту для користувачів з усього світу;

– кешування на різних рівнях: Кешування може бути реалізоване на різних рівнях, включаючи рівень сервера, рівень бази даних та рівень додатків. Оптимальна стратегія кешування залежатиме від конкретних потреб проекту.

В результаті оптимізації кешування, система CMS може значно підвищити швидкодію та продуктивність, особливо в умовах високого трафіку та великого обсягу даних. Кешування є важливим елементом для забезпечення користувачам швидкого та ефективного доступу до контенту в онлайн-середовищі.

Також, оскільки наша система керування контентом буде мати користувацькі комунікації, необхідно розробити гнучку архітектуру для успішної інтеграції з іншими системами, яка дозволяє легко додавати та налаштовувати інтеграційні модулі. Ця гнучкість забезпечує можливість адаптації до різних вимог інтеграції. Важливо розробити інтеграцію з використанням стандартів індустрії, таких як RESTful або SOAP API. Це робить інтеграцію більш сумісною з іншими системами, що підтримують такі стандарти.

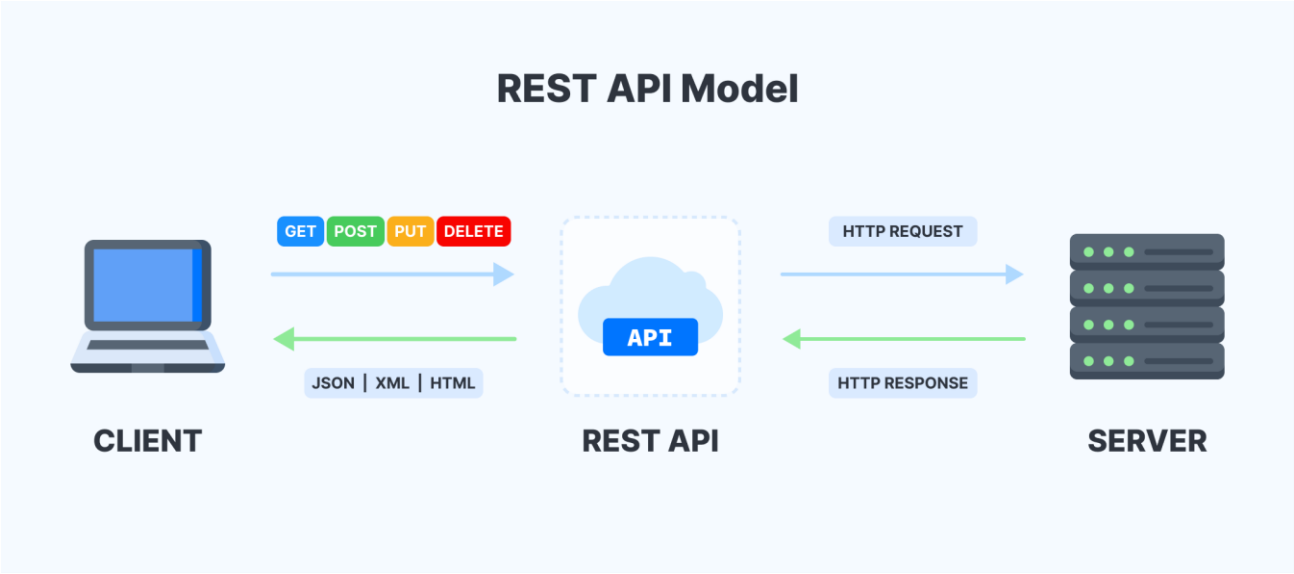


Рисунок 2.10 – Графічне зображення роботи з RESTful API

Інтеграція з соціальними мережами може включати можливість автоматично публікувати контент на сторінках соціальних мереж, взаємодіяти з API соціальних платформ для збору даних або навіть авторизацію через облікові записи соціальних мереж як Google, Facebook або X (колишній Twitter). Інтеграція з аналітичними інструментами, такими як Google Analytics або Adobe Analytics, дозволяє збирати та аналізувати дані про користувачів та їх взаємодію з контентом. Це допоможе розробляти більш інформовані стратегії змісту.

Для розширення функціональності, CMS може інтегруватися з різними зовнішніми сервісами, такими як платіжні системи, поштові служби або інші зовнішні API, які можуть бути корисними додатку.

Сучасний та естетично збалансований дизайн інтерфейсу є важливим аспектом забезпечення приємного та зручного користувацького досвіду. Він повинен враховувати принципи юзабіліті, забезпечуючи простоту навігації та доступ до ключових функцій.

Розробка зручного та інтуїтивно зрозумілого інтерфейсу є критично важливою складовою успіху системи керування контентом. Інтерфейс адміністратора та користувачів повинен бути приємним та ефективним для використання. Для досягнення цього можна використати технології розробки інтерфейсу, такі як HTML5, CSS3, JavaScript та фреймворки для створення веб-додатків, які дозволяють створити інтерактивні та зручні інтерфейси. Оскільки все більше користувачів відвідує веб-сайти з мобільних пристроїв, важливо мати мобільно-адаптивний інтерфейс, який зручно працює на різних типах пристроїв та розмірах екранів. Дизайн має бути інтуїтивно зрозумілим: меню, кнопки, форми та інші елементи повинні бути розташовані так, щоб користувачам було легко зорієнтуватися. Навігація по системі повинна бути інтуїтивно зрозумілою та швидкою. Меню, пошукові функції та посилання повинні спрощувати доступ до різних частин системи.

Взаємодія користувача з системою повинна бути інтуїтивно зрозумілою та привабливою. Використання анімацій, відзначень та інших елементів візуалізації підвищує залученість та робить взаємодію більш приємною.

Форми реєстрації, авторизації та інші елементи введення даних повинні бути зручними та легкими для користувача. Валідація даних та відповідні повідомлення про помилки сприяють уникненню непорозумінь. Реалізація функціоналу повинна відповідати вимогам користувачів та бути логічною. Програмування функцій та їх впровадження повинні дотримуватися сучасних стандартів та кращих практик розробки.

### 2.7 Проектування бази даних

Бази даних є невід'ємною частиною інформаційних систем, надаючи зручний та ефективний спосіб зберігання, організації, доступу і керування даними. Вони використовуються для забезпечення структурованого та безпечного зберігання інформації, що використовується в різних аспектах бізнесу та технології.

У контексті розробки систем керування контентом та систем користувацьких комунікацій, обрання відповідної бази даних є важливим завданням. Вибір бази даних залежить від конкретних потреб проекту, вимог до продуктивності, масштабуваності та типів даних, які будуть зберігатися і оброблятися. Для нашої системи будемо обирати серед реляційних та NoSQL баз даних.

Реляційні бази даних (RDBMS) [25] є одними з найпоширеніших та найбільш використовуваних систем у сфері зберігання та управління даними в сучасному світі. Їх популярність пояснюється їхньою структурованістю, надійністю та високою продуктивністю.

Основна ідея реляційних баз даних полягає в тому, щоб зберігати дані у вигляді таблиць, де дані організовані у рядки та стовпці. Кожна таблиця представляє собою окремий тип даних, і для кожного стовпця таблиці визначається його тип та обмеження (наприклад, цілі числа, рядки, дати тощо).

Ця структурованість дозволяє зберігати дані систематично та зв'язувати їх між собою, встановлюючи відносини між таблицями. Наприклад, велика таблиця з інформацією про клієнтів може бути пов'язана з таблицею замовлень через

унікальний ідентифікатор клієнта, що дозволяє легко отримувати інформацію про замовлення конкретного клієнта.

Реляційні бази даних забезпечують цілісність та консистентність даних. Це означає, що вони гарантують, що дані зберігаються у відповідності до визначених правил та обмежень. Наприклад, якщо в таблиці визначено, що стовпець "Вік" повинен містити цілі числа, система не дозволить зберегти в цьому стовпці нечислове значення.

Також реляційні бази даних дозволяють встановлювати зв'язки між таблицями та створювати обмеження щодо дій з даними. Наприклад, можна визначити обмеження, яке не дозволить видалити клієнта, який має активні замовлення, щоб уникнути втрати даних.

Один із ключових аспектів реляційних баз даних - це мови запитів, такі як SQL (Structured Query Language). SQL надає можливість виконувати різноманітні операції з даними, такі як вибірка (SELECT), вставка (INSERT), оновлення (UPDATE) та видалення (DELETE).

Ця мова дозволяє легко отримувати необхідну інформацію з бази даних, фільтрувати та сортувати дані за різними критеріями, а також об'єднувати дані з різних таблиць за допомогою з'єднань (JOIN). На малюнку 2.11 зображено приклад роботи з SQL базою даних.

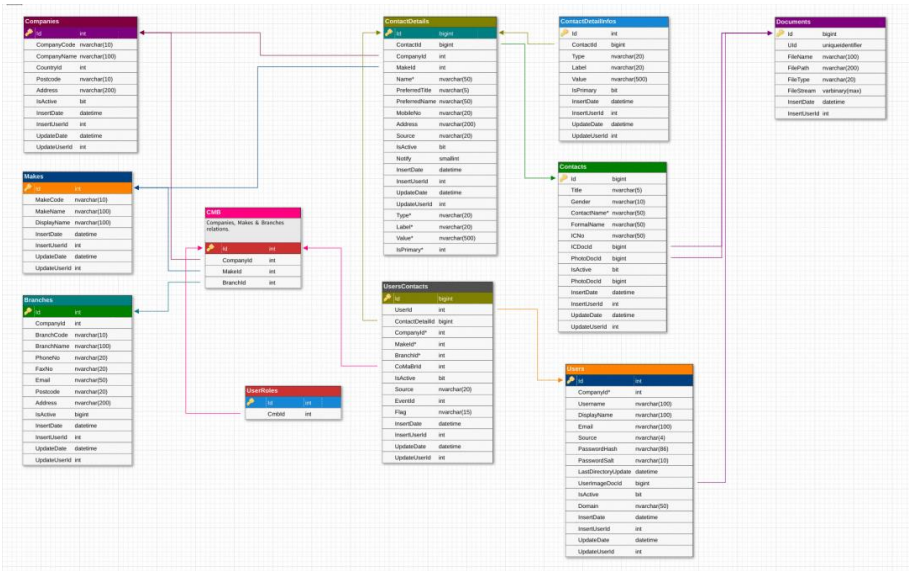


Рисунок 2.11 – Вигляд шаблону Microsoft SQL Database

Реляційні бази даних відомі своєю надійністю та можливістю відновлення даних у випадку виникнення помилок чи аварій. Багато RDBMS підтримують резервне копіювання, запис операцій та механізми відновлення, що забезпечують захист від втрати даних.

NoSQL бази даних є інноваційним підходом до зберігання даних, що забезпечує гнучкість та ефективність в роботі з неструктурованими та півструктурованими даними [26]. По суті, вони розширюють можливості традиційних реляційних баз даних та створюють нові можливості для розробників та архітекторів даних.

NoSQL бази даних відомі своєю гнучкістю та здатністю зберігати дані різних типів без необхідності дотримуватися жорсткої схеми даних. Це означає, що вони можуть зберігати не тільки табличні дані, але й документи, графи, ключ-значення, часові ряди та багато інших типів даних.

Наприклад, документ-орієнтовані NoSQL бази даних, такі як Firebase, дозволяють зберігати дані у вигляді JSON-подібних документів. Це особливо корисно для розробки веб-додатків та систем управління контентом, де структура даних може змінюватися з часом.

NoSQL бази даних добре підходять для сучасних застосунків, що вимагають великого обсягу даних та високої продуктивності. Вони підтримують горизонтальне масштабування, що означає можливість додавання нових серверів та ресурсів для обробки зростаючого навантаження.

Однією з популярних категорій NoSQL баз даних є графові бази даних, такі як Neo4j. Вони оптимізовані для зберігання та роботи з даними, які мають складні взаємозв'язки, такі як соціальні графи, мережі та рекомендаційні системи.

Ключ-значення бази даних, наприклад, Redis, використовуються для швидкого зберігання і отримання пар "ключ-значення" і є ідеальними для кешування даних та виконання операцій у реальному часі. NoSQL бази даних зазвичай не мають складних схем даних, що дозволяє розробникам бути більш

продуктивними та експериментувати зі структурою даних. Вони не вимагають визначення таблиць та зв'язків, що спрощує розробку та дозволяє швидше вносити зміни. На рисунку 2.12 зображені основні категорію баз даних.

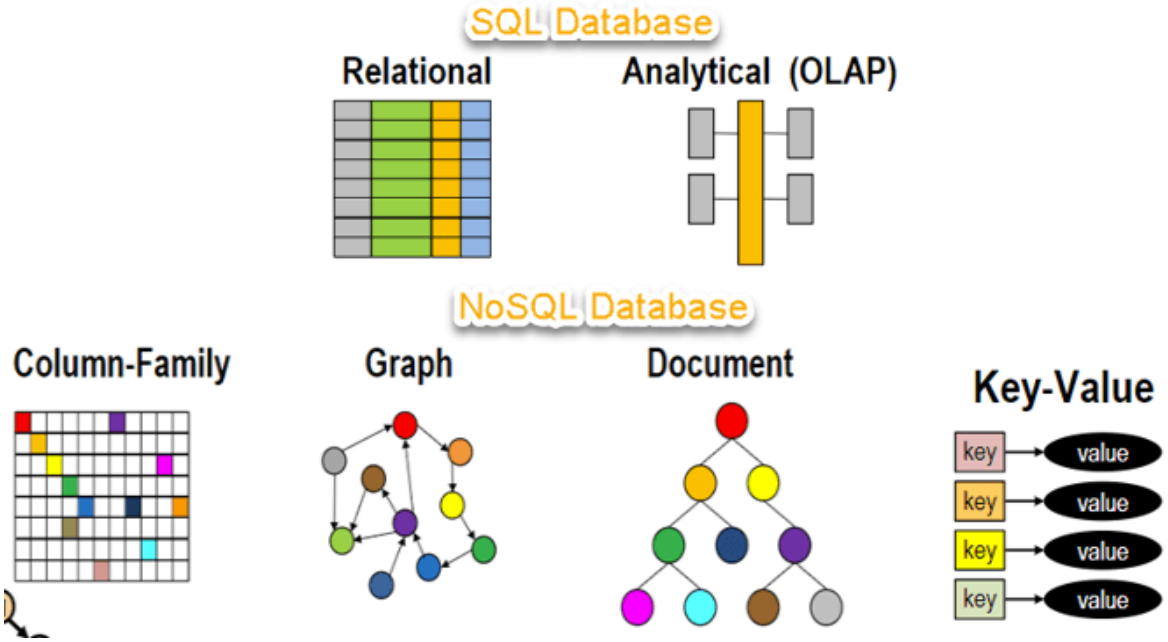


Рисунок 2.12 – Основні категорії баз даних

Обрання відповідної бази даних є ключовим етапом при розробці системи керування контентом (CMS) та системи користувацьких комунікацій. Очевидно, що обрана база даних повинна відповідати потребам проекту та забезпечувати оптимальну продуктивність та надійність. У нашому випадку було обрано Firebase.

Firebase є представником NoSQL баз даних, зокрема документ-орієнтованих баз даних. Однак він вирізняється з середини інших NoSQL систем та надає кілька переваг для систем керування контентом та користувацьких комунікацій:

Firebase дозволяє зберігати дані у вигляді JSON-подібних документів, що ідеально підходить для систем, де схема даних може змінюватися з часом. Це дає можливість додавати та модифікувати дані без необхідності перезавантаження бази даних або редагування таблиць.

Також ця база даних підтримує горизонтальне масштабування, що дозволяє обробляти великий обсяг даних та високі завантаження. Це важливо для систем, які мають велику активність користувачів та великі обсяги контенту. Firebase також відомий своєю швидкодією завдяки індексації даних та кешуванню. Це дозволяє забезпечити користувачам швидкий доступ до контенту та комунікацій.

Порівняємо Firebase з деякими аналогами баз даних, такими як MySQL, PostgreSQL та Cassandra, за кількома ключовими параметрами:

Таблиця 2.1 – Порівняльні характеристики програмних продуктів

| Параметр | Firebase | MySQL | PostgreSQL | Cassandra |
|---|---|---|---|---|
| Тип бази даних | NoSQL (документ-орієнтований) | Реляційна | Реляційна | NoSQL (колоночна) |
| Гнучкість схеми даних | + | - | - | + |
| Масштабованість | + | + | + | + |
| Масштабованість (горизонтальне масштабування) | + | - | + | + |
| Продуктивність | + | + | + | + |
| Можливості запитів та фільтрації даних | + | + | + | + |
| Спільнота та документація | + | + | + | - |
| Витрати ресурсів | + | + | + | + |
| Підсумковий результат | 7 | 5 | 6 | 6 |

Ця таблиця надає загальну інформацію про різні бази даних та їхні можливості. Firebase вирізняється гнучкістю схеми даних, горизонтальним масштабуванням, гарною продуктивністю та гнучкими можливостями запитів.

Враховуючи всі ці переваги, Firebase видається оптимальним вибором для розробки нашої системи керування контентом та користувацькими комунікаціями. Він поєднує гнучкість NoSQL бази даних з високою продуктивністю та надійністю, що робить його ідеальним варіантом для наших потреб.

### 2.8 Розробка загального алгоритму роботи програми

Щоб створення додатку колективної інформаційної системи необхідно розробити загальний алгоритм роботи додатку (рисунок 2.13).

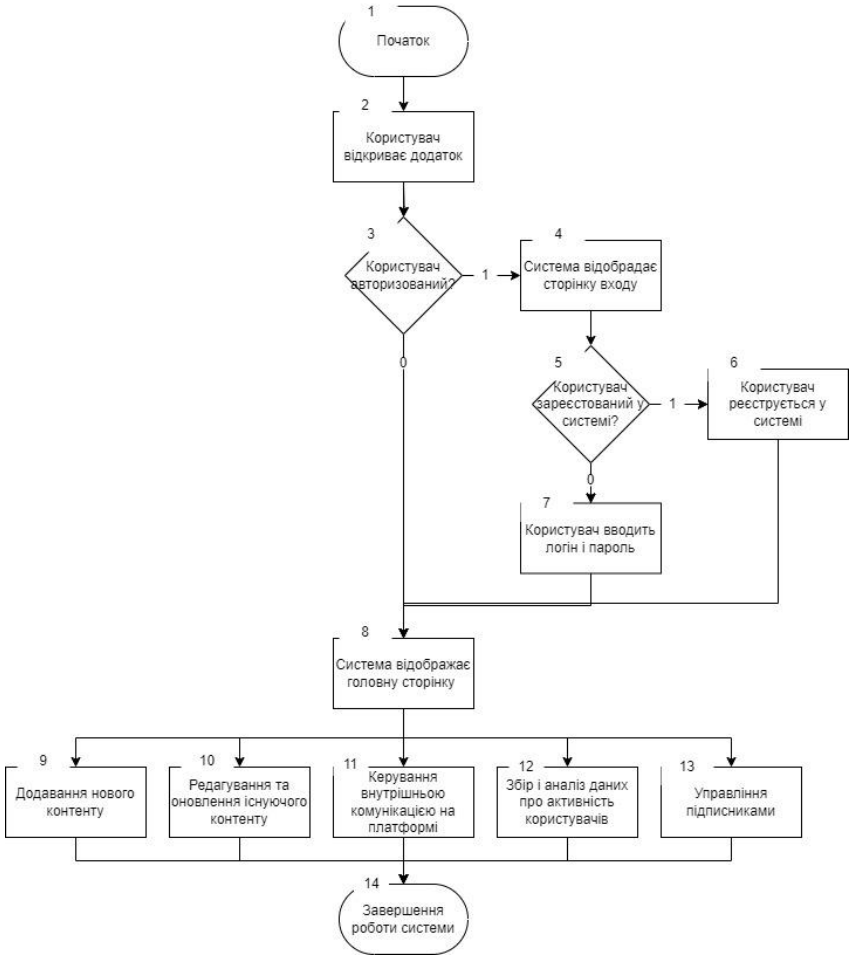


Рисунок 2.13 – Блок-схема загального алгоритму роботи додатку

Загальний алгоритм роботи додатку має такі кроки: користувач відкриває додаток, перевірка чи користувач авторизований, відображення сторінки входу, вхід або реєстрація користувача у системі, відображення головної сторінки, додавання нового контенту, редагування на оновлення існуючого контенту, керування внутрішньою комунікацією на платформі, збір і аналіз даних про активність користувачів, управління підписниками.

Спочатку користувач вводить особисті дані та обирає унікальне ім'я користувача для реєстрації. Система перевіряє введені дані на коректність та унікальність. У разі успішної реєстрації користувач отримує підтвердження та може увійти в систему. Користувач вводить свої дані для входу в систему. Система перевіряє логін та пароль. У випадку успішної авторизації, користувач отримує доступ до особистого кабінету. Користувач має можливість переглядати різні розділи системи, такі як "Головна", "Профіль", "Контент", тощо. Навігаційне меню дозволяє легко переміщатися між різними секціями та функціями. Користувач може створювати, редагувати та видаляти свій контент, такий як статті, зображення, аудіо та відео. Система забезпечує зручний інтерфейс для редагування та форматування контенту. Система аналізує поведінку користувачів для надання рекомендацій та персоналізованого контенту. Аналітичні засоби дозволяють адміністраторам відстежувати та аналізувати популярність функцій та контенту. Всі дані передаються зашифровано, забезпечуючи конфіденційність під час транспорту. Механізми аутентифікації та авторизації дозволяють контролювати доступ до ресурсів.

Система отримала значне удосконалення завдяки реформованому методу інтеграції із соціальними мережами. Це нововведення не лише дозволяє користувачам авторизуватися з використанням облікових записів у соціальних мережах, а й розширює можливості взаємодії між інформаційною системою та їхніми соціальними профілями. Цей перехід відкриває безліч нових можливостей та додає значний функціонал.

Крім того, реалізація авторизації через соцмережі спрощує процес входу для користувачів і робить його більш зручним та доступним. Це може сприяти

зростанню кількості зареєстрованих користувачів, оскільки багато людей вже мають облікові записи у соціальних мережах.

Оновлений метод інтеграції інформаційної системи з соціальними мережами є значним кроком уперед у покращенні користувацького досвіду та розширенні можливостей взаємодії між платформою та її користувачами.

Однією з ключових переваг цього удосконаленого методу є спрощена авторизація через облікові записи у соціальних мережах. Користувачам тепер не потрібно запам'ятовувати нові паролі або витрачати час на реєстрацію. Авторизація через соцмережі робить вхід на платформу швидким та зручним, що може значно підвищити кількість зареєстрованих користувачів.

У контексті подальшого розвитку інформаційної системи, важливим етапом стало впровадження методу аналізу та відстеження користувацької активності. Цей метод забезпечує високий рівень інтелектуалізації та оптимізації системи шляхом глибокого розуміння взаємодії користувачів з контентом, а також надає можливість постійного удосконалення інтерфейсу для забезпечення кращої взаємодії. В процесі взаємодії користувачів з системою, здійснюється збір та зберігання різноманітних даних, таких як час відвідування, перегляди сторінок, взаємодія з елементами інтерфейсу, тощо. Для збору та зберігання даних було використано Firebase Realtime Database. Це дозволяє ефективно обробляти та зберігати великі обсяги структурованих та неструктурованих даних. Зібрані дані аналізуються з метою виявлення патернів та поведінки користувачів. Це дозволяє системі розуміти, як користувачі взаємодіють з контентом та які аспекти інтерфейсу є найбільш привабливими чи ефективними. Для аналізу поведінки користувачів використовувався Google Analytics, інструмент з веб-аналітики. Цей інструмент дозволяє визначити ключові метрики та проводити детальний аналіз взаємодії користувачів з різними елементами системи. На основі аналізу формується індивідуальний профіль кожного користувача, що включає в себе його вподобання, інтереси та ступінь активності. Цей профіль використовується для персоналізації взаємодії та контенту. Для формування та оновлення профілю користувача

використовувалися механізми аутентифікації та авторизації, такі як Firebase Authentication та OAuth. Інформація про користувача зберігалася в базі даних разом із засобами для персоналізації. За допомогою отриманих даних система реалізує динамічну адаптацію інтерфейсу, що дозволяє реагувати на зміни в уподобаннях та потребах користувачів миттєво, забезпечуючи найкращий досвід взаємодії. Внесені зміни у систему піддаються тестуванню та валідації, забезпечуючи, що вони не лише оптимізують користувацький досвід, а й не впливають негативно на функціональність. Для тестування та валідації використовувалися різні методи, включаючи юніт-тестування, автоматизовані тести та A/B тести.

Також додатково було розроблено блок-схему роботи з базою даних, вона зображена на рисунку 2.14.

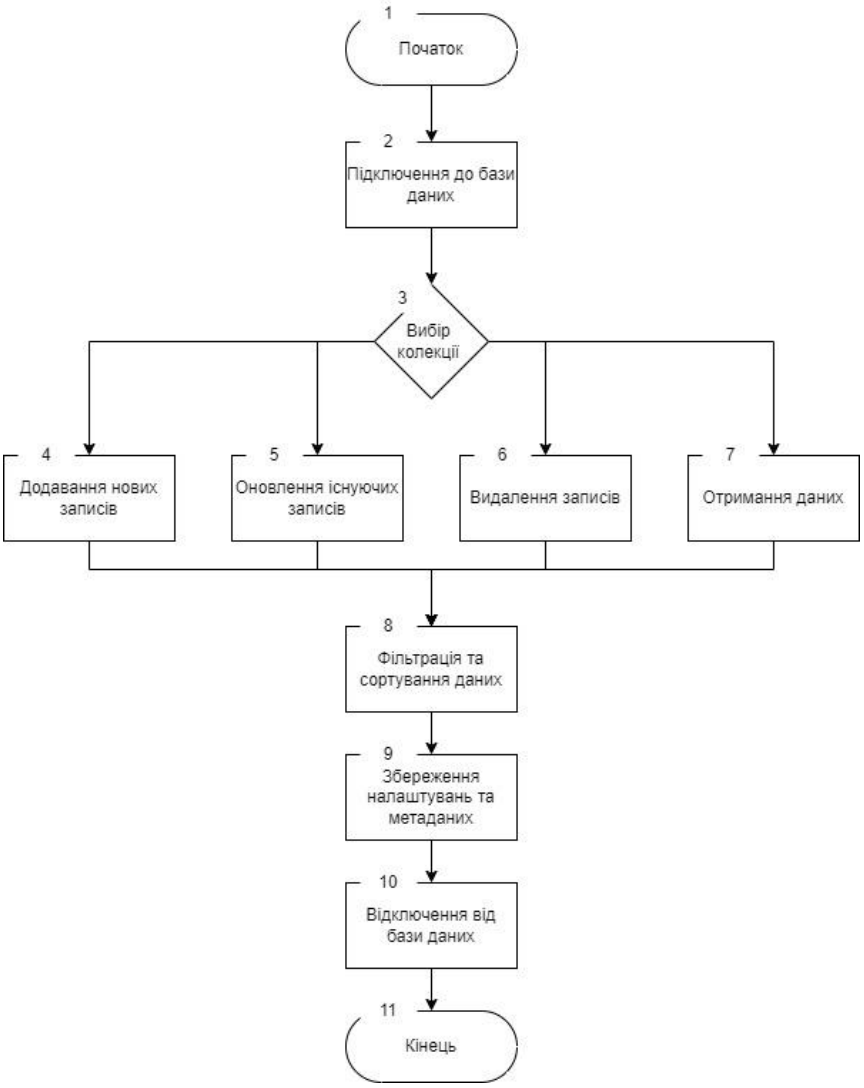


Рисунок 2.14 – Блок-схема роботи з базою даних

Додатково було створено UML діаграму яка представлена на рисунку 2.15

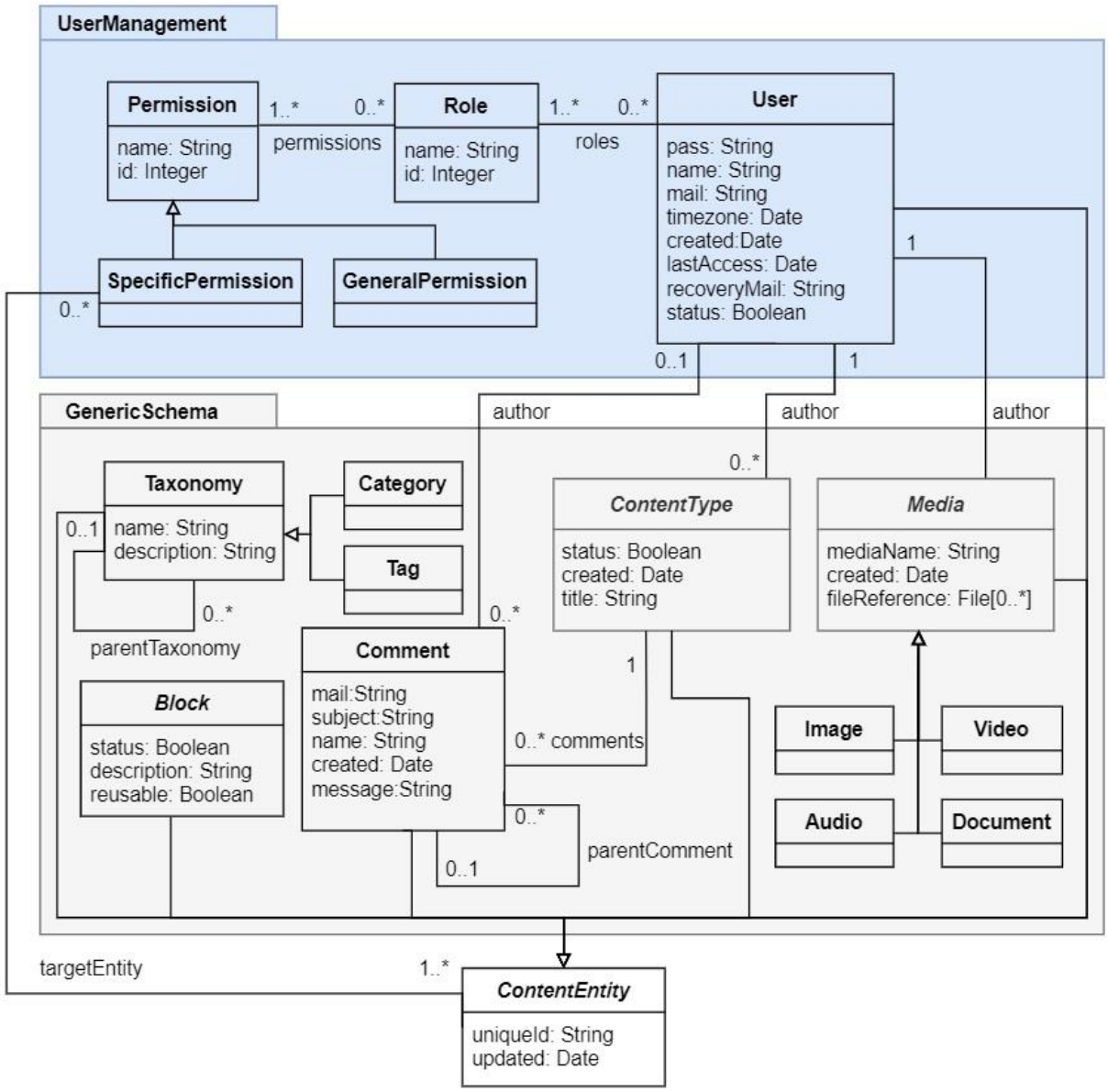


Рисунок 2.15 – UML діаграма системи

## 2.9 Висновки

В другому розділі магістерської роботи розглянути методи спільної фільтрації та фільтрації за змістом, запропоновано гібридний метод, розроблена архітектура системи керуванням контенту з користувацькими комунікаціями. Спроектовано базу даних. Розроблено загальний алгоритм роботи програми.

# 3. РЕАЛІЗАЦІЯ ПРОГРАМИ ІНФОРМАЦІЙНОЇ СИСТЕМИ КЕРУВАННЯ КОНТЕНТОМ

## 3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу

Для розробки клієнтської частини веб-додатку розглядались наступні найбільш вживані для цього мови програмування: JavaScript, Python, Java.

JavaScript – це мова програмування високого рівня, яка відповідає стандарту ECMAScript. Вона виділяється динамічною типізацією, об’єктно-орієнтованістю та використанням прототипів для роботи з об’єктами. Основні риси JavaScript включають в себе функції вищого порядку, можливість використання як імперативного, так і функціонального стилів програмування.

Ця мова програмування розроблена для використання в браузерах для взаємодії з об'єктною моделлю документа (DOM). Однак її механізми використовуються не тільки в браузерах, але й на серверах завдяки платформі Node.js.

JavaScript широко використовується для різних завдань веб-розробки. Сценарії на цій мові можуть забезпечувати динамічне завантаження контенту, створення анімацій, розробку браузерних ігор, керування медіапотоками, створення спливаючих оголошень та інше.

Наприклад, важливими сценаріями є зміна вмісту веб-сторінок без перезавантаження, анімація елементів, графіка в браузерних іграх, а також перевірка та обробка даних введених користувачем у веб-формах перед їх відправкою на сервер.

Багато веб-сайтів використовують сторонні бібліотеки та фреймворки для розробки клієнтської частини. Наприклад, jQuery, React і Angular є популярними інструментами для спрощення розробки та покращення продуктивності.

У світі веб-розробки термін "Vanilla JS" використовується для позначення коду, який написаний на чистому JavaScript без використання сторонніх

бібліотек чи фреймворків.

Основні особливості мови JavaScript:

JavaScript є імперативною та структурованою мовою програмування, де велику частину синтаксису взято з C. Вона підтримує ключові конструкції, такі як умови if, цикли while, switch, do while, але при цьому розвивається, додаючи нові можливості. Наприклад, введення блочного області видимості з ключовими словами let і const у стандарті ECMAScript 2015.

JavaScript володіє слабкою типізацією, де типи не завжди потрібно явно вказувати, і вони можуть неявно приводитися залежно від контексту використання. Це робить мову гнучкою та легкою для використання, але водночас вимагає уваги до типів при розробці більших проєктів.

Динамічна типізація в JavaScript означає, що тип змінної пов'язується з її значенням, а не зі змінною самою. Це дозволяє змінювати типи значень в процесі виконання програми, що забезпечує велику гнучкість, але також може призводити до непередбачуваного поведінки.

Об'єктно-орієнтований підхід в JavaScript визначається унікальним підходом до реалізації об'єктної моделі, що відрізняється від традиційних мов програмування, таких як Java або C++. У цьому контексті, об'єкти у JavaScript базуються на прототипах, а не на класах, надаючи мові своєрідну гнучкість та функціональність. У JavaScript об'єкти можуть слугувати прототипами для інших об'єктів, дозволяючи наслідувати властивості та методи. Коли властивість або метод не знаходиться в об'єкті, JavaScript автоматично шукає їх в його прототипі. Однією з основних особливостей є можливість створювати об'єкти без явного визначення класу. Прототипи дозволяють динамічно додавати та змінювати властивості об'єктів на льоту. JavaScript дозволяє динамічно додавати та змінювати властивості об'єктів протягом виконання програми. Це може спрощувати роботу з даними та дозволяти адаптувати об'єкти під конкретні потреби.

Замикання є потужним та важливим аспектом JavaScript, яке дозволяє функціям зберігати доступ до змінних зовнішнього контексту, навіть після

завершення виклику функції. Це створює замкнуті (замкнені) області видимості, що забезпечує контроль над тим, як змінні використовуються та зберігаються. Однією з ключових переваг замикань є їхню здатність зберігати стан. Функція, створена замиканням, може запам'ятовувати значення змінних на момент свого створення, навіть якщо вона викликається в іншому місці. Замикання дозволяють створювати приватні змінні, які недоступні зовнішнім функціям. Це сприяє створенню інкапсульованих модулів.

Асинхронність в JavaScript дозволяє виконувати операції, які займають багато часу, такі як завантаження файлів чи взаємодія з сервером, без блокування виконання інших операцій. Основним інструментом для роботи з асинхронним кодом є колбеки. Колбек - це функція, яка передається як аргумент до іншої функції та викликається пізніше після завершення певної операції або події. Колбеки використовуються для обробки результатів асинхронних викликів та забезпечення правильного порядку виконання коду. Callback-підходи використовуються для обробки подій, зчитування файлів, виклику API-запитів та інших асинхронних задач. Вони гарантують, що код виконається тільки після завершення конкретної операції, забезпечуючи ефективне використання ресурсів та уникнення блокування виконання. Callback-функції часто використовуються в функціях з зворотнім викликом подій, обробниках завантаження, таймаутах, та інших сценаріях, де важливо визначити, що робити після завершення асинхронної задачі. Цей підхід забезпечує вищий рівень паралелізму та ефективності у виконанні програм.

Можливість використання JavaScript у серверному середовищі використовуючи Node.js. Node.js є відкритою та ефективною платформою, побудованою на движку V8, розробленому Google. Вона дозволяє використовувати JavaScript для написання серверного коду. При цьому використовується однопоточна, подійно-орієнтована архітектура, яка забезпечує високу продуктивність та масштабованість [27].

Python - це високорівнева, інтерпретована, об'єктно-орієнтована мова програмування, яка була створена Гвідо ван Россумом та вперше випущена у

1991 році. Python зазнавав кілька важливих релізів, таких як Python 2.x та Python 3.x. Перехід на Python 3 був започаткований в 2008 році та призвів до несумісності між версіями, але сприяв покращенню мови та виправленню багатьох її недоліків. При цьому спільнота Python активно розвивається, вносячи нововведення та виправлення помилок. Python знайшов широке застосування в різних областях, включаючи розробку веб-додатків, науку про дані, штучний інтелект, автоматизацію та багато інших [28].

Python визначається своєю читабельністю і простотою синтаксису, сприяючи розробці зрозумілого та легко обслуговуваного коду. Мова підтримує багато парадигм програмування, включаючи імперативне, функціональне та об'єктно-орієнтоване програмування.

Однією з ключових особливостей Python є динамічна типізація, що дозволяє змінювати типи змінних під час виконання програми. Це полегшує розробку та зменшує кількість коду, але може призводити до помилок в час виконання.

Ще однією сильною стороною мови є велика стандартна бібліотека, яка включає в себе різноманітні інструменти та модулі для різних завдань, що сприяє розвитку екосистеми Python.

Недоліками Python можуть бути швидкодія та великий обсяг пам'яті, використовуваний програмами на цій мові, що може бути недоцільним для деяких вбудованих систем чи завдань, де критична продуктивність. Також важко підтримувати потокову обробку на рівні процесів через межу потоків GIL (Global Interpreter Lock).

Особливості мови програмування Python:

Читабельний синтаксис: Синтаксис Python призначений для того, щоб бути чітким і лаконічним, сприяючи легкості читання та розуміння коду.

Динамічна типізація: Python використовує динамічну типізацію, що дозволяє змінювати типи змінних під час виконання програми. Це полегшує розробку та зменшує кількість коду, але може вимагати уваги до типів при роботі з великими проектами.

Об'єктно-орієнтоване програмування (ООП): Python підтримує об'єктно-орієнтовану парадигму, що дозволяє розробляти програми з використанням об'єктів та класів для структурування коду.

Високорівневий мовний рівень: Python є високорівневою мовою програмування, що дозволяє програмістам виражати алгоритми меншою кількістю рядків коду порівняно з іншими мовами.

Широка стандартна бібліотека: Мова поставляється з обширною стандартною бібліотекою, що включає в себе різні інструменти і модулі для розв'язання різних задач, від мережевого програмування до обробки зображень.

Крос-платформенність: Python може працювати на різних операційних системах, таких як Windows, macOS та різні варіанти Linux, що робить його популярним в різних середовищах.

Інтерпретованість: Python є інтерпретованою мовою, що означає, що ви можете виконувати код без необхідності компіляції. Це полегшує тестування та швидше виконання прототипів.

Розширюваність: Python легко поєднується з кодом, написаним на C та C++, що дозволяє використовувати швидкодіючі бібліотеки та розширювати мову.

Python має багато різноманітних фреймворків, які розширюють можливості мови та спрощують розробку різних видів програм. Ось кілька ключових фреймворків Python та їх характеристики:

Django: Django є одним з найпопулярніших веб-фреймворків Python. Він надає високорівневий інструментарій для швидкої розробки стабільних та безпечних веб-додатків. Django має вбудовану адміністративну панель, ORM (Object-Relational Mapping) для роботи з базами даних, та вбудовані засоби безпеки.

Flask: Flask є легким та гнучким мікрофреймворком, який дозволяє швидко розробляти веб-додатки. Він має менше вбудованих функцій, ніж Django, але це дозволяє розробникам обирати та використовувати різноманітні розширення для своїх потреб.

Flask-RESTful та FastAPI: Ці фреймворки спеціалізуються на розробці RESTful API. Flask-RESTful є розширенням Flask, тоді як FastAPI є відносно новим, але швидким та зручним фреймворком для створення API на основі стандартів OpenAPI та JSON Schema.

Pyramid: Pyramid є фреймворком, який ставить більший акцент на гнучкість та масштабованість. Він підходить для розробки різних видів веб-додатків, від простих до складних та масштабованих.

Tornado: Tornado - це асинхронний веб-фреймворк, спроектований для високопродуктивних застосунків, таких як чати, стрімінгові сервіси та інші застосунки, що вимагають обробки багатьох одночасних з'єднань.

Java - це високорівнева, об'єктно-орієнтована мова програмування, яка заснована на концепціях "write once, run anywhere" (WORA) і "принципі найменшого дозволеного доступу" (principle of least privilege). Розроблена компанією Sun Microsystems (пізніше придбаною Oracle Corporation), Java вперше була випущена в 1995 році і швидко стала однією з найпопулярніших мов програмування завдяки своїй переносимості та масштабованості [29].

Історія Java сягає середини 1990-х, коли в компанії Sun Microsystems під керівництвом Джеймса Гослінга та його команди розпочалася розробка нової мови програмування для вбудованих систем. Вони хотіли створити мову, яка була б ефективною, переносною та простою для вивчення. На початку 1990-х ця робота вилилася в мову Oak, яка пізніше була перейменована в Java.

Java привернула увагу своєю здатністю працювати на будь-якій платформі, що мала віртуальну машину Java (JVM). Це дозволяє розробникам писати програмний код один раз і запускати його на різних платформах без потреби перекомпіляції. Ця ідея стала основою принципу "write once, run anywhere".

Однією з ключових особливостей Java є її об'єктно-орієнтована природа. Все в Java є об'єктом, що сприяє модульності, повторному використанню коду та полегшенню розробки складних систем. Мова також підтримує автоматичне управління пам'яттю, що дозволяє розробникам уникнути багатьох проблем, пов'язаних з витіканням пам'яті та десеріалізацією об'єктів.

Однак, також існують недоліки в мові Java. Деякі розробники вказують на те, що Java може бути більш "масштабованою" і менш "зручною" у порівнянні з іншими мовами, такими як Python чи JavaScript. Деякі вважають, що високий рівень абстракції, хоча і полегшує роботу з багатьма завданнями, може також впливати на продуктивність в деяких випадках.

Мова продовжує розвиватися, оновлюючи свої можливості та додаючи нові функції. Java підтримує широкий спектр бібліотек і фреймворків, що робить її популярною у розробці великих корпоративних систем.

Загалом, Java залишається однією з найважливіших мов програмування в індустрії та знаходить застосування в різних сферах, включаючи розробку мобільних додатків (за допомогою Android SDK), веб-розробку, корпоративні системи та вбудовані пристрої.

Головні особливості мови програмування Java:

Об'єктно-орієнтована мова: Java є повністю об'єктно-орієнтованою мовою програмування. Це означає, що все в Java є об'єктом, включаючи типи даних та функції. Об'єктно-орієнтований підхід сприяє модульності, повторному використанню коду та полегшує розробку складних систем.

Платформонезалежність: Java працює на принципі "write once, run anywhere". Код, написаний на Java, може бути виконаний на будь-якій платформі, яка має відповідну віртуальну машину Java (JVM). Це забезпечує переносимість програм між різними операційними системами та апаратними платформами.

Віртуальна машина Java (JVM): Java використовує віртуальну машину (JVM), яка виконує Java-байткод, скомпільований з програмного коду. Це дозволяє виконувати Java-програми на будь-якій платформі, яка підтримує JVM, не вимагаючи перекомпіляції.

Автоматичне управління пам'яттю: Java використовує систему автоматичного управління пам'яттю, відому як "сбірка сміття". Це дозволяє програмістам уникати багатьох проблем, пов'язаних з ручним виділенням та звільненням пам'яті, такими як витікання пам'яті та десеріалізація об'єктів.

Безпека: Java вбудовує ряд заходів безпеки, що дозволяє створювати безпечні програми. Віртуальна машина Java дозволяє відокремлювати виконуваний код від операційної системи, а механізми безпеки контролюють доступ до ресурсів.

Велика екосистема: Java має великий вибір бібліотек та фреймворків, що полегшує розробку програм. Велика активна спільнота розробників сприяє високій якості інструментів і ресурсів для розвитку проектів на Java.

Java має кілька потужних фреймворків для веб-розробки, які полегшують створення масштабованих та ефективних веб-додатків. Нижче представлено огляд декількох ключових фреймворків для веб-розробки на Java:

Spring Framework: Spring є одним з найпопулярніших та впливових фреймворків для веб-розробки на Java. Він пропонує обширний набір інструментів для розробки веб-додатків, включаючи управління залежностями, обробку подій, транзакції та безпеку. Spring MVC використовується для створення веб-слою додатків.

JavaServer Faces (JSF): JSF - це фреймворк для будівництва веб-інтерфейсів, який використовується для створення компонент-орієнтованих веб-додатків. Він надає готові компоненти для швидкої розробки та полегшує взаємодію з серверними ресурсами.

Apache Struts: Struts - це фреймворк, який використовує шаблон проектування Model-View-Controller (MVC) для побудови веб-додатків. Він надає різні інструменти для управління формами, обробки подій та навігації між сторінками.

Play Framework: Play - це модернізований фреймворк для веб-розробки, який відзначається високою продуктивністю та простотою використання. Він підтримує асинхронний код та вбудовані засоби розробки, що дозволяють швидко розгортати та впроваджувати додатки.

У таблиці 3.1 зведено результати порівняльного аналізу мов програмування.

Таблиця 3.1 — порівняння мов програмування

| Особливість | JavaScript | Python | Java |
|---|---|---|---|
| Синтаксис | Легкий для вивчення, особливо для веб-розробки. (+) | Простий та експресивний, сприяє швидкості розробки. (+) | Важкий, але строгий, що дозволяє попереджати помилки на етапі компіляції. (+/-) |
| Вибір фреймворків | Широкий вибір (React, Angular, Vue.js) для різних потреб. (+) | Є багато фреймворків (Django, Flask), але менший вибір порівняно з JS. (+/-) | Розширений вибір (Spring, Hibernate, Struts) для великих та складних проектів. (+) |
| Використання на веб-сторінках | Широке використання в браузерах, включаючи взаємодію з DOM. (+) | Обмежене використання, частіше для серверного програмування. (+/-) | Менше використовується на клієнтському боці, частіше використовується на сервері. (+/-) |
| Взаємодія з браузером | Найкраще підходить для веб-розробки, має доступ до DOM. (+) | Не підходить для клієнтської веб-розробки, але може використовуватися на сервері. (-) | Може використовуватися на клієнті та сервері, але менше популярний на клієнтському боці. (+/-) |
| Версіонування пакетів | Завдяки npm та yarn, легко керувати версіями пакетів. (+) | Використовує pip для керування пакетами, але менше гнучко. (+/-) | Має Maven і Gradle для керування залежностями, але менше інструментів у порівнянні з JS. (+/-) |
| Асинхронний код | Використовує event loop для асинхронності, підтримує Promise. (+) | Підтримує асинхронне програмування, включаючи async/await (+) | Має потужний механізм для асинхронного програмування, але менше сучасний порівняно з JS та Python. (+) |

Продовження таблиці 3.1

| | | | |
|---|---|---|---|
| Продуктивність | Висока продуктивність завдяки використанню JIT-компіляції. (+) | Інтерпретована мова, тому менш продуктивна у порівнянні з Java. (+) | Висока продуктивність завдяки використанню віртуальної машини та компіляції в байт-код. (+) |
| Розробка веб-застосунків | Ідеально підходить для розробки веб-застосунків та односторінкових додатків. (+) | Хороший вибір для веб-розробки, але менше популярний на клієнтському боці. (+/-) | Використовується для розробки великих корпоративних веб-застосунків та ентерпрайз рішень. (+/-) |
| Підсумок | 8 | 5 | 5.5 |

Як видно з таблиці JavaScript має перевагу над Python і Java, тому для реалізації програмного додатку було обрано саме це середовище, адже воно задовольняє всі необхідні вимоги для створення системи керуванням контенту та має великий набір інструментів, що грає значну роль в розробці програмних продуктів.

JavaScript фреймворки стали необхідним елементом веб-розробки, враховуючи зростання складності та обсягу проектів. Вони спрощують розробку, забезпечуючи готовий фундамент і стандартизовану архітектуру, що дозволяє ефективніше управляти кодом, підтримувати його та забезпечувати швидший розвиток додатків. Фреймворки дозволяють використовувати кращі практики, стандартні рішення та сприяють створенню масштабованих, стабільних та легко розширюваних веб-додатків. З огляду на це, для розробки були вирішено обрати фреймворк один з трьох найпопулярніших JavaScript фреймворків: React, Vue та Angular.

React – це бібліотека для створення інтерфейсів користувача від Facebook, яка використовується для розробки односторінкових веб-додатків та веб-сайтів. Він визначається своєю декларативною та компонентною природою, що

полегшує розробку та підтримку коду.

Особливості React:

- Компоненти: Базовий будівельний блок в React – це компонент, який є самодостатньою та перевикористовуваною частиною коду.
- Віртуальний DOM: React використовує віртуальний DOM для оптимізації оновлень та поліпшення продуктивності за рахунок ефективного взаємодії з реальним DOM.
- Декларативний Синтаксис: Реакт дозволяє описувати, як повинен виглядати інтерфейс в певний момент часу, і він відповідає за оновлення змін, не вимагаючи явного вказівання кожної зміни.
- Єдиностільний Джерело Правди (Single Source of Truth): Стан компонентів зберігається в одному місці (зазвичай в кореневому компоненті), що полегшує управління станом.
- Переваги React:
- Ефективність: Забезпечує високий рівень продуктивності завдяки віртуальному DOM та ефективним алгоритмам оновлення.
- Перевикористання коду: Компонентна архітектура дозволяє легко перевикористовувати та комбінувати частини коду.
- Велика екосистема: Багата екосистема бібліотек, інструментів та розширень, що полегшують розробку.
- Недоліки React:
- Складність серверного рендерінгу: Реалізація серверного рендерінгу може бути складною та вимагати додаткових налаштувань.

Vue.js – це прогресивний JavaScript фреймворк для створення користувацьких інтерфейсів. Його основна мета – забезпечити простий та гнучкий інструмент для розробки веб-інтерфейсів з підтримкою реактивних компонентів.

Особливості Vue.js:

- Легкість використання: Vue привертає розробників своєю простотою та легкістю вивчення, що дозволяє швидко інтегрувати його в проекти.

- Реактивність: Vue використовує систему реактивності, яка автоматично відслідковує зміни в стані та оновлює відображення відповідних компонентів.
- Компонентна архітектура: Побудований на ідеї компонентної розробки, дозволяючи легко перевикористовувати та організовувати код.
- Двостороннє зв'язування даних: Vue забезпечує простий спосіб двостороннього зв'язування даних, що спрощує роботу з формами та іншими елементами вводу.
- Переваги Vue.js:
- Легкість інтеграції: Легко інтегрується в існуючі проекти, не вимагаючи повного переписування коду.
- Приємна документація: Добре структурована та легко зрозуміла документація, що полегшує навчання та роботу з фреймворком.
- Прогресивний фреймворк: Використовуючи лише необхідні частини, розробники можуть поступово впроваджувати Vue в проекти.
- Недоліки Vue.js:
- Обмежена екосистема: У порівнянні з React та Angular, Vue має меншу кількість розширень та інструментів.
- Маленька кількість спільноти: Vue залучає все більше розробників, але спільнота все ще менша порівняно з React.

Angular – це великий та повноцінний фреймворк, розроблений і підтримуваний Google, призначений для створення складних веб-додатків. Він пропонує повноцінний набір інструментів та функцій для розробки, тестування та розгортання великих проектів [30].

Особливості Angular:

- Типізація та компіляція: Angular використовує TypeScript, що дозволяє використовувати статичну типізацію та використовувати ефективну компіляцію коду.
- Модульність: Організований за принципом модульності, Angular дозволяє легко розділяти функціонал на модулі для полегшення управління

проектами.

- Компонентна Архітектура: Angular застосунку будуються на компонентному підході, що полегшує перевикористання та роботу з окремими частинами додатка.
- RxJS та реактивне програмування: Angular використовує RxJS для реалізації реактивного програмування, що полегшує роботу з асинхронним кодом.

Переваги Angular:

- Повний функціонал: Angular надає всебічний функціонал для розробки, включаючи роутинг, HTTP-запити, стан додатка та інше.
- Сильна типізація: Використання TypeScript дозволяє виявляти та виправляти помилки на етапі компіляції.
- Широкий вибір інструментів: Angular пропонує велику кількість інструментів для тестування, розгортання та аналізу коду.
- Недоліки Angular:
- Складність вивчення: Завдяки своїй повноті та великому набору функцій, Angular може виглядати складним для новачків.
- Більший розмір коду: Велика кількість вбудованих функцій може призвести до більшого розміру вихідного коду.

Angular є потужним інструментом для розробки великих та складних веб-додатків, ідеально підходить для комерційних проектів з великим функціоналом та обсягом. Враховуючи ряд факторівів та особливостей фреймворків, було обрано фреймворк Angular для розробки системи керуванням контенту.

### 3.2 Розробка модулю керуванням контенту та рекомендацій

Першим кроком буде проектування архітектури модулю, визначення основних компонентів та їх взаємодії. Застосування компонентної архітектури Angular дозволяє розділити функціональність на незалежні елементи, що полегшує розробку та тестування.

Розробка інтерфейсу включала в себе створення компонентів для додавання, редагування та видалення контенту. Використання реактивних форм та елементів керування спрощує взаємодію з користувачем та забезпечує точність введених даних. Для швидкої та ефективної стилізації була використана бібліотека Bootstrap 5. Додано механізм категоризації для легкого класифікування контенту. Використання деревовидних структур дозволяє ефективно організовувати та швидко знаходити необхідний вміст.

Інтегровано редактор вмісту, який підтримує форматування тексту, додавання мультимедійних елементів та інші функції. Це забезпечує зручний інтерфейс для редагування контенту без необхідності використання зовнішніх редакторів.

Для створення гібридного методу рекомендацій необхідно реалізувати алгоритм спільної фільтрації на основі користувача. Спочатку необхідно обчислити подібність між цільовим користувачем та іншими користувачами. Потім вибираються користувачі з високою схожістю як набір найближчого сусіда. На основі схожих користувачів, можна запропонувати продукти, які можуть зацікавити.

Припустимо, що набір даних рекомендованої системи D{U, I, R}, де U = {u1, u2,. . . , un} — набір користувачів системи, I = {i1, i2, . . . , im} – набір елементів системи, і R – це матриця оцінки елементів користувача. Для набору даних із m користувачів і n елементів дані попередньо обробляються для отримання m × n рейтингової матриці елементів R(m × n), яка представлена рисунку 3.1. [12].

$$R = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1j} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2j} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ r_{i1} & r_{i2} & \cdots & r_{ij} & \cdots & r_{in} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mj} & \cdots & r_{mn} \end{pmatrix}$$

Рисунок 3.1 – Матриця рейтинга

У системі рекомендацій існує три основні методи, які використовуються

для розрахунку подібність між двома користувачами: подібність косинусів, схожість скоригованих косинусів і подібність Пірсона [31]. У цьому дослідженні буде використано подібність Пірсона, яка розрахована через загальні елементи рейтингу між будь-якими двома користувачами. Подібність Пірсона обраховується наступним чином:

$$P_{u,i} = \bar{R}_u + \frac{\sum_{v \in S(u,K) \cap N(i)} Sim_{\text{User}}(u,v) \times (R_{v,i} - \bar{R}_v)}{\sum_{v \in S(u,K) \cap N(i)} \mid Sim_{\text{User}}(u,v) \mid}$$

Рисунок 3.2 – Подібність Пірсона

де Ru,i та Rv,i представляють оцінки користувача u та користувача v щодо i-го пункту відповідно; R¯u та R¯v представляють середнє значення всіх оцінок користувача u та користувача v відповідно.

Перш ніж скласти рейтинговий список для формування рекомендації, необхідно визначити набір схожих сусідів цільового користувача. Подібний набір сусідів відноситься до набору користувачів, які мають подібні переваги з цільовим користувачем [32]. У системі рекомендацій найбільше K подібних користувачі зазвичай вибираються як найближчий набір сусідів для формування аналогічного набору сусідів цільового користувача. Після вибору набору сусідів цільового користувача він поєднується з усіма сусідніми рейтингами елементів і схожостями між користувачами, щоб передбачити оцінки цільового користувача на тестовому наборі. Рейтинговий список розраховується так [33]:

$$P_{u,i} = \bar{R}_u + \frac{\sum_{v \in S(u,K) \cap N(i)} Sim_{\text{User}}(u,v) \times (R_{v,i} - \bar{R}_v)}{\sum_{v \in S(u,K) \cap N(i)} \mid Sim_{\text{User}}(u,v) \mid}$$

де Pu,i представляє прогнозований рейтинг користувача u для невідомого елемента i; S(u, K) — множина K користувачів, найбільш схожих на користувача

u; і N(i) представляє набір користувачів, які оцінили елемент і. Після прогнозування рейтингу обирається N елементів із найвищим рейтингом і встановлюються як результати рекомендації цільовим користувачам [34].

Традиційний алгоритм спільної фільтрації розраховує матрицю схожості користувача на основі даних оцінки елементів, які легко піддаються впливу популярних елементів. Наприклад, «Втеча з Шоушенка» дуже хороший фільм. Якщо обидва користувачі А і користувач В поставили фільму «Втеча з Шоушенка» 5 балів, традиційний алгоритм спільної фільтрації прийде до висновку, що користувачі А і користувачі В мають високу схожість. Однак це не обов'язково так. Як відомо, однакова поведінка користувачів на популярних предметах не означає, що вони мають схожі інтереси. Навпаки, якщо два користувачі однаково поводилися з непопулярними предметами, більш імовірно, що їхні інтереси схожі. Наприклад, якщо обидва користувачі А і В переглянули відносно невелику кількість фільмів, таких як мюзикли, тоді можна вважати, що вони мають схожі інтереси. Таким чином, щоб усунути вплив популярних елементів на схожість користувачів, у цьому документі до традиційного алгоритму спільної фільтрації застосовано метод частоти термінів – інверсної частоти документів (TF-IDF), який використовується для нівелювання популярних елементів у список поведінки користувача. Основна причина використання методу TF-IDF полягає в тому, що він підходить для проблеми вилучення ваги. Крім того, метод TF-IDF є простим і легким для розрахунку [35].

TF-IDF — це статистичний метод, який часто використовують для оцінки важливості слова для файлу. Важливість слова прямо пропорційна кількості разів його появи у файлі, але в той же час вона обернено пропорційна частоті його появи в бібліотеці файлів [35]. На основі принципу TF-IDF пропонується вдосконалений метод обчислення схожості користувачів, щоб зменшити вагу впливу популярних елементів на схожість користувачів. Якщо елемент з'являється у списку поведінки користувача, але він також з'являється багато разів у списку поведінки інших користувачів, цей елемент вважається популярним, і його вплив на схожість користувача слід зменшити. Вага і-го

предмета в цьому документі розраховується як:

$$W_i = TF_i \times IDF_i = \frac{freq(i,u)}{|\ u\ |} \times \lg \frac{|\ U\ |}{(1 + popular(i))}$$

де f req(i, u) представляє кількість разів, коли і-й елемент з'являється в поведінці список користувача u; | u | представляє довжину списку поведінки користувача u; | U | представляє загальну суму кількість користувачів; і popular(i) представляє кількість разів, коли і-й елемент з'являється у всіх списках поведінки користувача. Потім вага предмета вводиться в рівняння подібності Пірсона  і вдосконалений метод обчислення подібності виглядає так:

$$Sim_{\text{User}}(u,v) = \frac{\sum_{i \in I} (R_{u,i} \times W_i - \bar{R}_u) \times (R_{v,i} \times W_i - \bar{R}_v)}{\sqrt{\sum_{i \in I} (R_{u,i} \times W_i - \bar{R}_u)^2 \sum_{i \in I} (R_{v,i} \times W_i - \bar{R}_v)^2}}$$

### 3.3 Розробка модулю користувацьких комунікацій

Розроблено інтерфейс для взаємодії користувачів з системою повідомлень. Застосовано принципи респонсивного дизайну для забезпечення комфортного використання інтерфейсу на різних пристроях. Додано можливість коментування на різних розділах веб-сайту, зокрема статей, зображень, відео та іншого контенту. Це стимулює взаємодію користувачів та надає їм можливість обмінюватись думками та враженнями.

Файл comments.component.ts відповідає за відправку коментаря. Усі коментарі зберігаються у базі даних Firebase. Для збереження коментарів  також враховуються аспекти безпеки та конфіденційності, забезпечуючи, що лише авторизовані користувачі можуть залишати коментарі, а доступ до бази даних обмежується лише необхідними дозволами. Функція getComments(), яка описана в компоненті comments, відповідає за завантаження коментарів з бази даних та передає їх у вигляді Observable за допомогою бібліотеки RxJS.

### 3.4 Розробка модулю аутентифікації та безпеки

Модуль аутентифікації та безпеки є важливою частиною системи, оскільки відповідає за захист конфіденційності користувачів та надання їм безпечного доступу до ресурсів веб-сайту. Створено функціонал для реєстрації нових користувачів. Враховано перевірку унікальності електронної пошти, використання надійних паролів та інші аспекти для забезпечення безпеки реєстраційного процесу. Розроблено модуль для безпечного входу користувачів в систему. Використані сучасні методи аутентифікації, а також вхід за допомогою Google акаунту.

Для безпечної авторизації користувачів через інші акаунти, такі як облікові записи у соціальних мережах, використана технологія OAuth. Технологія забезпечує безпечний обмін обліковими даними між інформаційною системою та соціальною платформою.

OAuth є протоколом авторизації, який дозволяє користувачам надавати обмежений доступ до своїх ресурсів без передачі свого облікового запису та пароля. Це стандартна технологія для безпечної авторизації через третіх сторін. При розробці системи, OAuth використовується для надання користувачам можливості авторизації через їхні облікові записи у соціальних мережах. Коли користувач обирає авторизуватися через Facebook, Google або іншу соціальну мережу, система взаємодіє з серверами цієї мережі, використовуючи OAuth, для отримання токену доступу. Інформаційна система отримує токен доступу, який може використовуватися для отримання обмеженого доступу до певних ресурсів користувача на соціальній мережі. Цей токен потім зберігається в безпечному місці на сервері системи. Отримавши такий токен, система може отримувати обмінюватись даними з соціальною мережею в ім'я користувача, при цьому не зберігаючи самі облікові дані користувача.

Серед особливостей протоколу можна виділити наступні:

Авторизація через соціальні мережі: При реалізації авторизації через соціальні мережі (наприклад, Facebook або Google), система взаємодіє з

сервісами цих мереж за допомогою OAuth. Користувач, обравши вхід через свій обліковий запис у соціальній мережі, переадресовується на сторінку цієї мережі для надання дозволу на доступ до облікових даних;

Отримання токену доступу: Коли користувач дає дозвіл, соціальна мережа генерує токен доступу. Цей токен є унікальним ідентифікатором, який дозволяє системі взаємодіяти з ресурсами соціальної мережі в ім'я користувача;

Безпечне зберігання та використання токену: Токен доступу зберігається в захищеному сховищі на стороні інформаційної системи. Він використовується для взаємодії з API соціальної мережі та отримання обміну контентом між системою та профілем користувача;

Безпека та конфіденційність: Використання OAuth дозволяє уникнути передачі облікових даних через систему, забезпечуючи безпеку і конфіденційність. Це робить процес авторизації більш безпечним для користувачів;

Оновлення та перевірка доступу: Токени доступу можуть мати обмежений строк дії. Система повинна регулярно оновлювати або перевіряти ці токени, щоб переконатися, що користувач продовжує мати доступ до відповідних ресурсів;

Таким чином, використання OAuth у системі забезпечує безпечний та зручний механізм авторизації через соціальні мережі, підвищуючи рівень безпеки та комфорту для користувачів. Схема роботи протоколу представлена на рисунку 3.3.

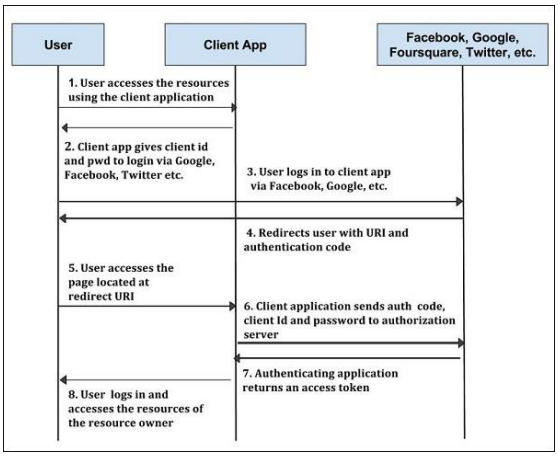


Рисунок 3.3 – Схема роботи протоколу OAuth

Для ефективної взаємодії системи з соціальними мережами було використано RESTful API. RESTful API є архітектурним стилем для створення веб-служб, який базується на принципах REST. REST (Representational State Transfer) визначає, як ресурси можуть бути представлені та маніпульовані за допомогою стандартних операцій HTTP. RESTful API використовується для взаємодії між веб-сайтами та додатками за допомогою HTTP-протоколу.

RESTful API використовується для взаємодії з API соціальних мереж, таких як Facebook або Twitter. Коли користувач вибирає ділитися чимось на своєму профілі чи отримує сповіщення через систему, використовуються RESTful запити для взаємодії з відповідними API цих соціальних мереж.

Система використовує RESTful API для отримання та оновлення даних користувачів на інформаційній платформі. Наприклад, для завантаження нових публікацій або оновлення інформації про користувача. RESTful API використовує стандартні HTTP-методи, такі як GET, POST, PUT та DELETE. Наприклад, для отримання даних використовується GET, для створення нових записів - POST, для оновлення - PUT, для видалення - DELETE. RESTful API повертає або отримує дані у форматі JSON або XML. Це дозволяє ефективно передавати структуровані дані між системою та соціальними мережами, забезпечуючи їхню взаємодію. Кожен ресурс (наприклад, користувач, публікація) має унікальний ідентифікатор URI. RESTful API використовує ці URI для ідентифікації та доступу до ресурсів, надаючи стандартизований спосіб взаємодії. Використання безстанової моделі дозволяє зменшити навантаження на сервер та підвищити продуктивність. Кешування може використовуватися для збереження результатів запитів, зменшуючи частоту повторних запросів до сервера [16].

Залежність angular-oauth2-oidc відповідає за SSO (Single Sign On) вхід за допомогою Google акаунту користувача. Сервіс створює спеціальний access token, завдяки якому Google надає нам інформацію про користувача, таку як його пошта, ім’я, фотографія. Для аутентифікації використовується Google API та

Firebase Authentication.

Angular Guards використовуються для контролю доступу до функціоналу модулю. За допомогою CanActivate та CanActivateChild Guards реалізовано перевірку авторизації користувача перед наданням доступу до редагування контенту. Модуль керуванням контентом взаємодіє із системою аутентифікації та безпеки. Доступ до редагування та додавання контенту можливий лише для користувачів, які пройшли аутентифікацію.

### 3.5 Висновки

У третьому розділі було обґрунтовано вибір мов програмування та технологій, які будуть використовуватися при розробці програмного продукту та виконано порівняння аналогів.

У результаті порівняння для клієнтської частини була обрана мова програмування JavaScript з фреймворком Angular та бібліотекою Bootstrap 5 для стилізації.

Було проведено розробку модулів керуванням контенту та рекомендацій, коментарів та аутентифікації та безпеки.

# 4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

## 4.1 Тестування програми

Перевірка програмного забезпечення – це стратегічний процес, спрямований на впевненість, що розроблений програмний продукт відповідає усім очікуваним вимогам та не має дефектів. Цей етап включає в себе виконання компонентів програми з використанням як ручних, так і автоматизованих інструментів для оцінки різних аспектів її функціональності.

Традиційно тестування програмного забезпечення проводилося відокремлено від інших етапів розробки, часто вже на пізніших стадіях життєвого циклу продукту. Основна мета полягає в виявленні помилок, багів та перевірці відповідності функціоналу вимогам. Правильно проведене тестування забезпечує надійність, безпеку та ефективність програмного продукту, що в свою чергу призводить до економії часу та високого задоволення клієнтів.

Сучасні підходи до тестування розглядаються в різних методологіях, але найпопулярнішими залишаються тестування чорного, білого та сірого ящика.

Тестування чорного ящика вражає своєю особливістю, оскільки тестувальники, що виконують його, не мають доступу до внутрішньої структури та вихідного коду програмного забезпечення. Їм не потрібні глибокі знання програмування для того, щоб виконати тести, оскільки їхня мета – не вдаватися в деталі коду, а взаємодіяти з інтерфейсом користувача, перевіряти функціональність та гарантувати відповідність роботи програми визначеним вимогам. Таке тестування часто називають функціональним або тестуванням на основі специфікації.

У складному процесі життєвого циклу тестування програмного забезпечення (STLC), велике значення надається тестуванню чорного ящика. Цей метод використовується з точки зору кінцевих користувачів і проводиться незалежною групою тестування. Тестер, надаючи різноманітні вхідні дані, систематично перевіряє вихідні дані на відповідність очікуваним результатам. Кожен несподіваний висновок чи відхилення докладно документуються і

повідомляються розробникам, що сприяє виявленню та виправленню функціональних помилок на ранніх етапах розробки.

Цей ефективний підхід застосовується на всіх рівнях тестування ПЗ: від одиничного та інтеграційного до системного та прийняття. У модульному тестуванні метод чорного ящика використовується для перевірки відповідності інтерфейсу специфікаціям, наданим клієнтом.

Під час інтеграційного тестування, основною метою є виявлення та усунення помилок у взаємодії інтегрованих компонентів інтерфейсу. Метод чорного ящика ефективно використовується і під час системного тестування для аналізу відповідності системи вимогам, а також під час прийняття, де він допомагає підтвердити прийняття програмного продукту в різних умовах.

Серед популярних методів проектування тестів чорної скриньки можна виокремити:

- Тестування таблицею рішень: застосовується для налагодження ПЗ на основі умовних операторів, що дозволяє ефективно виявляти помилки в логіці дій.

- Вгадування помилок: ґрунтується на інтуїції та досвіді тестувальника, що дозволяє визначити можливі причини збоїв чи помилок.

- Тестування параметрів: перевіряє всі можливі комбінації кожної пари вхідних параметрів, виявляючи поширені помилки у їх взаємодії.

- Техніка еквівалентного розподілу: дозволяє систематично тестувати різні класи вхідних даних, скорочуючи час, необхідний для тестування.

Тестування чорного ящика виявляє будь-які непрозорість, неузгодженості та невідповідності у функціональних специфікаціях, що дозволяє підвищити якість реалізації функціональних можливостей без втручання у великі сегменти програмного коду.

Тестування "чорного ящика" визначається абсолютною об'єктивністю, оскільки його проводить незалежна команда, відокремлюючи точку зору кінцевих користувачів від поглядів розробників. З серед трьох методів "чорного ящика" виокремлюється той, який найшвидше дозволяє створювати тестові

випадки, оскільки він не потребує спеціальних знань з програмування і може бути легко виконаний тестувальниками без технічної підготовки [36].

Проте важливо відзначити, що цей метод ефективний переважно для тестування обмежених частин програмного забезпечення. Розгортання детального тестування великої та складної програми методом "чорного ящика" може виявитися неефективним та вимагати значних зусиль та часу. Додатково, успішне застосування цього методу передбачає наявність чітких і вичерпних специфікацій, без чого розробка тестових випадків може стати важкою задачею, а їх покриття буде обмеженим.

У порівнянні з "чорним ящиком", тестування "білого ящика" фокусується на внутрішній структурі програмного забезпечення та логіці його функціонування. Також відоме як структурне тестування або тестування на основі логіки, цей метод вимагає від тестувальників глибоких знань програмування, доступу до вихідного коду та архітектурної документації.

Тестувальники, що застосовують "білий ящик", аналізують код та внутрішні аспекти програми, ідентифікують можливі вхідні дані, перевіряють умови та твердження, і вивчають шляхи коду та потоки даних. Їхня мета - виявлення прихованих дефектів та упевненість, що програма працює належним чином. Однак цей метод вимагає від тестувальників спеціалізованих інструментів для аналізу вихідного коду та налагодження.

Тестування білого ящика є важливою частиною процесу забезпечення якості програмного забезпечення. Хоча його застосовують у модульному тестуванні, сучасні тренди визначають його основне використання у великих і складних проектах під час інтеграційного та регресійного тестування. Цей метод дозволяє тестувальникам не лише перевіряти правильність логіки коду на рівні блоків, а й виявляти дефекти, які можуть впливати на роботу програмного забезпечення в цілому [37].

Під час інтеграційного тестування техніка білого ящика виявляється дуже корисною для аналізу взаємодій між різними інтерфейсами та підсистемами. Також під час регресійного тестування використання методу білого ящика

дозволяє ефективно перевіряти тестові випадки на рівні як одиничних блоків, так і їх інтеграції.

Найпоширеніші методи проектування тестів білого ящика включають тестування потоку керування, яке акцентує на перевірці логіки коду через вхідні значення та їх відповідність очікуваним результатам. Також ефективним є тестування потоку даних, спрямоване на виявлення помилкового використання значень даних та аномалій у потоці даних.

Важливо відзначити, що тестування білого ящика, хоча і дозволяє глибше вникнення в код, вимагає від тестувальників високого рівня експертизи у програмуванні. Це може бути реалізовано через найм висококваліфікованих фахівців, що забезпечить ефективне використання цього методу.

Хоча тестування білого ящика може бути більш часо та ресурсозатратним порівняно з тестуванням чорного ящика, його глибше розуміння коду дозволяє виявляти непомітні помилки та покращувати загальну якість програмного забезпечення. Такий підхід до тестування є важливою частиною циклу розробки, сприяючи вчасному виявленню та усуненню проблем у програмному продукті.

Тестування чорної скриньки та білої скриньки виявляють різницю в своїх підходах, при цьому кожен метод має свої переваги та обмеження. Тим не менше, тестування сірої скриньки об'єднує кращі аспекти обох підходів, забезпечуючи балансований підхід і вирішуючи багато проблем.

Метод сірої скриньки розширює обсяг тестування, фокусуючись на всіх рівнях програмного забезпечення, незалежно від його складності. Тестувальники, які використовують цей підхід, одночасно перевіряють інтерфейси та функціональність, а також вдивляються у внутрішню структуру та виправляють код програмного забезпечення.

Метод сірої скриньки особливо ефективний на етапі інтеграційного тестування, особливо для веб-додатків, де відсутній вихідний код. Також він може бути застосований для тестування бізнес-домену та підтвердження відповідності програмного забезпечення вимогам.

Метод сірої скриньки використовує різні методи проектування тестів, такі

як матричне тестування для повного покриття, регресійне тестування для аналізу впливу змін та техніку тестування шаблонів для аналізу раніше виявлених дефектів.

Невід'ємною перевагою є те, що тестування сірої скриньки не вимагає повного доступу до вихідного коду, забезпечуючи нейтральність процесу тестування. Однак цей метод може виявитися надто обтяжливим, якщо розробник вже запустив відповідні тести, а обмежена інформація про внутрішню структуру може призвести до часткового покриття тестами.

Щоб максимально використовувати переваги тестування сірої скриньки, важливо визначити чіткі та повнефункціональні вимоги до системи. Також, комбінування цього методу із тестуванням білої скриньки, де використовується знання внутрішньої структури програми, може забезпечити більш повне тестування продукту та зменшити ймовірність упущених дефектів.

Також важливо відзначити, що ідентифікація дефектів у розподілених системах за допомогою методу сірої скрині може виявитися викликом. Таким чином, хоча цей підхід є збалансованим та універсальним, йому властиві свої обмеження та вимоги до ефективного управління проектом.

Було проаналізовано та порівняно три ключові підходи до тестування програмного забезпечення, які компанії використовують для забезпечення якості своїх продуктів та суворої відповідності специфікаціям вимог. Використання цих методів тестування допомагає вирішити численні проблеми, які, в довгостроковій перспективі, можуть перетворитися на значні технічні зобов'язання.

Таким чином, в якості обраної методики тестування програмного додатку було визначено тестування методом сірого ящика. Це рішення базується на частковому описі алгоритмів роботи додатку. Вибір цього підходу передбачає високий рівень надійності та об'єктивності у визначенні ефективності та відповідності програмного забезпечення вимогам.

## 4.2 Тестування модулю інтеграції з соціальними мережами

Користувачі можуть залишати коментарі під постами. Для цього їм потрібно авторизуватись у системі. Це можна зробити за допомогою реєстрації з почтою та паролем або увійти за допомогою аккаунту з соціальних мереж. Протестуємо функцію логіна за допомогою Google акаунта користувача. Для цього тиснемо на кнопку “Sign in with Google” та потрапляєм у вікно, яке зображено на рисунку 4.1.

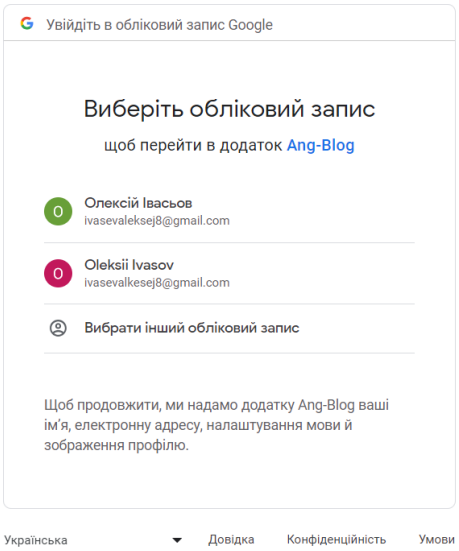


Рисунок 4.1 – Вікно авторизації у Google акаунт

Авторизувавшись, користувач потрапляє на головне вікно і у правому верхньому куті може побачити свою пошту.

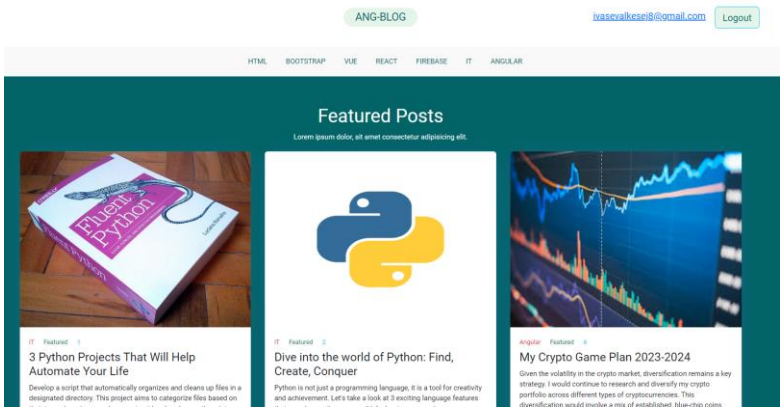


Рисунок 4.2 – Головне вікно додатку

Увійшовши в систему, користувач може писати коментарі і відповідати на коментарі інших. На рисунку 4.3 представлено секцію з коментарями.

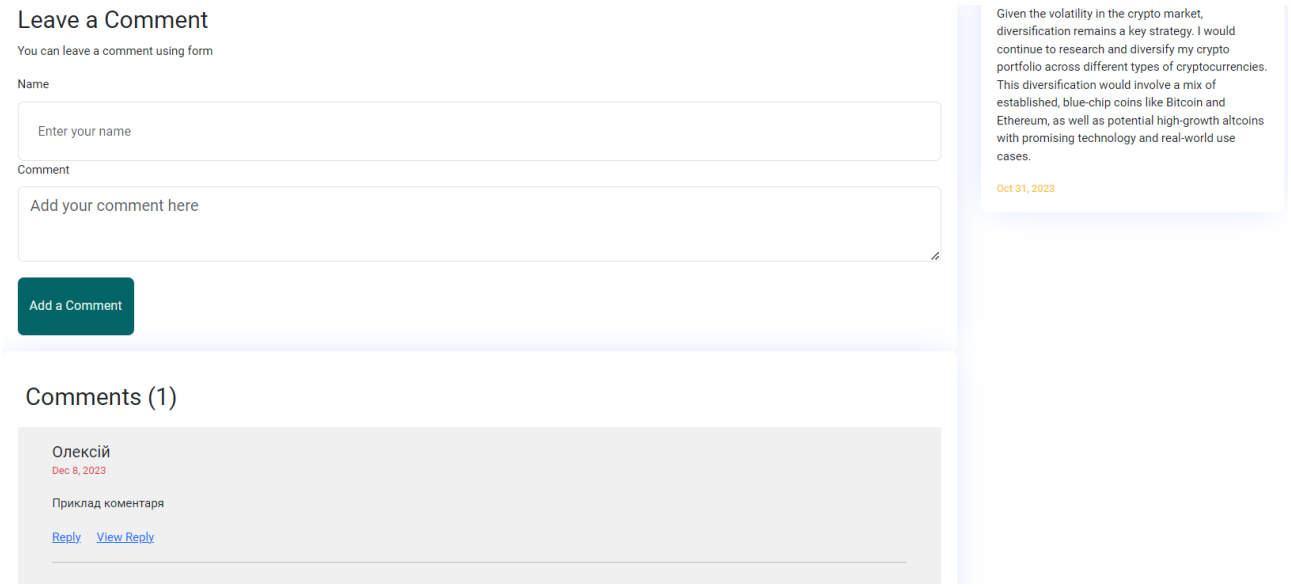


Рисунок 4.3 – Секція з коментарями

Також, інтеграція з соціальними мережами передбачає можливість ділитися постом в найбільш поширених соціальних мережах. Наприклад натиснувши на іконку Твітера, користувач попаде на сторінку соціальної мережі, де йому необхідно буде авторизуватись щоб поділитись цим постом.

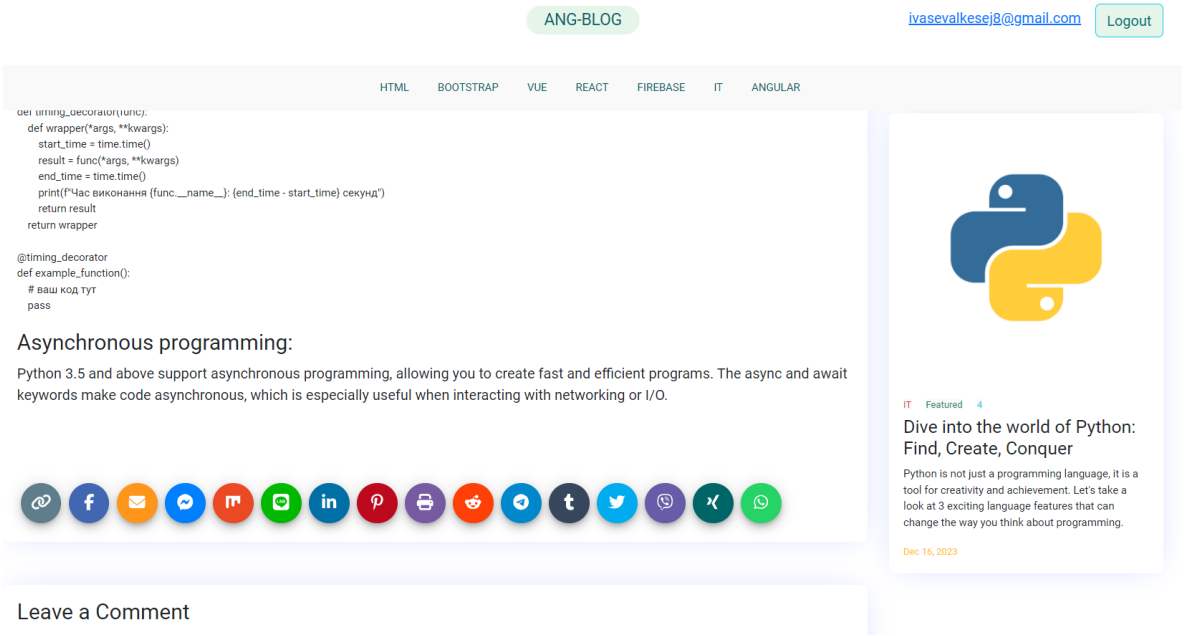


Рисунок 4.4 – Іконки для поширення

## 4.2 Тестування модулю керуванням контенту

Увійшовши до панелі адміністратора, користувач може керувати категоріями постів, самими постами, та підписниками.

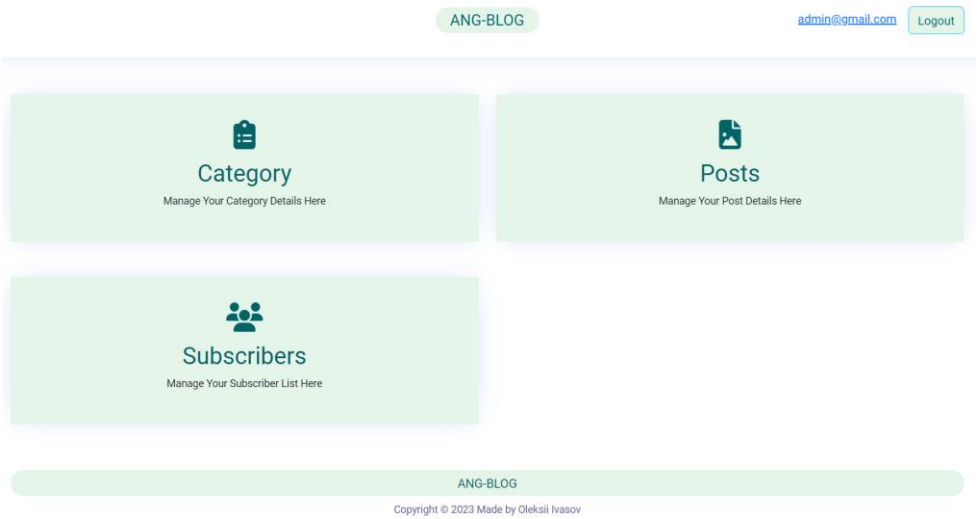


Рисунок 4.5 – Головне вікно панелі адміністратора

Усі пости сортуються за категоріями, адміністратор може додати категорії на сторінці “Add Categories”. Також доступне видалення та зміна вже існуючих категорій.

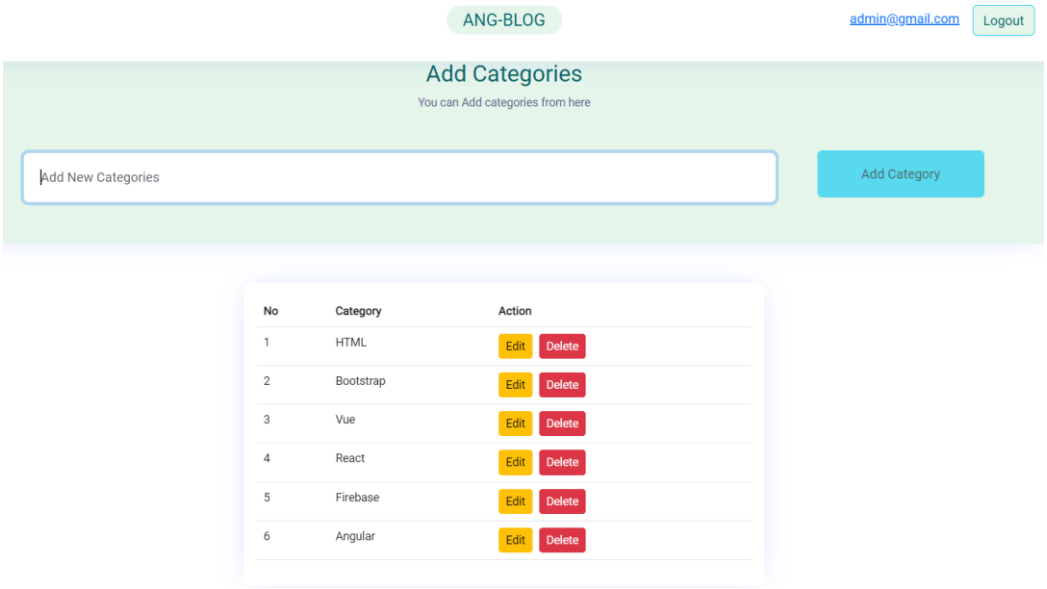


Рисунок 4.6 – Вікно додавання категорій

На сторінці “Posts” адміністратор може переглянути усі створені пости, додати нові, відредагувати вже існуючі, та помітити пост як особливий. Особливі пости користувачі будуть бачити першими.

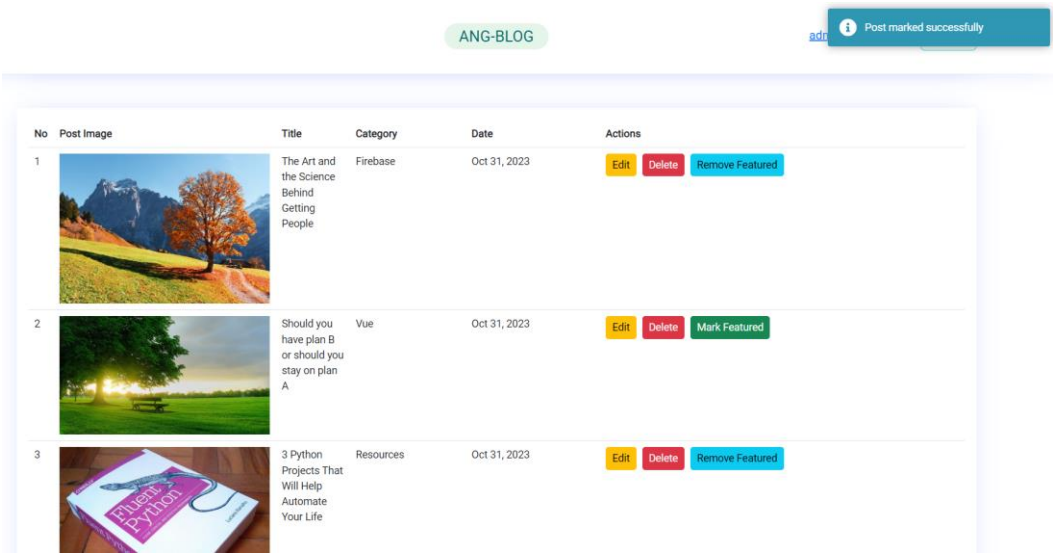


Рисунок 4.7 – Вікно перегляду постів

Адміністратор може додати пост заповнивши усі необхідні поля, а саме: заголовок, опис, категорія, картинка, та контент. Для редагування контенту доступний широкий набір інструментів для роботи з текстом.

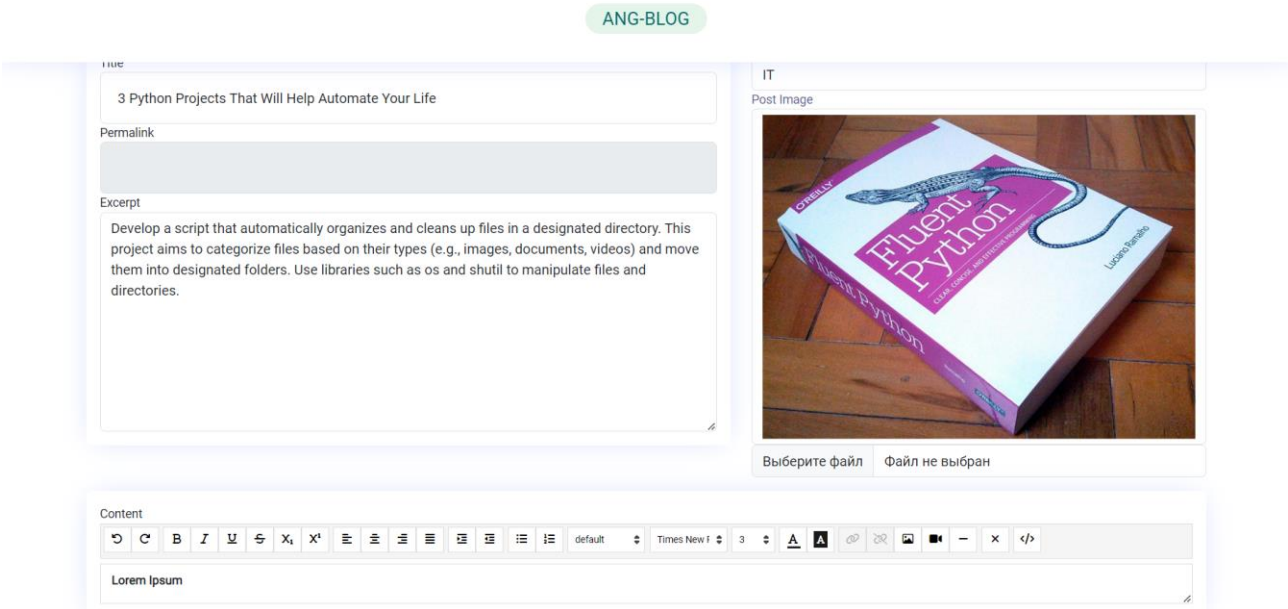


Рисунок 4.8 – Вікно додавання посту

У бічній панелі, відображаються пости, які можуть зацікавити користувача і які підбираються з залежності від того, які пости дивився користувач. Наприкад на рисунку 4.9 користувач переглядає пост про штучник інтелект і система рекомендує йому пости про мову програмування Python.

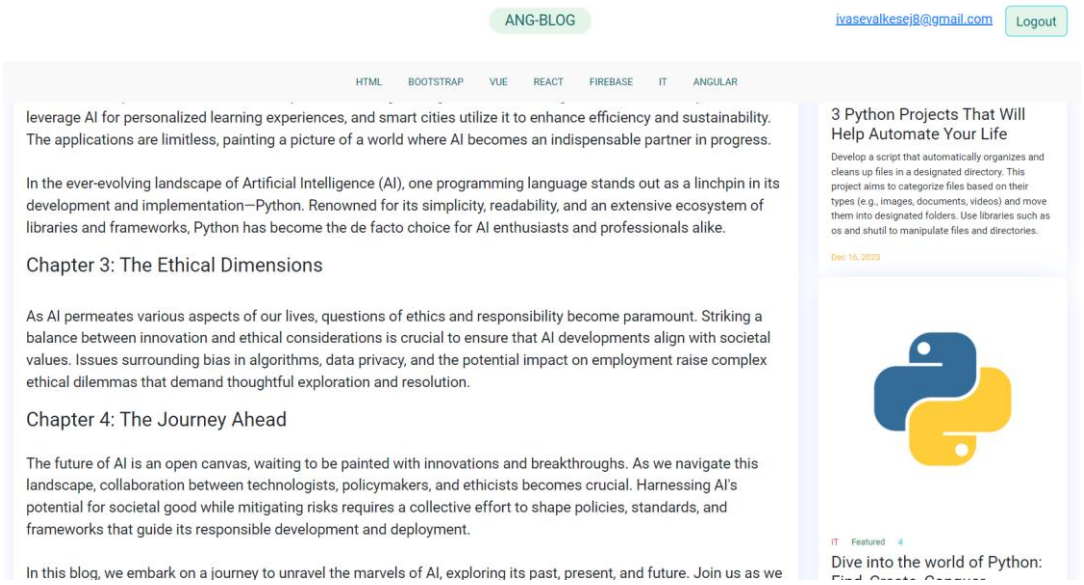


Рисунок 4.9 – Вікно з вмістом посту

Користувач також може підписатись на сповіщення про нові пости. Для цього йому необхідно вказати ім’я та свою електрону пошту.

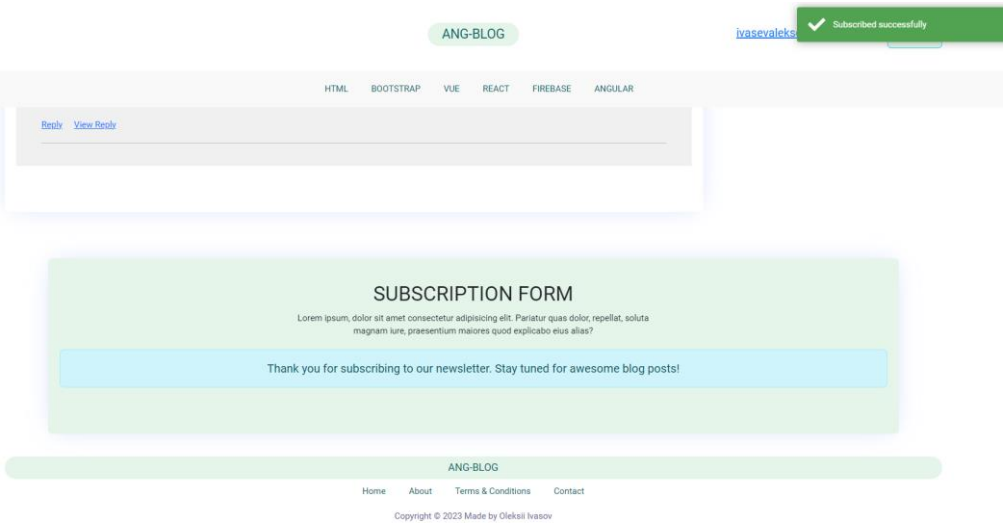


Рисунок 4.10 – Форма підписки

Таблиця 4.1 Тестові випадки

| Ідентифікатор | Назва | Методика проведення тестування | Очікуваний результат | Результат |
|---|---|---|---|---|
| ТВ-1 | Перевірка відображення елементів вікна логіну | 1. Завантажити додаток, відкривши сайт | 1. Відображається логотип та назва додатку.<br>2. Відображається меню.<br>3. Відображається вікно логіну. | Виконано |
| ТВ-2 | Перевірка функції логіну у застосунок | 1. Завантажити додаток, відкрити сайт.<br>2. Ввести електрону пошту та пароль.<br>3. Натиснути кнопку «Join». | 1. Відкривається головне вікно застосунку. | Виконано |
| ТВ-3 | Перевірка функції додавання категорій | 1. Увійти у панель адміністратора<br>2. Перейти у розділ “Categories”<br>3. Вписати назву категорії і натиснути кнопку «Add». | 1 Зелене сповіщення в правому верхньому куті “Category successfully added”.<br>2. У вікні додавання нового посту у списку категорій з’явилась нова категорія. | Виконано |
| ТВ-4 | Перевірка фунції додавання посту. | 1. Увійти у панель адміністратора<br>2. Перейти у розділ “Posts”<br>3. Заповнити усі необхідні поля. | 1. Зелене сповіщення в правому верхньому куті “Post successfully added”.<br>2. У головному вікні з’явився новий пост. | Виконано |

Продовження таблиці 4.1

| Іденти фікатор | Назва | Методика проведення тестування | Очікуваний результат | Результат |
|---|---|---|---|---|
| ТВ-5 | Перевірка функції авторизації у застосунку через Google | 1. Завантажити додаток, відкривши сайт.<br>2. Натиснути кнопку “Sign up with Google”<br>3. Увійти у свій Google акаунт. | 1. Користувач потрапляє на головну сторінку.<br>2. У верхньому правому куті видно електрону пошту користувача та кнопку “Logout” | Виконано |
| ТВ-6 | Перевірка функції залишення коментарів | 1. Увійти за допомогою пошти та паролю або повторити кроки вказані в ТВ-5.<br>2. Натиснути на один з постів<br>3. Під постом заповнити форму.<br>4. Натиснути кнопку “Add a comment” | 1. Зелене сповіщення в правому верхньому куті “Comment successfully added”.<br>2. Новий коментар одразу ж з’являється під постом | Виконано |

Отже, за допомогою вищенаведених тестових випадків було проведено тестування графічного інтерфейсу та функціоналу програмного продукту, та підтверджено його працездатність і коректність виконання дій.

### 4.3 Висновки

У четвертому розділі було проведено тестування додатку, у результаті якого було доведено його повну працездатність та відповідність поставленому технічному завданню.

Розглянуто методи тестування ПЗ та, проаналізувавши їх, було обрано метод «сірої скриньки» для тестування інтерфейсу і функціоналу додатку.

Створено таблицю тестових випадків для перевірки застосунку.

# 5 ЕКОНОМІЧНА ЧАСТИНА

## 5.1 Проведення комерційного аудиту науково-технічної розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Результатом магістерської кваліфікаційної роботи «Інформаційна система керування контентом та користувацькими комунікаціями» є програмний продукт у вигляді веб додатку для керуванням контенту.

Для проведення технологічного аудиту залучено трьох незалежних експертів: Чорноволик Галина Олександрівна (к.т.н. доц. кафедри ПЗ ВНТУ), Бабюк Наталя Петрівна (к.т.н. доц. кафедри ПЗ ВНТУ), Майданюк Володимир Павлович (к.т.н. доц. кафедри ПЗ ВНТУ).

Оцінювання комерційного потенціалу буде здійснене за критеріями, що наведені в таблиці 5.1

Таблиця 5.1 - Критерії оцінювання комерційного потенціалу розробки бальна оцінка

| Критерії оцінювання та бали (за 5-ти бальною шкалою) | | | | | |
|---|---|---|---|---|---|
| Кри-терій | 0 | 1 | 2 | 3 | 4 |
| Технічна здійсненність концепції: | | | | | |
| 1 | Достовірність концепції не підтверджена | Концепція підтверджена експертними висновками | Концепція підтверджена розрахунками | Концепція перевірена на практиці | Перевірено роботоздатність продукту в реальних умовах |
| Ринкові переваги (недоліки): | | | | | |
| 2 | Багато аналогів на малому ринку | Мало аналогів на малому ринку | Кілька аналогів на великому ринку | Один аналог на великому ринку | Продукт не має аналогів на великому ринку |
| 3 | Ціна продукту значно вища за ціни аналогів | Ціна продукту дещо вища за ціни аналогів | Ціна продукту приблизно дорівнює цінам аналогів | Ціна продукту дещо нижче за ціни аналогів | Ціна продукту значно нижче за ціни аналогів |

Продовження таблиці 5.1

| Критерії оцінювання та бали (за 5-ти бальною шкалою) | | | | | |
|---|---|---|---|---|---|
| Кри-тер. | 0 | 1 | 2 | 3 | 4 |
| 4 | Технічні та споживчі властивості продукту значно гірші, ніж в аналогів | Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів | Технічні та споживчі властивості продукту на рівні аналогів | Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів | Технічні та споживчі властивості продукту значно кращі, ніж в аналогів |
| 5 | Експлуатаційні витрати значно вищі, ніж в аналогів | Експлуатаційні витрати дещо вищі, ніж в аналогів | Експлуатаційні витрати на рівні експлуатаційних витрат аналогів | Експлуатаційні витрати трохи нижчі, ніж в аналогів | Експлуатаційні витрати значно нижчі, ніж в аналогів |
| Ринкові перспективи | | | | | |
| 6 | Ринок малий і не має позитивної динаміки | Ринок малий, але має позитивну динаміку | Середній ринок з позитивною динамікою | Великий стабільний ринок | Великий ринок з позитивною динамікою |
| 7 | Активна конкуренція великих компаній на ринку | Активна конкуренція | Помірна конкуренція | Незначна конкуренція | Конкурентів немає |
| Практична здійсненність | | | | | |
| 8 | Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї | Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців | Необхідне незначне навчання фахівців та збільшення їх штату | Необхідне незначне навчання фахівців | Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї |
| 9 | Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні | Потрібні незначні фінансові ресурси. Джерела фінансування відсутні | Потрібні значні фінансові ресурси. Джерела фінансування є | Потрібні незначні фінансові ресурси. Джерела фінансування є | Не потребує додаткового фінансування |
| 10 | Необхідна розробка нових матеріалів | Потрібні матеріали, що використовуються у військово-промисловому комплексі | Потрібні дорогі матеріали | Потрібні досяжні та дешеві матеріали | Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві |

Продовження таблиці 5.1

| Кри-тер. | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 11 | Термін реалізації ідеї більший за 10 років | Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років | Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років | Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років | Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років |
| 12 | Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту | Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу | Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу | Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту | Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту |

Результати оцінювання комерційного потенціалу експертами розробки зведено в таблицю 5.2.

Таблиця 5.2 - Результати оцінювання комерційного потенціалу розробки

| Критерії | Прізвище, ініціали, посада експерта | | |
|---|---|---|---|
| | 1 – Чорноволик | 2 – Бабюк | 3 – Майданюк |
| | Бали, виставлені експертами: | | |
| 1 | 4 | 3 | 4 |
| Ринкові переваги (недоліки): | | | |
| 2 | 3 | 2 | 3 |
| 3 | 4 | 4 | 4 |
| 4 | 4 | 3 | 4 |
| 5 | 3 | 4 | 3 |
| Ринкові перспективи | | | |
| 6 | 2 | 3 | 3 |
| 7 | 4 | 4 | 4 |
| Практична здійсненність | | | |
| 8 | 4 | 4 | 4 |
| 9 | 4 | 4 | 4 |
| 10 | 4 | 4 | 4 |
| 11 | 4 | 4 | 4 |
| 12 | 3 | 4 | 4 |
| Сума балів | $СБ_1$=45 | $СБ_2$=45 | $СБ_3$=45 |
| Середньоарифметична сума балів $\overline{СБ}$ | 45 | | |

За даними таблиці 5.2 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 5.3.

Таблиця 5.3 – Рівні комерційного потенціалу розробки

| Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів | Рівень комерційного потенціалу розробки |
|---|---|
| 0 – 10 | Низький |
| 11 – 20 | Нижче середнього |
| 21 – 30 | Середній |
| 31 – 40 | Вище середнього |
| 41 – 48 | Високий |

Рівень комерційного потенціалу розробки, становить 45 бали, що відповідає рівню «високий».

В якості аналога для розробки було обрано систему керування контентом WordPress. Основними недоліками аналога є уразливість до вторгнень, шкідливого програмного забезпечення та інших загроз. З ростом функціональності та розширень, деякі сайти на WordPress можуть стати повільними. Неправильно налаштовані або надмірно великі розширення можуть вплинути на продуктивність. Також до недоліків можна віднести велику кількість функцій, що роблять користування системою дещо складнішим.

У розробці дана проблема вирішується використанням безпечних бібліотек та фреймворків для розробки, які вже мають вбудовані заходи безпеки. Також програмний продукт випереджає аналог за такими параметрами як мінімізація інтенсивності використання ресурсів шляхом оптимізації коду та ресурсозберігаючих технік та добре спроектований інтерфейс з фокусом на інтуїтивність та легкість використання. У таблиці 5.4 наведені основні технічні показники аналога і нового програмного продукту

Таблиця 5.4 - Основні технічні показники аналога і нового програмного продукту

| Показники | Аналог | Нова розробка | Відношення параметрів нової розробки до параметрів аналога |
|---|---|---|---|
| Функціональність | 90% | 90% | 1 |
| Надійність | 70% | 80% | 1,14 |
| Сумісність | 50% | 70% | 1,4 |
| Супровід | 70% | 70% | 1 |
| Економія ресурсів і часу | 50% | 80% | 1,6 |
| Простота використання | 60% | 80% | 1,3 |

Нова розробка може використовуватися як у веб-розробці, так і в інших сферах інтернет-технологій. Безпосереднім споживачем є власники та адміністратори веб-сайтів, розробники веб-додатків, агентства з цифрового маркетингу та компанії, які прагнуть оптимізувати та автоматизувати управління своїм вмістом в Інтернеті через використання цієї системи.

Технічна готовність складає 95%. Була розроблена архітектура програмного продукту, розроблено прототип, який частково покриває необхідний функціонал. На даний момент існує декілька зацікавлених сторін з комерціалізації розробки.

## 5.2 Розрахунок витрат на здійснення науково-технічної розробки.

Проведемо розрахунок витрат на здійснення науково-технічної розробки. Обчислимо основну заробітну $З_о$ розробника, за формулою (5.1):

$$З_о = \frac{M}{T_р} \cdot t[грн], \quad (5.1)$$

де М – місячний посадовий оклад конкретного розробника, грн;

$T_р$ – число робочих днів в місяці –24;

t – число днів роботи розробника.

Обчислимо заробітну плату розробника з місячним окладом 18000 грн і кількістю робочих днів у місяці – 24. Число днів роботи розробника 72.

$$З_о = \frac{20000}{24} \cdot 72 = 60000{,}00(\text{грн}).$$

Результати розрахунків зведемо до таблиці 5.5.

Таблиця 5.5 – Основна заробітна плата розробників

| Найменування посади | Місячний посадовий оклад, грн. | Оплата за робочий день, грн. | Число днів роботи | Витрати на заробітну плату, грн. |
|---|---|---|---|---|
| Розробник | 18000,00 | 750,00 | 72 | 54000,00 |
| Керівник роботи | 20000,00 | 833,33 | 72 | 60000,00 |
| Всього | | | | 114 000,00 |

Обчислимо додаткову заробітну плату розробників $З_д$. Розрахуємо її як 10% від основної заробітної плати за формулою (5.2):

$$З_д = З_о \cdot 0{,}11\ [грн]. \quad (5.2)$$

$$З_д = 114\ 000 \cdot 0{,}11 = 12\ 540(\text{грн}).$$

Згідно діючого законодавства нарахування на заробітну плату (Єдиний соціальний внесок) складають 22% від суми основної та додаткової заробітної плати, як подано формулою (5.3):

$$Н_з = (З_{о\ р} + З_д) \cdot 0{,}22\ [\text{грн}]. \quad (5.3)$$

$$Н_з = (114\ 000+12\ 540) \cdot 0{,}22 = 27838{,}8\ (\text{грн}).$$

Обчислимо амортизацію обладнання, що використовувались для розробки. В спрощеному вигляді амортизаційні відрахування розраховується за формулою(5.4):

$$A = \frac{Ц \cdot T}{12 \cdot T_B} \ [грн], \tag{5.4}$$

де $Ц$ – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, грн;

Т – фактична тривалість використання, міс;

$Т_В$ – термін використання обладнання, приміщень тощо, роки.

Розрахуємо амортизаційні витрати на системний блок та монітор. Балансова вартість блоку становить 20000 грн, а термін його використання – 5 років, а фактична тривалість використання 3 місяці.

$$A = \frac{20000 \cdot 3}{12 \cdot 5} = 1000(грн).$$

Величина амортизаційних відрахувань наведена в таблиці 5.6.

Таблиця 5.6 – Величина амортизаційних відрахувань

| № | Найменування | Балансова вартість, грн | Термін використання, р | Фактична тривалість використання, міс. | Величина амортизаційних відрахувань, грн |
|---|---|---|---|---|---|
| 1 | Комп'ютер | 20000 | 5 | 3 | 1750 |
| Всього: | | | | | 1750 |

Обчислимо витрати на силову електроенергію за формулою (5.5):

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{впі}}{\eta_i} \tag{5.5}$$

де $W_{yi}$ – встановлена потужність обладнання на визначеному етапі розробки, кВт;

$t_i$ – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо Ц$_е$ = 7,50 грн;

$К_{вні}$ – коефіцієнт, що враховує використання потужності, $К_{вні} < 1$;

$\eta_i$ – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$В_е = 0,5 \cdot 504,0 \cdot 7,50 \cdot 0,95 / 0,97 = 1851,03$ *грн.*

Під час розробки програмного продукту використовувались лише безкоштовні програмні засоби.

Інші витрати ($В_{ін}$) охоплюють: витрати на Інтернет, управління організацією, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, тощо. Інші витрати приймаємо як 100% від суми основної заробітної плати розробників.

Величину інших витрат розрахуємо за формулою (5.6):

$$В_{ін} = З_о \cdot 50\% \text{ [грн]}. \quad (5.6)$$

$$В_{ін} = 60000 \cdot 0,5 = 30000 \text{ (грн)}.$$

Обчислимо витрати на виконання даної роботи, що є сумою всіх попередніх витрат.

$$В = З_о + З_р + З_д + Н_з + А + В_е + В_{ін} \text{ [грн]}.$$

В=60 000+12 540+27838,8 +1750,00+199,75+30 000 = 132 328, 55(грн).

Виконаємо прогнозування загальних витрат *(ЗВ)* на виконання та впровадження результатів виконаної науково-технічної роботи за формулою(5.7):

$$ЗВ = В_{заг} /\beta \ [грн], \quad (5.7)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Якщо розробка знаходиться на стадії впровадження, то β ≈ 0,9.

ЗВ = 132 328, 55/0,9=147 031, 722 (грн).

Витрати на виконання наукової роботи та впровадження її результатів становитиме 147 031, 722 грн.

## 5.3 Розрахунок економічної ефективності та обґрунтування економічної доцільності комерціалізації науково-технічної розробки.

Узагальнюючим результатом, який може отримати потенційний інвестор від комерціалізації науково-технічної розробки, є щорічне збільшення величини чистого прибутку. Зростання чистого прибутку надає можливість інвестору отримувати додаткові кошти, покращувати фінансові результати діяльності.

Виконання даної наукової роботи та впровадження її результатів складає приблизно до 1 року. Позитивні результати від впровадження розробки очікуються вже в перші місяці після впровадження.

Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

Збільшення чистого прибутку у потенційного інвестора протягом декількох років розраховується за формулою (5.8):

$$\Delta П_i = (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}) \quad (5.8)$$

де $\Delta Ц_o$ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році;

N – основний кількісний показник, який визначає величину попиту на аналогічні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої науково-технічної розробки у році до впровадження результатів;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році;

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. ρ = 0,3;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у ϑ = 18%.

В результаті впровадження наукової розробки покращується якість певного продукту, що дозволяє підвищити ціну його реалізації на 600 грн. Кількість користувачів збільшиться протягом першого року на 500, другого – 1500, третього – 1000. Реалізація продукції до впровадження результатів наукової розробки складала 1 користувача, а її ціна становила – 8500 грн.

Розрахуємо показник прибутку впродовж трьох років відносно базового.

Збільшення чистого прибутку $\Delta П_1$ протягом першого року складе:

$\Delta П_1 = (600 \cdot 30\,000 + (8500 + 600) \cdot 500) \cdot 0{,}8333 \cdot 0{,}3 \cdot (1 - 18/100) = 4\,622\,565{,}90$ (грн).

Обчислимо збільшення чистого прибутку $\Delta П_2$ протягом другого року:

$\Delta П_2$ = (600 · 30 000 + (8500 + 600) · (500+1500)) · 0,8333 · 0,3 · (1 – 18/100)= 7 420 703,16 (грн).

Збільшення чистого прибутку $\Delta П_3$ протягом третього року становитиме:

$\Delta П_3$ = (600 · 30 000 + (8500 +600) · (500+1500+1000)) · 0,8333 · 0,3 · (1 –18/100) = 9 286 128,54 (грн).

Отже, розрахунки показують, що комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку.

Далі розраховуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{\text{інв}} \cdot \text{ЗВ}$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. $k_{\text{інв}}$ = 2.

$$\text{PV} = 2 \cdot 147\,031,722 = 294\,063,444 \text{ (грн)}.$$

Чистий приведений дохід для інвестора розраховуємо за формулою (5.9):

$$\text{ПП} = \sum_1^{\text{T}} \frac{\Delta \text{П}_\text{i}}{(1+\tau)^{\text{t}}} \text{ [грн]}, \qquad (5.9)$$

де $\Delta П_i$ - збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої рроботи, грн;

т – період часу, протягом якого виявляються результати впровадженої роботи, роки;

$\tau$ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки „0”.

$$\text{ПП} = \frac{4\ 622\ 565{,}90}{(1+0{,}1)^1} + \frac{7\ 420\ 703{,}16}{(1+0{,}1)^2} + \frac{9\ 286\ 128{,}54}{(1+0{,}1)^3} = 17\ 312\ 000.\ (\text{грн}).$$

Розрахуємо абсолютну ефективність вкладених інвестицій $\text{Е}_{\text{абс}}$ за формулою(5.10):

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV),\ [\text{грн}], \qquad (5.10)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн.;

PV – теперішня вартість інвестицій PV = ЗВ, грн.

$$E_{абс} = 17\ 312\ 000. - 294\ 063,\ 444 = 17\ 017\ 936{,}6(\text{грн}).$$

Так як $E_{абс} > 0$, то результат від проведення наукових досліджень та їх впровадження принесе прибуток, але це також ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даного проекту.

На п’ятому етапі розрахуємо відносну (щорічну) ефективність вкладених у наукову розробку інвестицій $E_в$. Для цього використаємо формулу (5.11):

$$\text{Е}_{\text{в}} = \sqrt[\text{Тж}]{1 + \frac{\text{Е}_{\text{абс}}}{PV}} - 1 \qquad (5.11)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

$PV$ теперішня вартість інвестицій PV = ЗВ, грн;

$T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{17\,017\,936{,}6}{294\,063{,}444}} - 1 = 2{,}89 \text{ або } 289\%.$$

Розраховану величину $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладати не вигідно. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування визначається за формулою (5.12):

$$\tau = d + f, \tag{5.12}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; d = (0,09);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau = 0{,}09 + 0{,}1 = 0{,}19$$

Оскільки $E_в = 289\% > \tau$ мін = 0,19 = 19%, то потенційний інвестор може бути зацікавлений у фінансуванні науково-технічної розробки.

Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ можна розрахувати за формулою (5.13):

$$T_{ок} = \frac{1}{E_в}, \tag{5.13}$$

$$T_{ок} = \frac{1}{2{,}89} = 0{,}34 \ (роки),$$

Термін окупності становить 0,34 року ( $T_{ок}$ <3…5 років), що свідчить, що фінансування даної наукової розробки є доцільним.

## 5.4 Висновки

У п’ятому розділі було виконано оцінювання комерційного потенціалу розробки. Проведено комерційний аудит з залученням трьох незалежних експертів. Визначено, що рівень комерційного потенціалу розробки є високим.

Аналіз комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти, що підтверджує її перспективність. Програмний продукт має кращі функціональні показники, а тому є конкурентоспроможним товаром, а існуючі переваги нової розробки дозволять швидко поширити її на ринку.

Згідно із розрахунками всіх статей витрат на виконання науково-технічної роботи загальні витрати на розробку складають 147 031, 722 грн.

Розрахована абсолютна ефективність вкладених інвестицій в сумі 294 063, 444 грн свідчить про отримання прибутку інвестором від комерціалізації програмного продукту.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 289%, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 19%. Це означає потенційну зацікавленість інвесторів у комерціалізації нового програмного продукту.

Отже, розрахунок ефективності вкладених інвестицій та періоду їх окупності показав, що фінансування розробки є доцільним, адже інвестиції буде повернуто в термін до 1 року.

# ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи було розроблено програмний застосунок інформаційної системи керуванням контенту та користувацькими комунікаціями.

Було проаналізовано стан даної проблеми на сьогоднішній день.

Розглянуто основні аналоги, визначено їх особливості та недоліки і розроблено порівняння з власним програмним продуктом.

У результаті аналізу для створення графічного інтерфейсу застосунку було обрано мову програмування JavaScript з фреймворком Angular та бібліотекою Bootstrap. Для розробки було використано середовище програмування Visual Studio Code .

У результаті роботи було зроблено наступне:

- проаналізовано стан задачі;

- порівняно аналоги програмного застосунку, зроблено порівняльну таблицю;

- визначено найбільш ефективний підхід для створення інформаційної системи;

- розроблено графічний інтерфейс користувача для взаємодії із інформаційною системою;

- розроблено алгоритми роботи програмного застосунку та його модулів;

- розроблено програмні модулі керуванням контенту, користувацькими комунікаціями, персоналізації;

- проведено тестування програмних модулів.

Перед створенням програмного продукту було розроблено схеми загального алгоритму роботи додатку та роботи з базою даних, UML діаграму системи.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому завданню.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Barker D., Real World Content Modeling: A Field Guide to CMS Features and Architecture, Бостон: O'Reilly Media, 2019. 200 с.

2. Wells M., User Experience Design: An Introduction to Creating Interactive Digital Spaces, Laurence King Publishing , 2023. 160 с.

3. Under the Hood of CMS in Web Design: веб-сайт. URL: https://www.sciencedirect.com/science/article/pii/S1877050915034936

4. What is a Content Management System? the Ultimate Guide: веб-сайт. URL: https://solidwp.com/blog/content-management-system/

5. Online Social Newtwork Management Systems. State of the Art : веб-сайт. URL: https://www.webdesignbooth.com/what-is-cms-in-web-design/

6. WordPress. WordPress.: веб-сайт. URL: https://uk.wordpress.org/

7. Drupal – Open Source CMS. Drupal.: веб-сайт. URL: https://www.drupal.org/

8. Joomla Content Management System (CMS). Joomla.: веб-сайт. URL: https://www.joomla.org/

9. Magento. Magento.: веб-сайт. URL: https://www.magento.com/

10. The Ultimate Guide to Getting Started with Personalization [in 2023] веб-сайт. URL: https://ninetailed.io/blog/personalization-guide/

11. Build Recommendation Engine with Collaborative Filtering – Real Python веб-сайт. URL: https://realpython.com/build-recommendation-engine-collaborative-filtering/#what-is-collaborative-filtering

12. Jianjun N. , Yu C. , Guangyi T., Yingjuan X. Collaborative Filtering Recommendation Algorithm Based on TF-IDF and User Characteristics URL: https://www.mdpi.com/2076-3417/11/20/9554

13. 4 necessary algorithms for a product recommendation system. URL: https://gritglobal.io/blog/product-recommendation-system/

14. Pirasteh, P.; Jung, J.J.; Hwang, D. Item-based collaborative filtering with attribute correlation: A case study on movie recommen- dation. In Proceedings of the

6th Asian Conference on Intelligent Information and Database Systems, Бангкок, Тайланд, 7–9 April 2014; Volume 8398, с. 245.

15. Kumar, R.; Verma, B.K.; Rastogi, S.S. Social Popularity based SVD++ Recommender System. Int. J. Comput. Appl. 2014, с 87

16. Sun, B.; Dong, L. Dynamic Model Adaptive to User Interest Drift Based on Cluster and Nearest Neighbors. IEEE Access 2017, с. 1682–1691.

17. Wang, J.; Lan, Y.-X.; Wu, C.-Y. Survey of Recommendation Based on Collaborative Filtering. In Proceedings of the 2019 3rd International Conference on Electrical, Mechanical and Computer Engineering, ICEMCE 2019, Guizhou, China, 9–11 August 2019; Volume 1314.

18. Zarzour, H.; Jararweh, Y.; Al-Sharif, Z.A. An Effective Model-Based Trust Collaborative Filtering for Explainable Recommen- dations. In Proceedings of the 2020 11th International Conference on Information and Communication Systems, ICICS 2020, Copenhagen, Denmark, 24–26 August 2020; с. 238–242.

19. Chen, J.; Wang, B.; Ouyang, Z.; Wang, Z. Dynamic clustering collaborative filtering recommendation algorithm based on double-layer network. Int. J. Mach. Learn. Cybern. 2021, 12, с. 1097–1113.

20. Chen, Y.; Liu, Y.; Zhao, J.; Zhu, Q. Mobile edge cache strategy based on neural collaborative filtering. IEEE Access 2020, 8, 18475–18482.

21. Deng, J.; Guo, J.; Wang, Y. A Novel K-medoids clustering recommendation algorithm based on probability distribution for collaborative filtering. Knowl.-Based Syst. 2019, 175, с. 96–106.

22. Westa E., Modular Programming with Python. Монхейм: Packt Publishing, 2019. 246 с.

23. Lazy loading – Web Performance | MDN. MDN.: веб-сайт. URL: https://developer.mozilla.org/ru/docs/Web/Performance/Lazy_loading

24. What is Single Sign On (SSO) and How Does It Work?. TechTarget.: веб-сайт. URL: https://www.techtarget.com/searchsecurity/definition/single-sign-on

25. What is relational database | Oracle?. Oracle.: веб-сайт. URL: https://www.oracle.com/database/what-is-a-relational-database

26. What is Non-Relational Database | MongoDB Mongodb; веб-сайт URL: https://www.mongodb.com/databases/non-relational

27. JavaScript language overview | MDN: веб-сайт URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_overview

28. What is Python | Teradata: веб-сайт URL: https://www.teradata.com/glossary/what-is-python

29. What is Java? – Java Programming Language Explained - AWS: веб-сайт URL: https://aws.amazon.com/what-is/java/?nc1=h_ls

30. What is Angular? | Simplilearn: веб-сайт URL: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular

31. Alhijawi, B.; Kilani, Y. A collaborative filtering recommender system using genetic algorithm. Inf. Process. Manag. 2020, 57, 102310.

32. Li, X.; Li, D. An Improved Collaborative Filtering Recommendation Algorithm and Recommendation Strategy. Mob. Inf. Syst.2019, 3560968.

33. Xu, L.; Li, X.; Guo, Y. Gauss-core extension dependent prediction algorithm for collaborative filtering recommendation. Clust. Comput. 2019, 22, 11501–11511.

34. Wang, W.; Chen, J.; Wang, J.; Chen, J.; Liu, J.; Gong, Z. Trust-Enhanced Collaborative Filtering for Personalized Point of Interests Recommendation. IEEE Trans. Ind. Inform. 2020, 16, 6124–6132.

35. Wu, S.; Kou, H.; Lv, C.; Huang, W.; Qi, L.; Wang, H. Service recommendation with high accuracy and diversity. Wirel. Commun. Mob. Comput. 2020, 2020, 8822992.

36. What is Black Box Testing? | Techniques & Examples | Imperva: веб-сайт URL: https://www.imperva.com/learn/application-security/black-box-testing/

37. White/black/grey box-тестування - QALight: веб-сайт URL: https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya/

# ДОДАТКИ

# ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ

(Обов'язковий)

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

д.т.н., професор Романюк О.Н.

"19" вересня 2023 р.

**Технічне завдання**

**на магістерську кваліфікаційну роботу**

**«Інформаційна система керування контентом та користувацькими комунікаціями»**

**за спеціальністю 121 – Інженерія програмного забезпечення**

Керівник магістерської кваліфікаційної роботи:

к.т.н., доц. О.О. Коваленко

"19" вересня 2023 р.

Виконав:

студент гр.1ПІ-22м О.С. Івасьов

"19" вересня 2023 р.

Вінниця – 2023 року

**1. Найменування та галузь застосування**

Магістерська кваліфікаційна робота: «Інформаційна система керування контентом та користувацькими комунікаціями».

Галузь застосування – системи керування контентом.

**2. Підстава для розробки.**

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

**3. Мета та призначення розробки.**

Метою роботи є підвищення ефективності використання інформаційної системи керування контентом та користувацькими комунікаціями в організації. Призначення роботи – використання розробленої спеціалізованої інформаційної системи в організації.

**4. Вихідні дані для проведення НДР**

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Barker D., Real World Content Modeling: A Field Guide to CMS Features and Architecture, Бостон: O'Reilly Media, 2019. 200 с.

2. Wells M., User Experience Design: An Introduction to Creating Interactive Digital Spaces, Laurence King Publishing , 2023. 160 с.

3. Under the Hood of CMS in Web Design: веб-сайт. URL: https://www.sciencedirect.com/science/article/pii/S1877050915034936

4. What is a Content Management System? the Ultimate Guide: веб-сайт. URL: https://solidwp.com/blog/content-management-system/

5. Online Social Newtwork Management Systems. State of the Art : веб-сайт. URL: https://www.webdesignbooth.com/what-is-cms-in-web-design/

**5. Технічні вимоги**

Для функціонування систем управління навчанням достатньо задовольнити мінімальні вимоги для браузерів Інтернет Google Chrome, Firefox Mozilla. Операційні системи Windows 7-11, Linux.

**6. Конструктивні вимоги**

Архітектура програмного продукту повинна відповідати вимогам розгортання та функціонування у веб-середовищі.

Графічна та текстова документація повинна відповідати діючим стандартам України.

**7. Перелік технічної документації, що пред'являється по закінченню робіт:**

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

**8. Вимоги до рівня уніфікації та стандартизації**

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

**9. Стадії та етапи розробки:**

| №з/п | Назва етапів магістерської кваліфікаційної роботи | Строк виконання етапів роботи |
|---|---|---|
| 1 | Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі | 20.09.2023 - 27.09.2023 |
| 2 | Розробка структури та методів програмного продукту | 28.09.2023 - 10.10.2023 |
| 3 | Реалізація програми інформаційної системи керування контентом | 11.10.2023 - 27.10.2023 |
| 4 | Тестування програмного застосунку | 28.10.2023 - 15.11.2023 |

**10. Порядок контролю та прийняття.**

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

# ДОДАТОК Б – ПРОТОКОЛ ПЕРЕВІРКИ НА ПЛАГІАТ

**(Обов'язковий)**

## ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Інформаційна система керування контентом та користувацькими комунікаціями

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 22м

Науковий керівник: Коваленко О.О.

| Unicheck | |
|---|---|
| **Оригінальність** | **95,6%** |
| Схожість | 4,4% |

### Аналіз звіту подібності

**Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Івасьов О.С.

Керівник роботи _____ Коваленко О.О.

# ДОДАТОК Г – ЛІСТИНГИ ПРОГРАМИ

## (Довідниковий)

```
export class HomeComponent {
  featuredPosts: { data: Post; id: string }[] = [];
  latestPosts: { data: Post; id: string }[] = [];
   constructor(private postService: PostsService, private oAuthService: OAuthService, private authService:AuthService) {
    this.oAuthService.configure(oAuthConfig);
    this.oAuthService.loadDiscoveryDocument().then(() => {
      oAuthService.tryLoginImplicitFlow().then(() => {
        if (oAuthService.hasValidAccessToken()) {
          oAuthService.loadUserProfile().then((userProfile: any) => {
            this.authService.loginAfterGoogle(userProfile.info);
          })
        }
      })
    })
    this.postService.loadData('featured').subscribe((val) => {
      this.featuredPosts = val;
    });
    this.postService.loadData('latest').subscribe((val) => {
      this.latestPosts = val;
    });
  }
}
export class SinglePostComponent implements OnInit {
  constructor(
    private route: ActivatedRoute,
    private postService: PostsService,
    private toastr: ToastrService
  ) {}

  postData!: Post | false;
  similarPosts: { data: Post; id: string }[] = [];
```

```
  notFoundError: boolean = false;

  ngOnInit(): void {
    this.route.params.subscribe((val) => {
      this.postService.countViews(val['id'])
      this.postService
        .loadSinglePost(val['id'])
        .then((post) => {
          if (post) {
            this.postData = post as Post;
            this.postService.loadData('category',post['category'].categoryId).subscribe(posts => {
              this.similarPosts = posts
            })
            this.notFoundError = false;
          } else {
            this.notFoundError = true;
          }
        })
        .catch((error) => {
          this.toastr.error(error);
        });
    });
  }
}
```

**export *class* SingleCategoryComponent implements OnInit {**

```
  constructor(
    private route: ActivatedRoute,
    private postService: PostsService
  ) {}

  posts: { data: Post; id: string }[] = [];
  categoryObj: Params = {};

  ngOnInit(): void {
    this.route.params.subscribe((val) => {
```

```
    this.categoryObj = val;
    this.postService.loadData('category', val['id']).subscribe((posts) => {
      this.posts = posts;
    });
  });
 }
}

export class LoginComponent {
 constructor(
  private authService: AuthService,
  private toastr: ToastrService,
  private router: Router,
  private googleService: GoogleApiService
 ) {}

 async onSubmit(form: NgForm) {
  try {
   await this.authService.login(form.value.email, form.value.password);
   this.router.navigate(['/']);
  } catch (error) {
   this.toastr.error(error as any);
  }
 }
 loginWithGoogle() {
  // googleWrapper.click();
  this.googleService.signUp();
 }
}
export class HeaderComponent {
 constructor(
  private authService: AuthService,
  private toastr: ToastrService
 ) {}
```

```
  userEmail: string = '';
  isLoggedIn$: Observable<boolean> = new Observable();

ngOnInit() {
  try {
    const userData = localStorage.getItem('user');
    if (userData !== null) {
      this.userEmail = JSON.parse(userData).email
      this.isLoggedIn$ = this.authService.isLoggedIn()
    } else {
      console.log('User data not found in localStorage')
    }
  } catch(error) {
    console.error('Error with localStorage: ', error)
  }
}

  onLogOut() {
    this.authService
      .logOut()
      .then(() => {
        localStorage.removeItem('user');
        this.toastr.success('User logged out successfully');
      })
      .catch((error) => {
        this.toastr.error(error);
      });
  }
}
export class GoogleSigninComponent {

  @Output() loginWithGoogle: EventEmitter<any> = new EventEmitter<any>();

  createFakeGoogleWrapper = () => {
    const googleLoginWrapper = document.createElement('div');
```

```
    googleLoginWrapper.style.display = 'none';
    googleLoginWrapper.classList.add('custom-google-button');
    document.body.appendChild(googleLoginWrapper);
    window.google.accounts.id.renderButton(googleLoginWrapper, {
      type: 'icon',
      width: '200',
    });

    const googleLoginWrapperButton = googleLoginWrapper.querySelector(
      'div[role=button]'
    ) as HTMLElement;

    return {
      click: () => {
        googleLoginWrapperButton?.click();
      },
    };
  };

  handleGoogleLogin() {
    this.loginWithGoogle.emit(this.createFakeGoogleWrapper());
  }
}
```

**export *class* CommentListComponent implements OnInit {**

```
  constructor(
    private commentService: CommentsService,
    private router: ActivatedRoute
  ) {}

  postId: string = '';
  comments: Comment[] = [];
  commentsCount: number = 0;

  ngOnInit(): void {
```

```
    this.router.params.subscribe((val) => {
      this.postId = val['id'];
    });
    this.commentService.getComments(this.postId).subscribe((comments) => {
      this.comments = comments;
    });
    this.commentService.countComments(this.postId).then((count) => {
      this.commentsCount = count;
    });
  }
}
export class CommentFormComponent implements OnInit {
  constructor(
    private commentService: CommentsService,
    private toastr: ToastrService,
    private route: ActivatedRoute,
    private authsService: AuthService,
    private router: Router
  ) {}

  postId: string = '';

  ngOnInit(): void {
    this.route.params.subscribe((val) => {
      this.postId = val['id'];
    });
  }
  async onSubmit(form: NgForm) {
    if (!form.valid) {
      return;
    }
    const commentData = {
      name: form.value.name,
      content: form.value.comment,
      postId: this.postId,
```

```
  };
  try {
    this.authsService.isLoggedIn().subscribe(async (isLoggedIn) => {
      if (isLoggedIn) {
        await this.commentService.addComment(commentData);
        this.toastr.success('Your comment added successfully');
      } else {
        this.router.navigate(['/login']);
      }
    });
  } catch (error) {
    this.toastr.error('Failed to add your comment');
  }
 }
}
export class AuthService {
 constructor() {
  this.initLoggedStatus();
 }

 loggedIn: BehaviorSubject<boolean> = new BehaviorSubject<boolean>(false);

 async login(email: string, password: string) {
  try {
    const auth = getAuth();
    const userCredentials = await signInWithEmailAndPassword(
      auth,
      email,
      password
    );
    this.loadUser();
    this.loggedIn.next(true);
  } catch (error) {
    console.error('Authentication error: ', error);
    throw new Error('Authentication error');
```

```
  }
}

private initLoggedStatus() {
  const auth = getAuth();
  authState(auth).subscribe((user) => {
    const isLoggedIn = !!user;
    this.loggedIn.next(isLoggedIn);
  });
}

loadUser() {
  const auth = getAuth();
  authState(auth).subscribe((user) => {
    localStorage.setItem('user', JSON.stringify(user));
  });

}

loginAfterGoogle(user: any) {
  this.loggedIn.next(true);
  localStorage.setItem('user', JSON.stringify(user));
}

async logOut() {
  const auth = getAuth();
  try {
    await signOut(auth);
    this.loggedIn.next(false)
  } catch(error) {
    throw new Error('Error during logout');
  }
}

isLoggedIn() {
```

```
    return this.loggedIn.asObservable()
  }
}
```

```
export class CategoriesService {
  constructor(private afs: Firestore) {}

  loadData(): Observable<CategoryArray[]> {
    return new Observable<CategoryArray[]>((observer) => {
      const unsubscribe = onSnapshot(
        collection(this.afs, 'categories'),
        (snapshot) => {
          let data: CategoryArray[] = [];
          snapshot.docs.forEach((doc) => {
            data.push({
              data: doc.data(),
              id: doc.id,
            });
          });
          observer.next(data);
        },
        (error) => {
          observer.error(error);
        }
      );
      return () => {
        unsubscribe();
      };
    });
  }
}
```

```
export class CommentsService {
  constructor(private afs: Firestore) {}
```

```
async addComment(commentData: {
  name: string;
  content: string;
  postId: string;
}) {
  const commentObjData = { ...commentData, createdAt: new Date() };
  try {
    await addDoc(collection(this.afs, 'comments'), commentObjData);
  } catch (error) {
    console.error('Error adding comment');
    throw new Error('Failed to add new comment');
  }
}

async countComments(postId: string) {
  const q = query(
    collection(this.afs, 'comments'),
    where('postId', '==', postId)
  );
  const snapshot = await getCountFromServer(q);
  return snapshot.data().count;
}

getComments(postId: string): Observable<Comment[]> {
  return new Observable((observer) => {
    const unsubscribe = onSnapshot(
      query(collection(this.afs, 'comments'), where('postId', '==', postId)),
      (snapshot) => {
        const data: Comment[] = [];
        snapshot.docs.forEach((doc) => {
          data.push({
            name: doc.data()['name'],
            content: doc.data()['content'],
            id: doc.id,
```

```
            postId: doc.data()['postId'],
            createdAt: doc.data()['createdAt'],
          });
        });
        observer.next(data);
      },
      (error) => {
        observer.error(error);
      }
    );
  });
 }
}
```

**export *class* PostsService {**

```
 constructor(private afs: Firestore) {}

 loadData(
  queryType: 'featured' | 'latest' | 'category',
  id?: string
 ): Observable<{ data: Post; id: string }[]> {
  const queryMap = {
   featured: () =>
    query(collection(this.afs, 'posts'), where('isFeatured', '==', true)),
   latest: () => query(collection(this.afs, 'posts'), orderBy('createdAt')),
   category: () =>
    query(
     collection(this.afs, 'posts'),
     where('category.categoryId', '==', id)
    ),
  };

  const queryFn = queryMap[queryType];
  if (!queryFn) {
   throw new Error('Invalid query type');
```

```
  }
  return new Observable((observer) => {
    const unsubscribe = onSnapshot(
      queryFn(),
      (snapshot) => {
        const data: { data: Post; id: string }[] = snapshot.docs.map(
          (doc) => {
            const postData: DocumentData = doc.data();
            return { data: postData as Post, id: doc.id };
          }
        );
        observer.next(data);
      },
      (error) => {
        observer.error(error);
      }
    );

    return () => {
      unsubscribe();
    };
  });
}

async loadSinglePost(id: string) {
  try {
    const docSnap = await getDoc(doc(this.afs, 'posts', id));
    if (docSnap.exists()) {
      return docSnap.data();
    } else return false;
  } catch (error) {
    console.error('Error loading post', error);
    throw new Error('Failed to load post');
  }
}
```

```
  async countViews(id: string) {
    await updateDoc(doc(this.afs, 'posts', id), {
      views: increment(1),
    });
  }
}
export class SubscribersService {
  constructor(private afs: Firestore) {}

  async addSubs(subData: Sub) {
    try {
      await addDoc(collection(this.afs, 'subscribers'), subData);
    } catch (error) {
      console.error('Error adding subscriber');
      throw new Error('Failed to add new subscriber');
    }
  }

  async checkSubs(email: string) {
    try {
      const q = query(
        collection(this.afs, 'subscribers'),
        where('email', '==', email)
      );
      const snapshot = await getDocs(q);
      return !snapshot.empty;
    } catch (error) {
      console.error('Error checking subs');
      throw new Error('Failed to check subscribers');
    }
  }
}

export class HeaderComponent implements OnInit {
```

```
  constructor(
    private authService: AuthService,
    private toastr: ToastrService
  ) {}

  userEmail: string = '';
  isLoggedIn$: Observable<boolean> = new Observable();

  ngOnInit(): void {
    try {
      const userData = localStorage.getItem('user');
      if (userData !== null) {
        this.userEmail = JSON.parse(userData).email;
        this.isLoggedIn$ = this.authService.isLoggedIn();
      } else {
        console.log('User data not found in localStorage.');
      }
    } catch (error) {
      console.error('Error with localStorage: ', error);
    }
  }

  onLogOut() {
    this.authService
      .logOut()
      .then(() => {
        localStorage.removeItem('user');
        this.toastr.success('User logged out successfully');
      })
      .catch((error) => {
        this.toastr.error(error);
      });
  }
}
```

```
export class LoginComponent {
  constructor(
    private authService: AuthService,
    private toastr: ToastrService,
    private router: Router,
  ) {}

  async onSubmit(form: NgForm) {
    try {
      await this.authService.login(form.value.email, form.value.password);
      this.router.navigate(['/']);
    } catch (error) {
      this.toastr.error(error as any);
    }
  }
}
export class CategoriesComponent implements OnInit {
  constructor(
    private categoryService: CategoriesService,
    private toastr: ToastrService
  ) {}

  categoryArray: CategoryArray[] = [];
  formCategory: string = '';
  formStatus: string = 'Add';
  categoryId: string = '';

  ngOnInit(): void {
    this.categoryService.loadData().subscribe({
      next: (data) => {
        this.categoryArray = data;
      },
      error: (error) => {
```

```
      console.error('Error fetching data:', error);
      this.toastr.error('Error fetching data from the database');
    },
  });
}

async onSubmit(formData: NgForm) {
  try {
    let categoryData: Category = {
      category: formData.value.category,
    };
    if (this.formStatus === 'Add') {
      await this.categoryService.saveData(categoryData);
      this.toastr.success('Data saved successfully');
      formData.reset();
    } else if (this.formStatus === 'Edit') {
      await this.categoryService.updateData(this.categoryId, {
        ...categoryData,
      });
      this.toastr.success('Data updated successfully');
      formData.reset();
      this.formStatus = 'Add';
    }
  } catch (error) {
    this.toastr.error((error as any).message, 'Error');
  }
}

onEdit(category: string, id: string) {
  this.formCategory = category;
  this.formStatus = 'Edit';
  this.categoryId = id;
}

async onDelete(id: string) {
```

```
  try {
    await this.categoryService.deleteData(id);
    this.toastr.success('Data deleted successfully');
  } catch (error) {
    this.toastr.error('An error occured while deleting data');
  }
 }
}
```

```
export class HeaderComponent implements OnInit {
 constructor(
  private authService: AuthService,
  private toastr: ToastrService
 ) {}

 userEmail: string = '';
 isLoggedIn$: Observable<boolean> = new Observable();

 ngOnInit(): void {
  try {
   const userData = localStorage.getItem('user');
   if (userData !== null) {
    this.userEmail = JSON.parse(userData).email;
    this.isLoggedIn$ = this.authService.isLoggedIn();
   } else {
    console.log('User data not found in localStorage.');
   }
  } catch (error) {
   console.error('Error with localStorage: ', error);
  }
 }

 onLogOut() {
  this.authService
   .logOut()
```

```
    .then(() => {
      localStorage.removeItem('user');
      this.toastr.success('User logged out successfully');
    })
    .catch((error) => {
      this.toastr.error(error);
    });
  }
}
export class NewPostComponent implements OnInit {
  permalink: string = '';
  imgSrc: any = './assets/placeholder-image.png';
  selectedImg!: File;
  categories: CategoryArray[] = [];
  postForm!: FormGroup;
  formStatus: string = 'Add New';
  postId: string = '';

  constructor(
    private categoryService: CategoriesService,
    private fb: FormBuilder,
    private postService: PostService,
    private route: ActivatedRoute,
    private toastr: ToastrService
  ) {
    this.route.queryParams.subscribe((val) => {
      if (val['id']) {
        this.postService.loadOneData(val['id']).subscribe((post) => {
          this.postForm = this.fb.group({
            title: [
              post['title'],
              [Validators.required, Validators.minLength(10)],
            ],
            permalink: new FormControl(
              { value: post['permalink'], disabled: true },
```

```
          Validators.required
        ),
        excerpt: [
          post['excerpt'],
          [Validators.required, Validators.minLength(50)],
        ],
        category: [
          `${post['category'].categoryId}-${post['category'].category}`,
          Validators.required,
        ],
        postImg: ['', Validators.required],
        content: [post['content'], Validators.required],
      });
      this.imgSrc = post['postImgPath'];
      this.formStatus = 'Edit';
      this.postId = val['id'];
    });
  } else {
    this.postForm = this.fb.group({
      title: ['', [Validators.required, Validators.minLength(10)]],
      permalink: new FormControl(
        { value: '', disabled: true },
        Validators.required
      ),
      excerpt: ['', [Validators.required, Validators.minLength(50)]],
      category: ['', Validators.required],
      postImg: ['', Validators.required],
      content: ['', Validators.required],
    });
  }
 });
}

ngOnInit(): void {
 this.categoryService.loadData().subscribe({
```

```
    next: (data) => {
      this.categories = data;
    },
    error: (error) => {
      console.error('Error fetching data:', error);
      this.toastr.error('An error occured while loading data');
    },
  });
}

onTitleChanged($event: any) {
  const title = $event.target.value;
  this.permalink = title.replace(/\s/g, '-');
}

showPreview($event: any) {
  const reader = new FileReader();
  reader.onload = (e) => {
    this.imgSrc = e.target?.result;
  };
  reader.readAsDataURL($event?.target.files[0]);
  this.selectedImg = $event.target.files[0];
}

async onSubmit() {
  try {
    let splitted = this.postForm.value.category.split('-');
    const postData: Post = {
      title: this.postForm.value.title,
      permalink: this.permalink,
      category: {
        categoryId: splitted[0],
        category: splitted[1],
      },
      postImgPath: '',
```

```
      excerpt: this.postForm.value.excerpt,
      content: this.postForm.value.content,
      isFeatured: false,
      views: 0,
      status: 'new',
      createdAt: new Date(),
    };
    await this.postService.uploadImage(
      this.selectedImg,
      postData,
      this.formStatus,
      this.postId
    );
    this.toastr.success('Post saved successfully');
    this.postForm.reset();
    this.imgSrc = './assets/placeholder-image.png';
  } catch (error) {
    this.toastr.error(
      (error as any).message,
      'An error occured while submiting form'
    );
  }
 }

 get fc() {
  return this.postForm.controls;
 }
}
```

**export *class* AuthService {**

```
 loggedIn: BehaviorSubject<boolean> = new BehaviorSubject<boolean>(false);

 constructor(private router: Router) {
  this.initLoggedStatus();
 }
```

```
async login(email: string, password: string) {
  try {
    const auth = getAuth();
    const userCredentials = await signInWithEmailAndPassword(
      auth,
      email,
      password
    );
    this.loadUser();
    this.loggedIn.next(true);
  } catch (error) {
    console.error('Authentication error: ', error);
    throw new Error('Authentication error');
  }
}

private initLoggedStatus() {
  const auth = getAuth();
  authState(auth).subscribe((user) => {
    const isLoggedIn = !!user;
    this.loggedIn.next(isLoggedIn);
  });
}

loadUser() {
  const auth = getAuth();
  authState(auth).subscribe((user) => {
    localStorage.setItem('user', JSON.stringify(user));
  });
}

async logOut() {
  const auth = getAuth();
  try {
    await signOut(auth);
```

```
    this.loggedIn.next(false);
    this.router.navigate(['/login']);
   } catch (error) {
    throw new Error('Error during logout');
   }
  }

  isLoggedIn() {
   return this.loggedIn.asObservable();
  }

  getLoggedInStatus(): boolean {

   return this.loggedIn.getValue();
  }
}
```

```
export class CategoriesService {
  constructor(private afs: Firestore) {}

  async saveData(data: Category) {
   try {
    await addDoc(collection(this.afs, 'categories'), data);
   } catch (error) {
    console.error('Error saving data: ', error);
    throw new Error('Failed to save data');
   }
  }

  loadData(): Observable<CategoryArray[]> {
   return new Observable<CategoryArray[]>((observer) => {
    const unsubscribe = onSnapshot(
     collection(this.afs, 'categories'),
     (snapshot: QuerySnapshot) => {
      const data: CategoryArray[] = [];
```

```
        snapshot.docs.forEach((doc) => {
          data.push({
            data: doc.data(),
            id: doc.id,
          });
        });
        observer.next(data);
      },
      (error) => {
        observer.error(error);
      }
    );
  });
 }

 async updateData(id: string, editData: { category: string }) {
  try {
    await updateDoc(doc(this.afs, 'categories', id), editData);
  } catch (error) {
    console.error('Error updating data', error);
    throw new Error('Failed to update data');
  }
 }

 async deleteData(id: string) {
  try {
    await deleteDoc(doc(this.afs, 'categories', id));
  } catch (error) {
    console.error('Error deleting data', error);
    throw new Error('Failed to delete data');
  }
 }
}
export class PostService {
 constructor(private afs: Firestore) {}
```

```
async uploadImage(
  file: File,
  postData: Post,
  formStatus: string,
  id: string
) {
  try {
    const storage = getStorage();
    const filePath = `postIMG/${Date.now()}`;
    const storageRef = ref(storage, filePath);
    await uploadBytes(storageRef, file);
    postData.postImgPath = await getDownloadURL(ref(storage, filePath));
    if (formStatus === 'Edit') {
      await this.updateData(id, postData);
    } else {
      await this.saveData(postData);
    }
  } catch (error) {
    console.error('Error saving data', error);
    throw new Error('Failed to update data');
  }
}

async saveData(postData: Post) {
  try {
    await addDoc(collection(this.afs, 'posts'), postData);
  } catch (error) {
    console.error('Error adding post: ', error);
    throw new Error('Failed to save data');
  }
}

loadData(): Observable<CategoryArray[]> {
  return new Observable((observer) => {
```

```
    const unsubscribe = onSnapshot(
      collection(this.afs, 'posts'),
      (snapshot: QuerySnapshot) => {
        const data: { data: DocumentData; id: string }[] = [];
        snapshot.docs.forEach((doc) => {
          data.push({
            data: doc.data(),
            id: doc.id,
          });
        });
        observer.next(data);
      },
      (error) => {
        observer.error(error);
      }
    );
  });
}

loadOneData(id: string): Observable<DocumentData> {
  return new Observable((observer) => {
    onSnapshot(
      doc(this.afs, 'posts', id),
      (doc) => {
        observer.next(doc.data());
      },
      (error) => {
        observer.error(error);
      }
    );
  });
}

async updateData(id: string, postData: any) {
  try {
```

```
    const docSnap = await getDoc(doc(this.afs, 'posts', id));
    if (docSnap.exists()) {
      const postImgPath = docSnap.data()['postImgPath'];
      this.deleteImage(postImgPath);
      await updateDoc(doc(this.afs, 'posts', id), postData);
    }
  } catch (error) {
    console.error('Error updating document', error);
    throw new Error('Failed to update data');
  }
}

async deleteImage(postImgPath: string) {
  try {
    const storage = getStorage();
    await deleteObject(ref(storage, postImgPath));
  } catch (error) {
    console.error('Error deleting image: ', error);
    throw new Error('Failed to delete image');
  }
}

async deleteData(postImgPath: string, id: string) {
  try {
    await deleteDoc(doc(this.afs, 'posts', id));
    await this.deleteImage(postImgPath);
  } catch (error) {
    console.error('Error deleting data: ', error);
    throw new Error('Failed to delete data');
  }
}

async markFeatured(id: string, featuredData: { isFeatured: boolean }) {
  try {
    await updateDoc(doc(this.afs, 'posts', id), featuredData);
```

```
    } catch (error) {
      console.error('Error', error);
      throw new Error('Failed to mark post');
    }
  }
}
export class SubscribersService {
  constructor(private afs: Firestore) {}

  loadData(): Observable<Sub[]> {
    return new Observable((observer) => {
      const unsubscribe = onSnapshot(
        collection(this.afs, 'subscribers'),
        (snapshot: QuerySnapshot) => {
          const data: Sub[] = [];
          snapshot.docs.forEach((doc) => {
            data.push({
              name: doc.data()['name'],
              email: doc.data()['email'],
              id: doc.id
            });
          });
          observer.next(data);
        },
        (error) => {
          observer.error(error);
        }
      );
    });
  }

  async deleteData(id: string) {
    try {
      await deleteDoc(doc(this.afs, 'subscribers', id));
    } catch (error) {
```

```
    console.error('Error deleting data', error);
    throw new Error('Failed to delete data');
   }
  }
}
```

```
export class SubscribersComponent implements OnInit {
  constructor(
   private subService: SubscribersService,
   private toastr: ToastrService
  ) {}

  subscribers: Sub[] = [];

  ngOnInit(): void {
   this.subService.loadData().subscribe((val) => {
    this.subscribers = val;
   });
  }

  async onDelete(id: string) {
   try {
    await this.subService.deleteData(id);
    this.toastr.success('Subcsriber deleted successfully');
   } catch (error) {
    this.toastr.error('Failed to delete subscriber');
   }
  }
}
```

# ДОДАТОК Д

**(Довідниковий)**

## ІЛЮСТРАТИВНА ЧАСТИНА

«Інформаційна система керування контентом та користувацькими комунікаціями»

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Вінницький національний технічний університет
Кафедра програмного забезпеченння

# Магістерська кваліфікаційна робота
## на тему:

ІНФОРМАЦІЙНА СИСТЕМА КЕРУВАННЯ КОНТЕНТОМ ТА КОРИСТУВАЦЬКИМИ КОМУНІКАЦІЯМИ

Роботу виконав:
студент групи 1ПІ-22м
Івасьов Олексій Станіславович

Науковий керівник:
Кандидат технічних наук, доцент
Коваленко Олена Олексіївна

Рисунок Д1 – Назва роботи

## Актуальність теми

Сучасний інформаційний вік вимагає від підприємств та організацій високофункціональних інформаційних систем, спрямованих на ефективне управління контентом та користувацькими комунікаціями. Швидкий розвиток технологій та зростання обсягу інформації ставлять перед сучасними організаціями завдання забезпечення не лише надійного зберігання даних, але й активного управління ними для досягнення стратегічних цілей.

Інформаційні системи керування контентом та користувацькими комунікаціями відіграють ключову роль у формуванні іміджу організації, взаємодії зі зацікавленими сторонами та забезпеченні ефективної внутрішньої комунікації. Застосування сучасних технологій у цих системах може значно поліпшити якість комунікації та сприяти підвищенню продуктивності діяльності організації.

Враховуючи постійні зміни у сфері інформаційних технологій, важливо проводити наукові дослідження для розробки та впровадження новітніх рішень в області керування контентом та комунікаціями. Дана дипломна робота спрямована на аналіз сучасних тенденцій, визначення найефективніших підходів та розробку інформаційної системи, що відповідає вимогам сучасного бізнесу та сприяє підвищенню конкурентоспроможності організації.

Рисунок Д2 – Актуальність теми

## Мета, предмет, об’єкт, завдання

**Мета та завдання дослідження.** Метою роботи є підвищення ефективності використання інформаційної системи керування контентом та користувацькими комунікаціями в організації. Для досягнення цієї мети передбачається вирішення наступних завдань:

- розгляд сучасних тенденцій у сфері керування контентом та користувацькими комунікаціями.

- запропонувати методи покращення функціоналу інформаційної системи. Це включає в себе розробку нових можливостей, які сприятимуть зручності користування системою, підвищенню її продуктивності та забезпеченню більш ефективної взаємодії з користувачами.

- розробка інформаційної системи керування контентом та користувацькими комунікаціями.

**Об’єкт дослідження** – процес керуванням контентом.

**Предмет дослідження** – методи та програмні засоби керуванням контенту.

Рисунок Д3 – Мета, предмет, об’єкт, завдання

## Наукова новизна

**Наукова новизна одержаних результатів.**

- **отримав подальшого розвитку метод інтеграції інформаційної системи з соцмережами**, який, на відміну від існуючих, дозволяє не лише авторизуватись через соціальні мережі, а також обмінюватись контентом між системою та соціальними профілями користувачів, що сприяє покращенню показників залучення аудиторії.

- **отримав подальшого розвитку метод аналізу й відстеження користувацької активності**, який, на відміну від існуючих, об’єднує в собі метод рекомендацій на основі вмісту та метод спільної фільтрації, що дозволяє оптимізувати персоналізацію системи за визначеними параметрами для кращої взаємодії з користувачами за цільовими показниками відповідності контенту.

Рисунок Д4 – Наукова новизна

Практична цінність, особистий внесок

**Практичне значення одержаних результатів.** Практична цінність одержаних результатів полягає у можливості практичного використання розробленої спеціалізованої інформаційної системи.

**Особистий внесок.** Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, автору належать: розробка інформаційна системи керування контентом та користувацькими комунікаціями та пропозиції та способи покращення інформаційних систем керування контентом.

**Апробація матеріалів роботи.** Матеріали роботи доповідалися та обговорювалися на науково-технічній міжнародній онлайн-конференції «МІСЦЕ УКРАЇНИ У СВІТОВОМУ РОЗВИТКУ НАУКИ ТА ТЕХНІКИ» м.Хмельницький та на всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету.

Рисунок Д5 – Практична цінність, особистий внесок

Аналоги


Рисунок Д6 – Аналоги

Моделі персоналізації

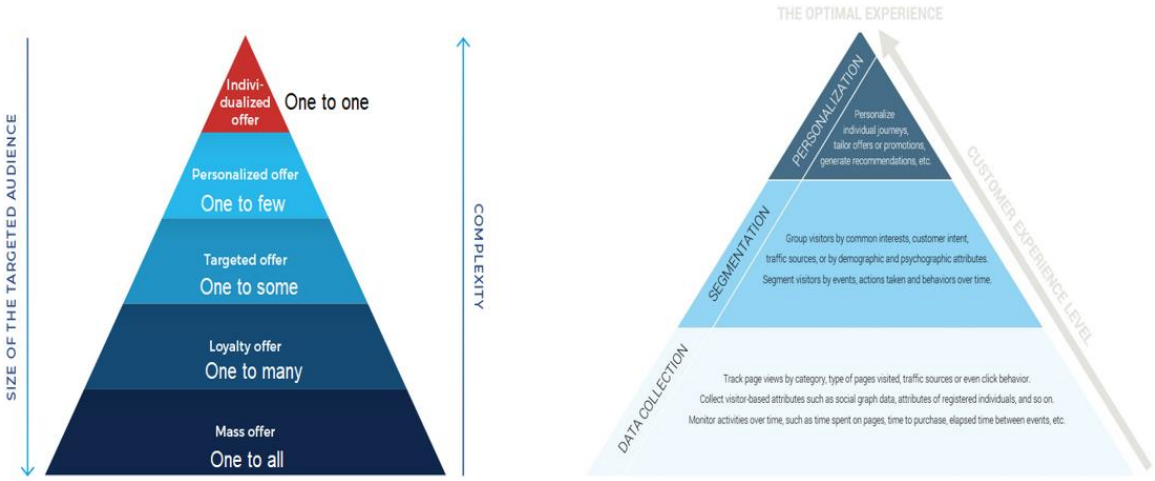


Рисунок Д7 – Моделі персоналізації

Методи фільтрації

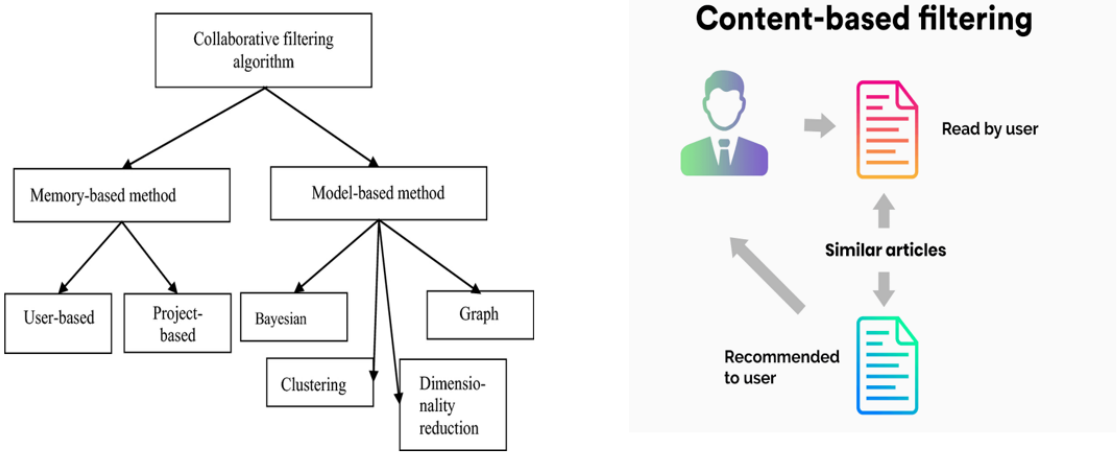


Рисунок Д8 – Методи фільтрації

Моделі рекомендації

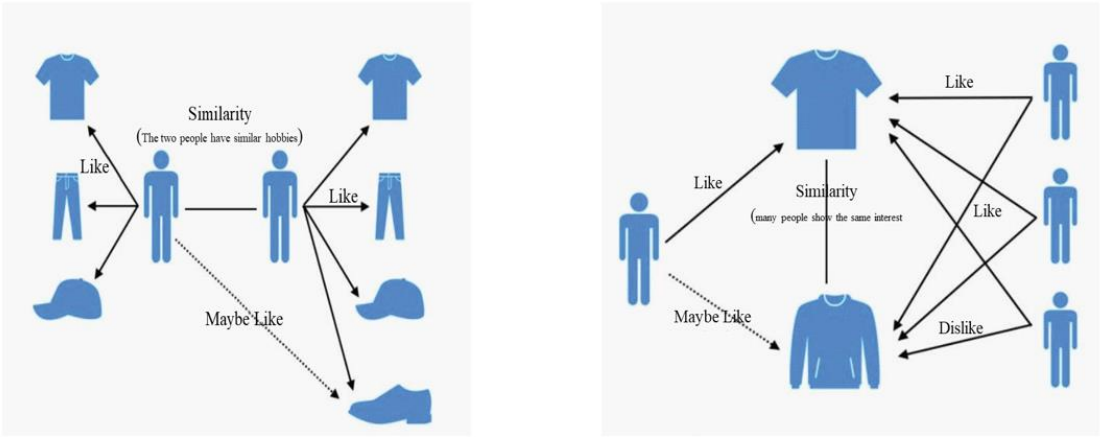


Рисунок Д9 – Моделі рекомендації

Гібридна модель рекомендацій

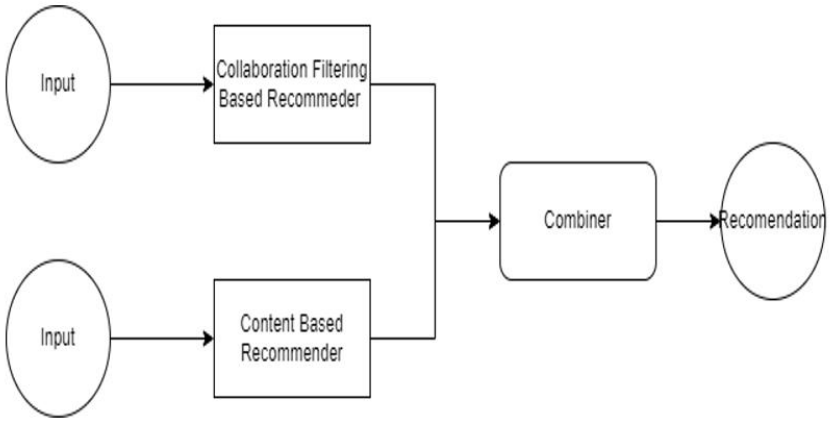


Рисунок Д10 – Гібридна модель рекомендацій

Архітектура застосунку

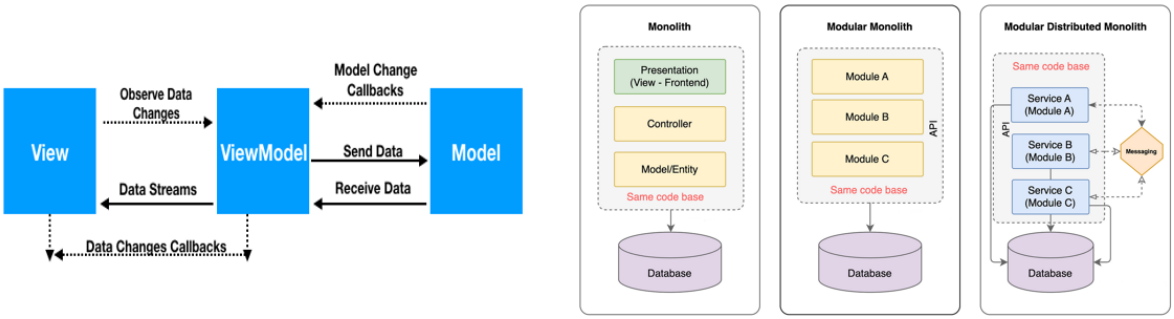


Рисунок Д11 – Архітектура застосунку

Загальний алгоритм додатку

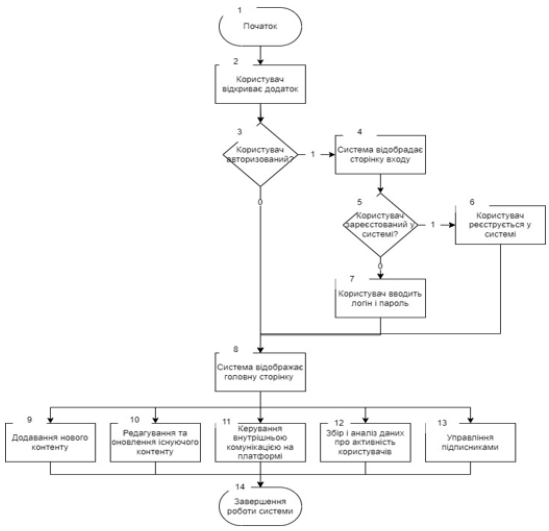


Рисунок Д12 – Загальний алгоритм застосунку

ULM діаграма системи

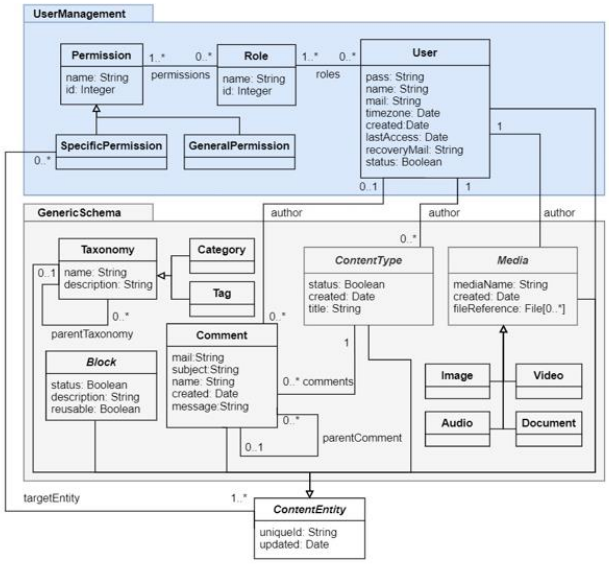


Рисунок Д13 – ULM діаграма системи

Тестування модулю інтерграції з соцмережами

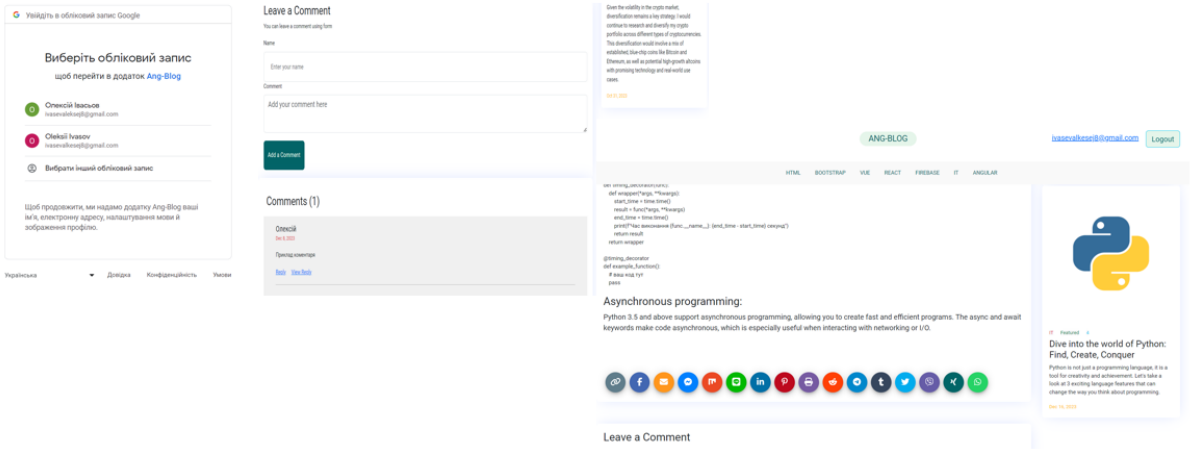


Рисунок Д14 – Тестування модулю інтеграції з соцмережами

Тестування модулю персоналізації

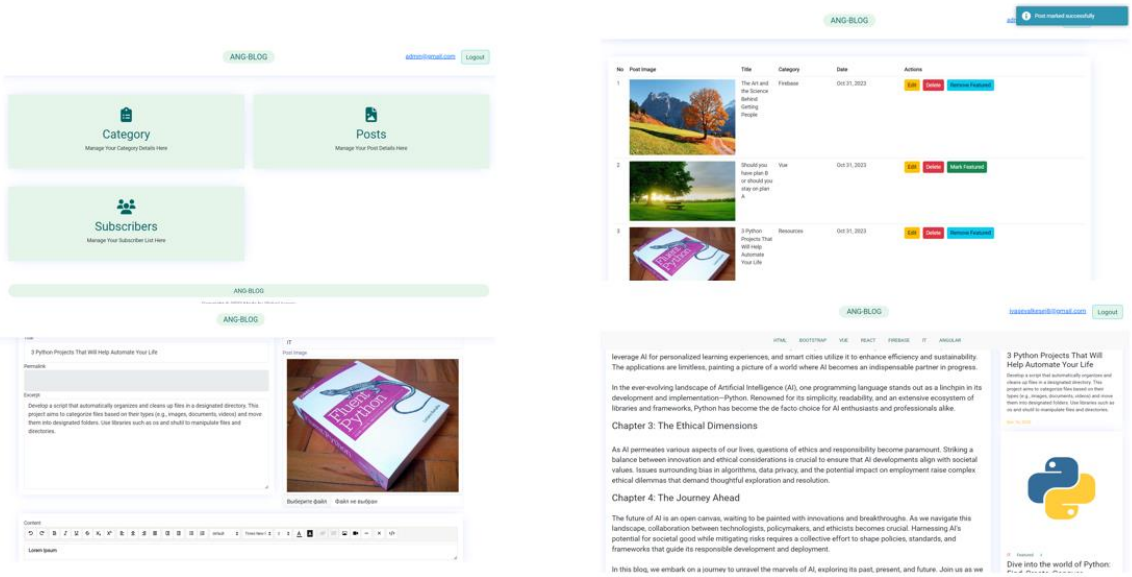


Рисунок Д15 – Тестування модулю персоналізації