ДЕРЖАВНИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка Web-орієнтованої системи для обміну контентом»

Студента 4 курсу, 8 групи, Кушніра Євгенія Олександровича

спеціальності
122 «Комп'ютерні науки» _____ *підпис студента*

Науковий керівник
кандидат технічних наук, доцент _____ *підпис керівника* Томашевська Тетяна Володимирівна

Гарант освітньої програми
кандидат технічних наук, доцент _____ *підпис керівника* Демідов Павло Георгійович

Київ 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О. І.
«12» грудня 2022р.

Завдання
на випускну кваліфікаційну роботу (проект) студенту

Кушніру Євгенію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи (проекту)
«Розробка Web-орієнтованої системи для обміну контентом»
Затверджена наказом ректора від *«09» грудня 2022 р. № 3332*
2. Строк здачі студентом закінченої роботи *30 травня 2023 року*
3. Цільова установка та вихідні дані до роботи
Мета роботи: проектування та розробка системи для обміну текстовою інформацією, фото та відео контентом
Об'єкт дослідження: процеси розробки Web-орієнтованих систем
Предмет дослідження: Web-орієнтована система обміну контентом
4. Перелік графічного матеріалу_____
_____
_____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Томашевська Т. В.	15.12.2022 р.	15.12.2022 р.
2	Томашевська Т. В.	15.12.2022 р.	15.12.2022 р.
3	Томашевська Т. В.	15.12.2022 р.	15.12.2022 р.

6. Зміст випускної кваліфікаційної роботи (проекту) (перелік питань за кожним розділом)

ВСТУП
РОЗДІЛ 1. ОСНОВНІ ПРИНЦИПИ ПОБУДОВИ ВЕБ-ДОДАТКУ
1.1 Поняття веб-додатку
1.2 Підходи до створення веб-додатку
1.3 Огляд готових рішень для систем обміну контентом
Висновки до розділу 1
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ОБМІНУ КОНТЕНТОМ
2.1 Вимоги до системи обміну контентом
2.2 Моделювання системи обміну контентом
2.2 Розробка проєктного рішення для системи обміну контентом
Висновки до розділу 2
РОЗДІЛ 3. СТВОРЕННЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ОБМІНУ КОНТЕНТОМ
3.1 Обгрунтування стеку технологій для розробки веб-додатку
3.2 Програмна реалізація системи обміну контентом
3.3 Тестування системи обміну контентом
Висновки до розділу 3
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТКИ

7. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної роботи*	*04.10.2022*	*04.10.2022*
2	*Розробка та затвердження завдання на випускну кваліфікаційну роботу*	*15.12.2022*	*15.12.2022*
3	*Вступ*	*03.02.2023*	*03.02.2023*
4	*Розділ 1. Основні принципи побудови веб-додатку*	*28.02.2023*	*28.02.2023*
5	*Розділ 2. Проєктування системи обміну контентом*	*06.04.2023*	*06.04.2023*
6	*Розділ 3. Створення Веб-орієнтованої системи обміну контентом*	*12.05.2023*	*12.05.2023*
7	*Висновки*	*15.05.2023*	*15.05.2023*
8	*Здача випускної кваліфікаційної роботи на кафедру науковому керівнику*	*30.05.2023*	*30.05.2023*
9	*Попередній захист випускної кваліфікаційної роботи*	*31.05.2023 -01.06.2023*	*31.05.2023 -01.06.2023*
10	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*	*02.06.2023*	*02.06.2023*
12	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*	*05.06.2023*	*05.06.2023*
13	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання «15» грудня 2022 р.

Керівник випускної кваліфікаційної роботи (проекту)
Томашевська Т. В.
(прізвище, ініціали, підпис)

Гарант освітньої програми Демідов П.Г.
(прізвище, ініціали, підпис)

Завдання прийняв до виконання студент-дипломник
Кушнір Є.О.
(прізвище, ініціали, підпис)

9. Відгук керівника випускної кваліфікаційної роботи (проекту)

Керівник випускної кваліфікаційної роботи (проекту)
_____30.05.2023 р.
(підпис, дата)

10. Висновок про випускну кваліфікаційну роботу (проект)

Випускна кваліфікаційна робота (проект) студента Кушнір Є.О.
(прізвище, ініціали)
може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми_____ Демідов П.Г.
(підпис, прізвище, ініціали)

Завідувач кафедри _____ Пурський О.І.
(підпис, прізвище, ініціали)
«_____»_____2023 р.

Анотація

У випускній кваліфікаційній роботі проведено комплексне дослідження та розробка Web-орієнтованої системи обміну контентом, спрямованої на популяризацію та розповсюдження ігрової інформації в українському сегменті за допомогою зручної та ефективної користувацької платформи. Проведено детальний огляд підходів до розробки Web-орієнтованих систем та технологічного стеку, сформульовано вимоги до системи обміну контентом, побудовано UML-діаграми, а саме: діаграму варіантів використання, діаграму послідовності та діаграму класів. На основі цього аналізу було розроблено Web-орієнтовану систему обміну контентом, що складається з клієнтського додатку (фронтенду) та веб-сервера (бекенду).

Ключові слова: Web -орієнтовані системи, система обміну контентом, UML-діаграми, фронтенд, бекенд, вимоги до системи обміну контентом.

Annotation

The graduation qualification work encompasses a comprehensive research and development of a web-oriented content exchange system aimed at popularizing and disseminating gaming information in the Ukrainian segment through a convenient and efficient user platform has been conducted in the graduation qualifying work. A detailed review of approaches to the development of web-oriented systems and technological stack was performed, requirements for the content exchange system were formulated, and UML diagrams were constructed, including the use case diagram, sequence diagram, and class diagram. Based on this analysis, a web-oriented content exchange system consisting of a client application (frontend) and a web server (backend) was developed.

Keywords: web-oriented systems, content exchange system, UML diagrams, frontend, backend, requirements for the content exchange system.

ЗМІСТ

ПЕРЕЛІК УМОНИХ ПОЗНАЧЕНЬ

Бандл – це один файл, або кілька файлів об’єднаних в один, що містить всі залежності та код програми.

Бандлер – це програма яка збирає, об’єднує кілька файлів JavaScript, CSS та інші ресурси в один або кілька бандлів.

Обліковий запис користувача – це набір інформації, що може ідентифікувати користувача на веб-сайті та надати доступ до визначеної функціональності.

Сторінка «персонажа» – це веб-сторінка яка містить різноманітну інформацію (фото, відео, текст) про персонажа з ігор чи інших розважальних сервісів.

Сторінка «Тайва через фото» – це веб-сторінка що містить мультимедійний (фото) контент з ігровими елементами, такі як: ландшафт, персонажі, механіки гри, тощо.

Категорія «улюблені фото» – це список з фото контентом, що прив’язаний до облікового запису користувача.

Список «улюблених персонажів» – це список який містить інформацію про персонажа, таку як: ім’я, фото та інші, що прив’язані до облікового запису користувача.

БД – база даних.

REST – архітектурний стиль для побудови розподілених систем.

API – програмний інтерфейс програми.

NPM (Node Package Module) – стандартний менеджер пакетів для JavaScript.

CSS – мова таблиць стилів.

SDK (Software Development Kit) – це набір інструментів, бібліотек, компонентів, що дозволяють розробнику в зручному способі взаємодіяти із платформою.

ВСТУП

Новий твіт, повідомлення в месенджері, брати участь в онлайн конференції, перегляд популярного відео, залишити відгук під постом – це все результат безперервної інтеграції людини в системи обміном контентом. Такі системи є різними за реалізацією та специфікою їх використання, але всі вони мають спільну ознаку – це надання можливість поширювати та отримувати будь-яку інформації незалежно від місця та часу перебування користувача.

Системи обміном контентом безпосередньо створюються на платформи «Android», «Windows», «Linux», але найбільше розповсюдження, завдяки системі «Internet», мають Web-орієнтовані системи. Оскільки вони не прив'язані до платформи на якій запускаються, а отже кількість користувачів які будуть користуватись такою системою є значно вища в порівнянні з іншими реалізаціями такої системи.

Актуальність тематики дослідження зумовлена її важливістю в дослідженні та популяризації нової ніші для українського сегмента. Веб-додаток спрямований на конкретний тип контенту, а саме текстова інформація, відео та фото галерея. Саме ж створення системи обміну контентом це безпосередньо творчий та клопіткий процес, який вимагає від дослідника не лише знань з технічної дисципліни, а ще впевнених знань з теоретичної сторони та застосування їх на практиці. Окрім цього досліднику також потрібно застосовувати свої творчі уміння для розв'язування задач, уміння критично думати та знаходити різні шляхи вирішення що не будуть відхиляти від суті досліджуваного об'єкта.

Метою і завдання дослідження є проектування та розробка системи для обміну текстовою інформацією, фото та відео контентом. Для досягнення мети потрібно вирішити наступні **завдання**:

- Проаналізувати вимоги для веб-орієнтованої системи;
- Дослідити стек технологій, що використовується для реалізації веб-орієнтованих системи;

- Проаналізувати всі аспекти розробки веб-додатку;
- Реалізувати веб-орієнтовану систему обміну контентом.

Об’єктом дослідження є процеси розробки Web-орієнтованих систем.

Предметом дослідження є Web-орієнтована системи обміном контентом, як засіб для отримання та розповсюдження інформації та технології їх розробки.

Методи дослідження. У даній роботі теоретичною основою досліджень є:

- Методи аналізу літератури та наукових джерел: огляд публікацій пов’язаних з розробкою системи для обміну контентом;
- Методи управління вимогами: визначення вимог користувача, а саме функціоналах та не функціоналах вимог;
- Методи алгоритмічного програмування для створення функціоналу веб-додатку;
- Методи аналізу для вибору відповідних технологій, платформи для розробки системи обміном контентом.

Практичне значення. Програмна реалізація системи обміном контентом надає можливість користувачу ознайомитись із мультимедійним контентом та текстовою інформацією про певну гру чи інший розважальний сервіс, а також популяризація таких сервісів в українському сегменті з метою створення об’єднаної спільноти.

Структура та обсяг випускної кваліфікаційної роботи. Випускна кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел із 48 найменувань, 3 додатків і містить 68 сторінки основного тексту, 32 рисунків і 4 таблиці.

Результати досліджень пройшли апробацію на студентській науковій конференції «Інформаційні технології та кібербезпека в умовах воєнного часу», за темою «Проблеми розробки сучасних веб-сайтів», Київ 13 квітня 2023 року.

РОЗДІЛ 1.
ОСНОВНІ ПРИНЦИПИ ПОБУДОВИ ВЕБ-ДОДАТКУ

1.1 Поняття Веб-додаток

Веб-додаток – це програма або програмне забезпечення яке можна відкрити за допомогою браузера, або за допомогою його елементів як браузерного двигуна. Веб-інтерфейс розробляється за допомогою HTML, CSS, JavaScript, тоді як серверна частина (веб-сервер) може розроблятись з використанням наступних мов програмування – Java, Python, PHP, NodeJS, Ruby та інші.

До особливостей веб-додатку відносять:

- Кроссплатформенність: виконання коду додатку не залежить від платформи (таких як Android, Windows, Linux) на якій запускається додаток;
- Універсальність: додаток не потрібно встановлювати чи оновлювати та не потребує ліцензії від конкретної платформи на які може запускатись;
- Швидкодія: всі операції по зберіганні даних, обробки інформації виконуються на сервері, тому ресурси пристрою використовуються мінімально;
- Масштабованість: оскільки додаток не виконує важкі по ресурсам операції на стороні клієнта, то збільшення швидкості відгуку додатку при напливі великої кількості користувачів зводиться до горизонтального або вертикального масштабування на стороні веб-сервера;
- Інтерактивність: взаємодія між сервером та клієнтом надає можливість не тільки збирати статистику по користувачу, а також персоналізувати для кожного користувача індивідуальні дані.

Веб-додаток та веб-сайт – це два ідентичні терміни, але варто зауважити, що веб-додаток має більше інтерактивних елементів з якими користувач може взаємодіяти ніж веб-сайт. У табл. 1.1 наведений більш детальний розгляд між двома термінами.

Таблиця 1.1 Основні відмінності веб-додатку і веб-сайту

Параметр	Веб-додаток	Веб-сайт
Основне призначення	Створюється для того, щоб взаємодіяти з користувачем [1].	Має на увазі лише наявність контенту можливого впливу на читача [1].
Взаємодія з користувачем	Користувач може проводити маніпуляції з даними, але з обмеженим доступом [1].	Користувач може читати інформаційний контент, проте не може його ніяк міняти [1].
Аутентифікація	Більшість веб додатків вимагають автентифікації, щоб користувач міг скористатися програмою [1].	Для даних сайтів не потрібна обов'язкова автентифікація. Сайт може лише мати пароль, який відсилає адміністратор ресурсу [1].
Завдання та складність	Веб-програма має багато функцій і закриває безліч проблем [1].	Веб-сайт відображає лише статичну сторінку, на якій зображена текстова інформація [1].

Продовження табл. 1.1

Зміна проекту	Щоб внести будь-які зміни в проект, потрібно робити ревю всього коду і після вписувати те, що ви хочете змінити [1].	Досить просто вносити зміни, лише зробивши пару змін у html коді сторінки [1].

1.2 Підходи до створення веб-додатку

Для створення веб-додатку розробнику в «світі JavaScript» має обрати з великої кількості варіантів, не тільки типів веб-додатку, а також з використанням якої технології буде створений додаток. Це зумовлено тим що JavaScript є однією з найрозповсюджених мов програмування, а також можливістю кожної людини створити та розповсюдити власне рішення якоїсь проблеми чи нове рішення як Node Package Module.

Визначення «веб-додаток» може сприйматись по-різному в залежності від середовища використання веб-технологій. Для прикладу, веб-додаток можна створити як повноцінний «Android», «Desktop», веб-сервер чи консольний застосунок, але головною сферою використання є запуск у браузері.

До типів веб-додатку відносять:

- Single Page Application (SPA): цей тип додатку який запускається тільки на стороні клієнта. Додаток є інтерактивним для користувача та навігація по всьому додатку відбувається без перезавантаження сторінки;
- Multi Page Application (MPA): традиційні багатосторінкові веб-додаток. При переході між сторінками користувач завантажує нову сторінку. Тому обмін даними відбувається повільніше, ніж у SPA.

Зокрема, якщо виникають проблеми з доступом до Інтернету або з хостингом веб-сайту [1];

- Progressive Web Application (PWA) мають близькі можливості, функції та якість користувацького досвіду, подібні до традиційних комп'ютерних та мобільних додатків. Хоча не існує чіткого розрізу між не-PWA та PWA додатками, можна виділити кілька характеристик. Наприклад, PWA повинні мати проксі-шар (Service Worker) та Web App маніфест. У практичному сенсі, браузер виступає в якості віртуальної машини для запуску веб-додатків, подібно до того, як Windows запускає exe-файли, а Android – apk. Service Worker виступає як проксі-шар між серверною та клієнтською частиною. Він розташований у браузері та обробляє всі запити. Таким чином, існує два клієнтські шари – один відповідає за інтерфейс, а інший – за логіку [1]. Такий підхід до створення веб-застосунку надає можливість створювати повноцінні онлайн та офлайн веб-датки які можна запускати на будь-якій платформі.

Популярними веб-технології які використовуються для створення веб-додатку є React, Vue, Angular. Також ці технології можуть використовувати підхід SPA та PWA, об’єднуючи їх для покращення користувацького досвіду. Більш детально про технології:

- React – створив JSX підхід для написання коду (можливість писати у JavaScript в HTML), також модель «Data це View», оновлення даних призводить до Re-Render, та Virtual-DOM для оптимізації перемальовування сторінки браузера. Оскільки це бібліотека що контролює re-render то ця технологія популярна в випадках де важливий продуктивність додатку;
- Angular – це також єдиний з двох інших технологій що використовує стандарт Web-components як основну технологію, вимагає знання TypeScript та використовує реактивну бібліотеку як ядро реактивності.

Технологія добре підходить для випадків коли потрібно писати надійний код, коли важливо що було не складно будувати додаток відповідно до визначеної архітектури;

- Vue – переймає найкращі практики двох інших технологій. Цю технологію обирають через простоту у вивчені, швидкість розробки, менший розмір бандлу.

Окрім зазначених технологій є такі, що використовують у собі одну із вже зазначених технологій для відмолювання сторінок, використовують підхід Server Side Rendering (SSR) для CEO оптимізації сторінки, пришвидшення швидкість завантаження сторінки та надають можливість будувати додаток із використанням всіх трьох типів веб-додатків.

У випадках коли веб-додаток не потребує використання жодної із вище зазначених технологій, а достатньо використання чистих рішення на JavaScript, використовують банлдери такі як: Vite, Webpack, Rollup, Parcel; Gulp. Також варто зазначити, що більшість із зазначених технологій використовують у собі бандлер для покращення досвіду розробника. Бандлери призначенні для розробки коли кінцевий веб-додаток SPA, PWA, або MPA складається з маленьких частинок які треба зібрати в один або кілька статичних файлів. На Рис. 1.1 зображено принцип роботи бандлера.

Рис. 1.1. Зображення роботи бандлера [2]

1.3 Огляд готових рішень для систем обміну контентом

Готовими рішення для систем обміну контентом є CMS (Content Management System) – це система керування вмістом сайту, або їх ще називають CMS-движок [3]. Такі системи пропонують використовувати готові шаблони для сайту, також надають готові функціональні рішення для керування вмістом сайту та його дизайном, різноманітні плагіни які реалізують певний функціонал для веб-сайту. Приклади CMS є: WordPress, Joomla! OpenCart, Shopify, Fix.

Відповідно до даних агентства W3Techs, 63.3% сайтів в інтернеті створенні за допомогою CMS рішень, решта 31.7% веб-сайтів не використовують жодну з систем управління контентом [4].

Системи керуванням вмісту поділяються на чотири види:

- CMS з відкритим кодом мають важливу особливість – вони дозволяють будь-кому модифікувати їх двигун. Це сприяє постійному збагаченню таких CMS новими додатками, темами і швидшому виявленню та усуненню вразливостей. І саме це є однією з основних причин, чому WordPress став настільки популярним двигуном. До популярних CMS з відкритим кодом відносяться WordPress, OpenCart, Joomla!, Drupal, Magento, PrestaShop [3].
- Коробкові CMS відрізняються тим, що в них використовується закритий код, що означає, що лише офіційні розробники можуть вносити зміни в двигун. Це не означає, що такі CMS менш безпечні або мають обмежений функціонал. Однак, зазвичай кількість доступних тем та доповнень у них менша. До популярних коробкових CMS відносяться Tilda, Wix, SitePro, Shopify, Squarespace [3].
- Самописні CMS розробляються на замовлення для конкретного проекту. Їх функціонал може бути менш розгалуженим, ніж у коробкових CMS або CMS з відкритим кодом, проте він максимально відповідає потребам проекту і не містить зайвих інструментів. Однак,

якщо необхідно розширити функціонал або виправити вразливості, потрібно звертатися до розробника двигуна або шукати спеціаліста, який детально розбирається у коді [3].

- Фреймворки: при такій реалізації потрібно створювати власноруч CMS рішення з нуля, тобто корім функціональності притаманній такій системі також потрібно реалізувати панель керування. Такий підхід дозволяє реалізувати будь-які функціональні можливості, а також підвищити продуктивність веб-сайту створеного з використанням фреймворку. Популярні рішення у такому підході є: Laravel, Ruby on Rails, Django.

CMS – це веб-додаток, що надає можливість створювати та обслуговувати сайти. Однією з вагомих переваг таких рішень є те що для початку створення веб-сайту не обов'язково знати програмування.

У всіх CMS рішеннях, робота над створенням та редагуванням вмісту сайту відбувається через особистий кабінет, а саме з панелі керування. Панель керування CMS системи WordPress зображено на Рис. 1.2.

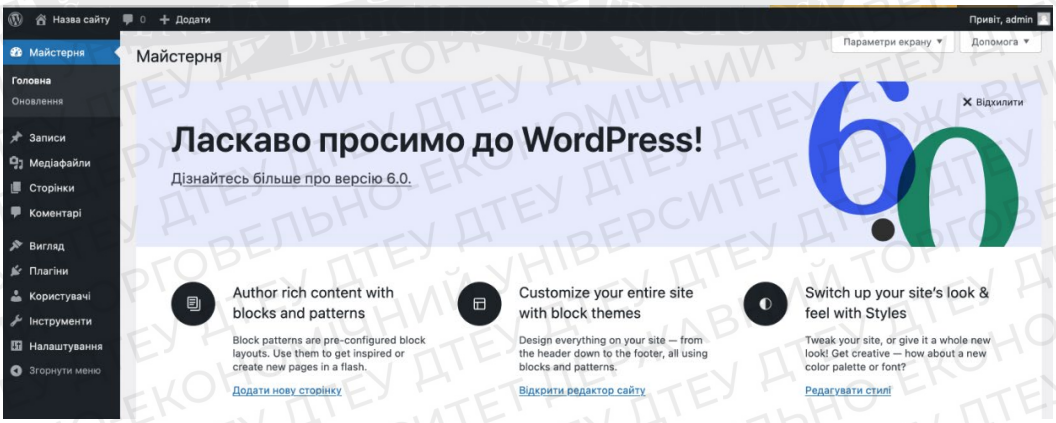

Рис. 1.2. Панель керування в WordPress [3]

Для дизайну веб-сайту CMS система може надавати можливість завантажити готовий шаблон та налаштувати конкретні елементи дизайну такі як: посилання, кольори, шрифти, логотип, зображення, тощо. Варто зазначити,

що можливість модифікації дизайну залежать від конкретної системи контролю вмістом. На Рис. 1.3 зображено розділ із шаблонами в WordPress.

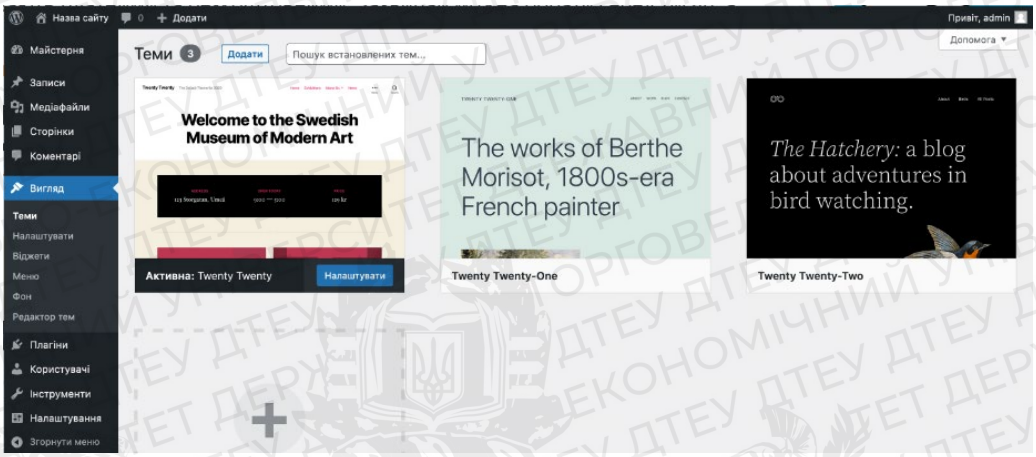

Рис. 1.3. Розділ із шаблонами в WordPress [3]

Окремі CMS рішення надають можливість самостійно створювати сторінки веб-сайту з використанням вбудованого конструктора сторінок. Конструктор сторінок – візуальний редактор подібний до Canva, PowerPoint де можна створювати власний дизайн використовувати заздалегідь визначені елементи інтерфейсу та стилізуючи їх. На Рис. 1.4 зображений конструктор сторінки в WordPress.

Рис. 1.4. Конструктор сторінки в WordPress [3]

Якщо веб-сайту потрібні потрібно додати функціональність як: підписка на розсилку, контактну форму, онлайн-чат, метод доставки чи оплати, CMS надають можливість встановити розширення, або плагіни які реалізують потрібний функціонал. На Рис. 1.5 зображений інтерфейс додавання плагіну в WordPress.

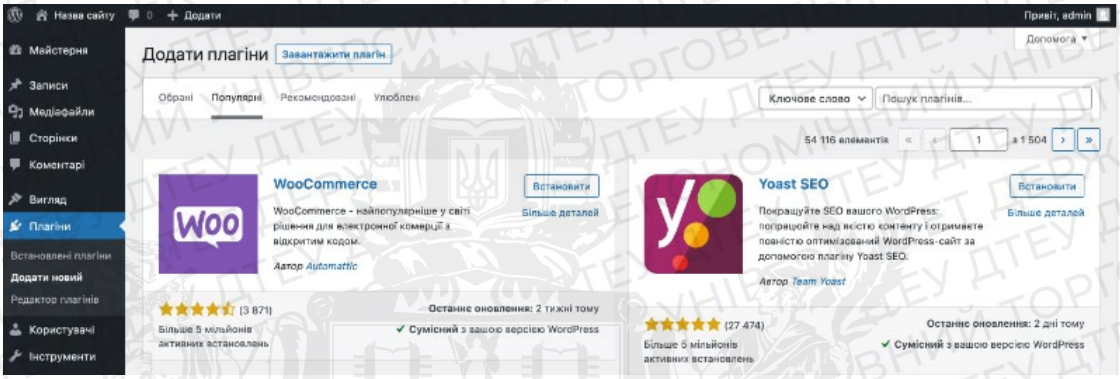

Рис. 1.5. Інтерфейс додавання плагіну в WordPress [3]

Висновки до розділу 1

Підсумовуючи все вище сказане веб-додаток – це інтерактивний веб-сайт, який не залежний від платформи на якій запускається. Також існує велике різноманіття способів створення веб-додатку, але варто зазначити, що велика кількість технологій для розробки веб-додатку є як перевагою так мінусом. Оскільки можна обрати технології які розв’язують певну проблему найпродуктивнішим шляхом, але обрання відповідних технологій вимагає відповідно як досвіду в сфері розробки так і досвіду з технологією для врахування всіх можливих плюсів та мінусів обраного варіанту.

CMS системи вважають універсальним рішенням, що підходять для більшої реалізації веб-додатків, але на практиці виходить, що є певні критерії коли розробка з використанням CMS системи не є доцільною. Причини через які не варто застосувати CMS рішення:

- Обмежений по функціональності вибір готових рішень не підходить для сайтів із унікальними рішеннями [5];
- Перехід на нові версії або перенесення проекту на інший двигун може призвести до спотворень структури [5];
- При підключенні плагінів потрібні додаткові ресурси на сервері [5];
- Спеціалізовані системи, розроблені з урахуванням вимог проекту, є ідеальними для вирішення поставлених завдань. Таке програмне забезпечення відрізняється високою стабільністю та має меншу вразливість порівняно зі загальнопоширеними CMS рішеннями [5].

РОЗДІЛ 2.
ПРОЄКТУВАННЯ СИСТЕМИ ОБМІНУ КОНТЕНТОМ

2.1 Вимоги до системи обміну контентом

Визначення вимог є невід'ємним етапом у процесі реалізації від ідеї до продукту. Вимоги зазначають, що потрібно розробляти, як це має працювати, а також допомагають визначити критерії успішності для розробленої системи.

Визначення точних та конкретних вимог забезпечує всім сторонам залучених до створення системи мати чітке розуміння того, що вони розробляють.

Бізнес-вимоги визначають стратегічний шлях проекту. Вони включають вимоги до проекту високого рівня, які описують загальну бізнес-цілі з точки зору компанії. Ціллю може бути збільшення прибутку або захоплення більшої частки ринку [6]. Бізнес-вимогами системи обміном контентом є надання користувачу можливість із ознайомленням з історією персонажа та мультимедійним контентом (фото, відео) з ігор та інших розважальних сервісів, інтерактивна взаємодія з користувачем для його зацікавлення в контенту, що пропонується та популяризація нової ніші в українському сегменті шляхом створення об'єднаної спільноти.

Вимоги до рішення описують конкретні характеристики, якими повинен володіти продукт, щоб відповідати потребам зацікавлених сторін і самого бізнесу. Вони діляться на дві великі групи [7].

- Функціональні вимоги описують можливості, які має мати рішення з точки зору поведінки та інформації, якими рішення буде керувати [8];
- Нефункціональні вимоги не пов'язані безпосередньо з поведінкою функціональності рішення, а скоріше описують умови, за яких рішення повинно залишатися ефективним, або якості, які має мати рішення [8].

Функціональні вимоги для системи обміном контентом мають реалізувати наступні можливості:

- Створення облікового запису користувача шляхом реєстрації;
- Авторизація користувача;
- Переглядати мультимедійний контент як фото галерея, відео та текстової інформації на сторінці «персонажа» для авторизованого користувача;
- Видаляти мультимедійний контент (фото) з категорії «улюблених фото»;
- Додавати мультимедійний контент (фото) до категорії «улюблених фото»;
- Створити особистий список «улюблених персонажів»;
- Обрати фото профілю облікового запису із запропонованого списку;
- Редагувати персональну інформацію облікового запису;
- Фільтрація фото контенту за критеріями;
- Відправити електроне повідомлення розробнику.

Нефункціональні вимоги для системи обміном контентом:

- Вимоги до застосування:
  1) Система має бути веб-орієнтованою;
  2) Дизайн веб-додатку має бути побудований за підходом «Desktop first».
- Вимоги до продуктивності: система має підтримувати кілька тисяч одночасних запитів на веб-сервер та сторонній сервіс для зберігання медіа контенту та інформації користувача.
- Вимоги до реалізації:
  1) Мови програмування для реалізації вед-додатку та веб-сервера мають бути наступні JavaScript, TypeScript;
  2) Архітектурний шаблон для проектування та розробки веб-сервера має бути MVC (Model View Controller);

3) Архітектурний шаблон для проектування та розробки веб-додатку має бути MVVM (Model View ViewModel);
4) Проєкт має бути побудований з використанням монолітної архітектури;
5) Веб-сервер має використовувати підхід SSR (Server Side Rendering) для динамічних сторінок;
6) Для веб-додатку, а саме статичних сторінок має бути використаний бандлер;
7) Використовувати сторонній сервіс для зберігання медіа контенту, інформації користувача та іншої потрібної інформації для системи. Сервіс має надавати SDK (Software Development Kit) рішення для роботи із базою даних та для роботи із хмарним сховищем для медіа контенту;
8) Редагувати, додавати та змінювати медіа контент в системі має бути можливо тільки через використання панелі керування стороннього сервісу для збереження всіх медіа ресурсів;
9) Мінімальна версія браузерів в яких має працювати веб-додаток є: Chrome 49, Firefox 18, Safari 10, Edge 12.

- Вимоги до надійності:
  1) Веб-сервер має витримувати одночасне навантаження не менше 2000 користувачів;
  2) Веб-додаток має надавати користувачеві тільки заздалегідь визначений функціонал.
- Вимоги до інтерфейсу:
  1) Веб-додаток має взаємодіяти з веб-сервером через REST API (Representational State Transfer Application Programming Interface);
  2) Веб-сервер має взаємодіяти із стороннім сервісом для зберігання медіа ресурсів та інформації користувача через SDK.
- Вимоги до безпеки:

1) Маршрути та API, що вимагають аутентифікацію користувача мають бути захищені на стороні веб-сервера;
2) Веб-сервер має використовувати CSRF (Cross-Site Request Forgery) захист від підробки міжсайтових запитів;
3) Авторизовані користувачі можуть змінювати дані тільки ті що безпосередньо пов’язані з ними.

2.2 Моделювання системи обміну контентом

Представлення розроблюваної системи у вигляді візуальних модель є важливим етапом у розробці програмного забезпечення, тому що надає змогу візуалізувати поведінку й структуру системи чи процесу. Уніфікована мова моделювання (UML) допомагає визначити потенційні помилки в структурі програм, поведінці системи й інших бізнес-процесах [9].

Для розробки важливо чіткої визначити границі між зовнішнім середовищем і цільовою системою. До складу системи будуть входити всі елементи, що забезпечують функціонування системи. Все що не буде входити до системи, буде відноситься до навколишнього середовища. Відповідно до цільової системи буде відноситись веб-додаток та веб-сервер. Сервіси що надаватимуть можливість зберігання всіх медіа ресурсів та інформації користувача, відправлення електронного повідомлення буде відноситись до зовнішнього середовища.

Границя системи обміном контентом показана на Рис. 2.1.

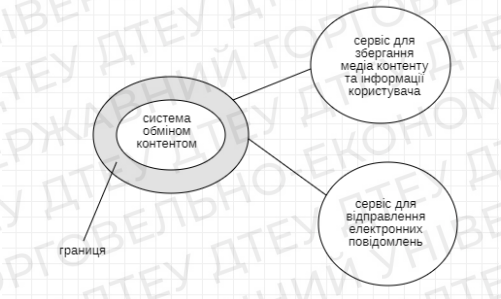

Рис. 2.1. Система обміном контентом, її границі і зовнішнє середовище

Діаграма прецедентів (сценарій використання) – це діаграма на якій зображують які є відносити між актором та системою. Але не вказує на порядок виконання кроків. Діаграма є представленням функціональних вимог зі сторони користувача та може бути описана текстом або з використанням діаграм. [10].

На Рис. 2.2 наведена діаграма прецедентів для системи обміном контентом.

Опис діаграма прецедентів потрібен для конкретизації функцій та задач які повинна виконувати система обміном контентом. Виконання цього етапу дозволить зменшити ризики помилок та невдач при розробці програмного забезпечення.

У табл. 2.1 представленні актори і прецеденти відповідно до даних з діаграма прецедентів (див. Рис. 2.2).

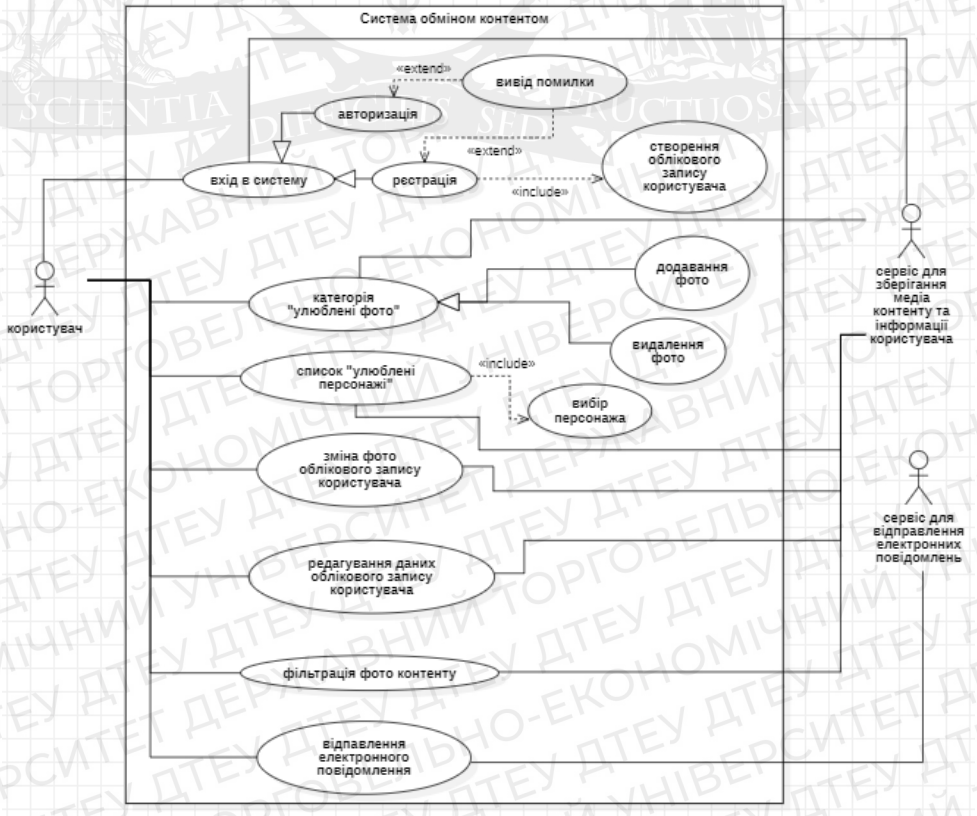

Рис. 2.2. Діаграма прецедентів

Таблиця 2.1 Елементи моделі варіантів використання

Список акторів	Список прецедентів
Користувач Сервіс для зберігання медіа контенту та інформації користувача Сервіс для відправлення електронних повідомлень	Вхід в систему Авторизація Реєстрація Створення облікового запису користувача Вивід помилки (при вході в систему) Категорія «улюблені фото» Додавання фото до категорії «улюблені фото» Додавання фото з категорії «улюблені фото» Список «улюблених персонажів» Зміна фото облікового запису користувача Редагування даних облікового запису користувача Фільтрація фото контенту Відправлення електронного повідомлення

Зазвичай прецеденти виражаються на різних рівнях деталізації та в різних форматах, зважаючи на етап проєктування є:

- Прецедент високого рівня – використовують на початку формулюванні вимог, оскільки опис процесу є дуже стислим;
- Прецедент середнього рівня – зазначають короткий опис;
- Розгорнутий прецедент – більш розгорнутий опис процесу в порівнянні з прецедентом високого рівня.

У табл. 2.2 наведений детальний опис високорівневих прецедентів із табл. 2.1.

Таблиця 2.2 Опис високорівневих прецедентів

Прецедент	Актор	Опис
Вхід в систему	Користувач Сервіс для зберігання медіа контенту та інформації користувача	Користувач виконує потрібні дії для прецеденту «авторизація» або «реєстрація» для входу в систему.
Категорія «улюблені фото»	Користувач Сервіс для зберігання медіа контенту та інформації користувача	Користувач може додавати або видаляти фото зі списку категорії «улюблені фото». Система зберігає зміни в БД.
Зміна фото обліково запису	Користувач Сервіс для зберігання медіа контенту та інформації користувача	Користувач змінює фото облікового запису та підтверджує дію. Система зберігає зміни в БД.
Список «улюблених персонажів»	Користувач Сервіс для зберігання медіа контенту та інформації користувача	Користувач обирає улюблених персонажів та підтверджує дію збереження. Система зберігає зміни в БД.
Редагування даних облікового запису користувача	Користувач	Користувач заповнює відповідні поля як на

Продовження табл. 2.2

	Сервіс для зберігання медіа контенту та інформації користувача	прецеденті «реєстрація» та підтверджує дію. Система зберігає зміни в БД.
Фільтрація фото контенту	Користувач Сервіс для зберігання медіа контенту та інформації користувача	Користувач обирає із списку запропонованих категорій та підтверджує дію. Система повертає список фото контенту із врахуванням опцій користувача.
Відправлення електронного повідомлення	Користувач Сервіс для відправлення електронних повідомлень	Користувач заповнює поля з поштою та текстом повідомлення та підтверджує дію. Система відправляє електронне повідомлення розробнику.

Для пріоритезації розробки важливої функціональності, прецеденти потрібно ранжувати відповідно за категоріями на основні, другорядні та додаткові.

Класифікація прецедентів відповідно за категоріями:

- Основні прецеденти:

1. Вхід в систему;
2. Категорія «улюблені фото»;
3. Список «улюблені персонажі»;
4. Фільтрація фото контенту.

- Другорядні прецеденти:
  1. Редагування даних облікового запису;
  2. Зміна фото облікового запису;
- Додаткові прецеденти:
  1. Відправлення електронного повідомлення.

Діаграма послідовності (Sequence Diagram) демонструє взаємодію об'єктів і відображає хронологічну послідовність передачі та отримання повідомлень між ними [10].

Діаграма послідовностей для головних прецедентів системи обміну контентом представлено на Рис 2.3-2.6.

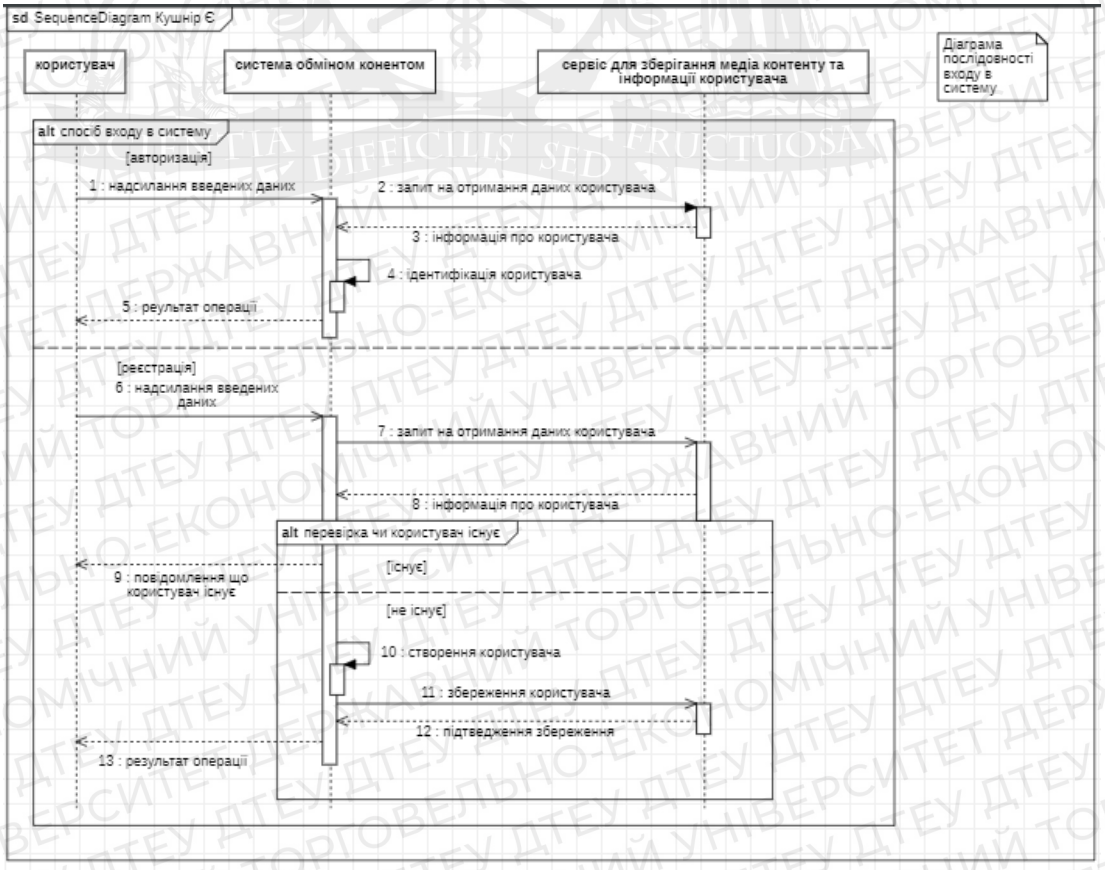

Рис. 2.3. Діаграма послідовності (прецедент вхід в систему)

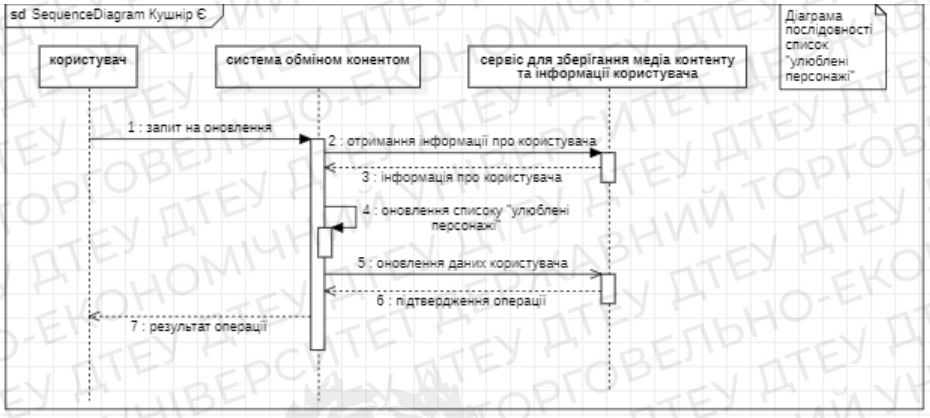

Рис. 2.4. Діаграма послідовності (прецедент список «улюблені персонажі»)

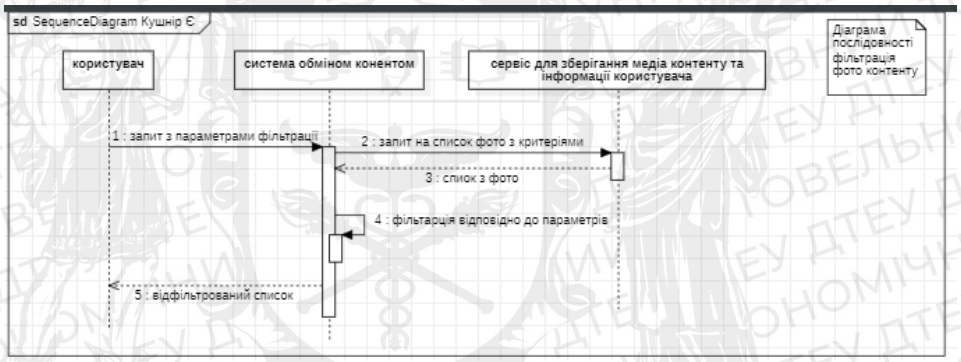

Рис. 2.5. Діаграма послідовності (прецедент фільтрація фото контенту)

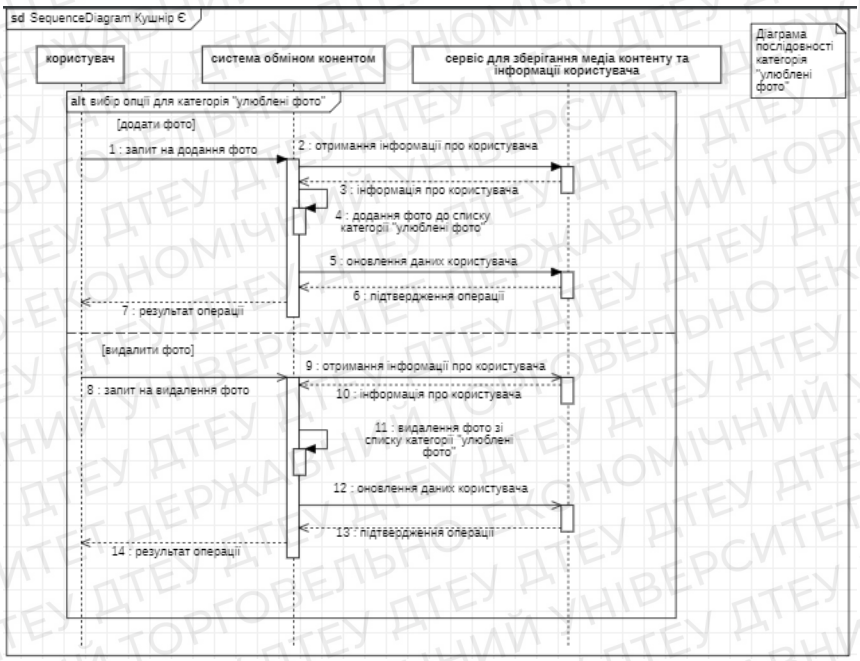

Рис. 2.6. Діаграма послідовності (прецедент категорія «улюблені фото»)

Діаграма класів включає набір статичних, декларативних елементів моделі. Цей тип діаграми надає детальне та повне уявлення про зв'язки в програмному коді, функціональність та інформацію про окремі класи та їх взаємозв'язки [11]. На Рис. 2.7 наведена діаграма класів для системи обміном контентом.

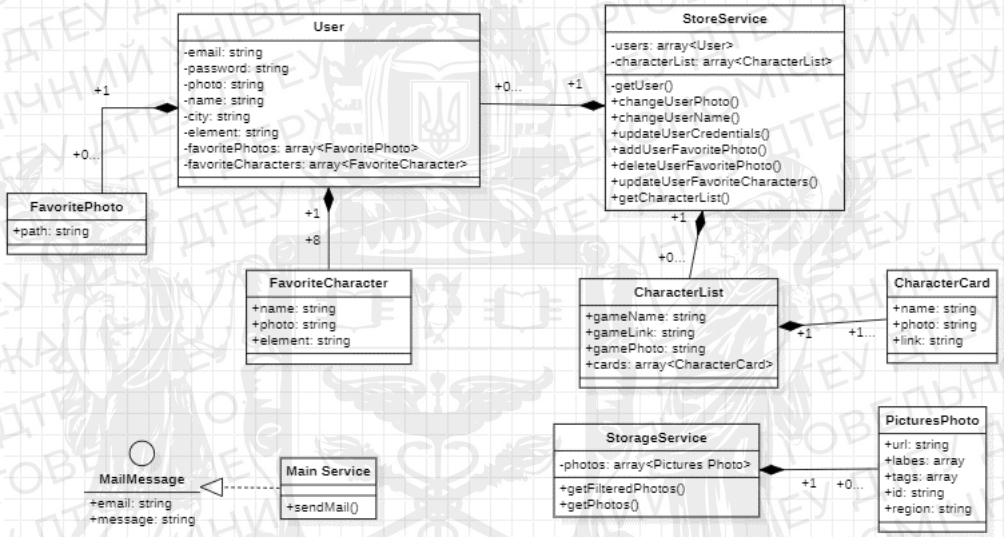

Рис. 2.7. Діаграма класів системи обміном контентом

2.3 Розробка проєктного рішення для системи обміну контентом

Проєктне рішення – це обґрунтована пропозиція щодо досягнення певного бажаного результату, яка визначає доцільність його реалізації у вигляді проєкту. Проєктне рішення є складовою проєкту і спів-відносяться як частина і ціле [12].

Проєктне рішення для системи обміном контентом буде представляти набір поступових етапів, які детермінують кількість роботи яку потрібно виконати для успішної реалізації проєкту.

Етапи розробки системи обміном контентом:

- Етап «дослідження»:

– Дослідження ринку та конкурентів: порівняння можливостей конкурентів та цільової системи (див. табл. 2.3).

Таблиця 2.3 Дослідження особливостей конкурентів

Особливість / Конкуренти	Fandom Website	«Вісник Паймон» Website	Honey Impact Website	Цільова система
Різноманітність контенту	так	-	-	так
Інтерактивна взаємодія з користувачем	так	так	так	так
Фото галерея	-	так	-	так
Створення профілю користувача	так	-	так	так
Наявність категорії «улюблені фото»	-	-	-	так
Направленість на україномовний сегмент	-	так	-	так

– Аналіз цільової аудиторії: відповідно до використання способу «5W»:

1. «What»: можливість користувачам ознайомитись із мультимедійним контентом та текстовою інформацією по світі

ігор та інших розважальних сервісів та інтерактивна взаємодія з користувачем;

2. «Who»: люди будь-якого віку, що люблять проводити час граючи в ігри;
3. «Why»: можливість швидко ознайомитись із певним контентом;
4. «Where»: пошукові системи;
5. «When»: будь-якої миті, в більшості, вечірній час після 19:00.

- Етап «Розробка повноцінного дизайну сайту»:
  - Проєктування інтерфейсу та архітектури сайту;
  - Розробка розділів і сторінок для макету сайту.
- Етап «Верстка сайту за дизайном»:
  - Верстка всіх сторінок за дизайном;
  - Робота з анімацією елементів на сторінках.
- Етап «Реалізація серверної частини»:
  - Створення маршрутизації для сайту;
  - Створення публічних та приватних REST API;
  - Створення API документації;
  - Використання сторонніх сервісів для отримання інформації користувача та даних про мультимедійний контент.
- Етап «Тестування веб-додатку та веб-серверу»:
  - Написання Unit тестів для сервісів веб-серверу;
  - Написання End to End тестів для веб-застосунку.
- Етап «Підтримка»:
  - Публікація початкового коду в репозиторії одного з веб-сервісів для сховищ систем контролю версій (Git);
  - Створення дошки Kanban із завданнями для цільової системи.

Висновки до розділу 2

В ході проєктування системи обміну контентом було сформульовані функціональні та не функціональні вимого, а також зазначені бізнес-вимоги. На основі вимог система обміном контентом була змодельована з використанням наступних діаграм: варіантів використання, послідовності та класів. Після цього було сформовано проєктне рішення для системи обміном контентом, що визначає необхідні етапи для успішної розробки системи.

РОЗДІЛ 3.
СТВОРЕННЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ОБМІНУ КОНТЕНТОМ

3.1 Обгрунтування стеку технологій для розробки веб-додатку

У веб-розробці вибір технологій з допомогою яких буде розроблятись програмна реалізація – це завжди важливий етап оскільки в світі JavaScript є велика кількість рішень, які розв'язують конкретні вимоги надзвичайно ефективним шляхом.

Стек технологій – це набір технологій, які складаються разом для створення будь-якого додатку. Відомий як стек технологічної інфраструктури або стек рішень, стек технологій став необхідним для створення простих в обслуговуванні, масштабованих веб-додатків [13]. Технологічний стек складається з двох однаково важливих елементів: фронтенду (клієнтської частини) та бекенду (серверної частини) [14].

Від правильного вибору стеку технологій буде залежати також успішність бізнес проєкту. Тому для вибору стеку технологій варто орієнтуватись на наступні ключові пункти:

- Підтримка робочих процесів: впровадження нових програмних рішень може займати значний час. Команді необхідно адаптуватися, а для підвищення ефективності потрібен час. Однак, правильний вибір набору технологій від самого початку може допомогти уникнути зайвих змін у майбутньому [15];
- Економія бюджету: перебудова технологічного стеку є витратною справою. Це може включати переробку програм, переогляд та зміну процесів і навіть прийом на роботу нових співробітників зі спеціальними навичками [15];

- Підтримка інновацій та масштабування: обмеження бюджету та кадрового складу команди може уповільнювати розвиток продукту і якщо не можна допустити, щоб технологічний стек став головною причиною провадження нових можливостей та ідей, то потрібно завжди бути певним що технологічний стек заохочує інновації, а не обмежувати їх. [15].

Відповідно до визначених вимог (див. пункт 2.1) та етапів проєктного рішення (див. пункт 2.3) для успішної реалізації веб-додатку потрібно:

- Створити макет сайту;
- Реалізувати веб-сервер із використанням сторонніх сервісів та створення API документації;
- Відповідно до макету реалізувати веб-додаток.

А отже, перш ніж перейти до реалізації веб-додатку потрібно визначити які технології варто використовувати на кожному з вище зазначених етапів.

Макет сторінки веб-сайту (або прототип) – це візуальне відображення майбутнього сайту, його дизайну. Макет включає елементи, такі як колірна гама, шрифти, зображення кнопок, розташування елементів та інше. Якість виконання макету впливає на графічне оформлення сторінок, сприйняття представленого контенту та загальне враження від інтернет-ресурсу. [16].

Створення прототипу надає наступні переваги:

- Заздалегідь зрозуміти який вигляд матиме новий сайт [16];
- Своєчасно виявити помилки та виправити те, що не подобається [16];
- Запропонувати замовнику декілька альтернативних варіантів є важливим, оскільки зазвичай клієнт не має чіткого уявлення про те, що саме він бажає [16];
- Привернути увагу потенційних інвесторів та показати свій продукт [16].

Для створення макету сайту можуть використовувати один із наступних інструментів:

- Figma – це потужний професійний інструмент для дизайнерів, який призначений для розробки багатьох графічних елементів та компонентів. Створити прототип сайту на Figma можна настільки точно, що розробникам залишиться перенести елементи у фреймворк та задати їм параметри [17];
- Axure RP – це потужна програма для створення якісних та складних прототипів. Вона призначена для UX дизайнерів, яким необхідно створювати інтерактивні програми та сайти [17];
- Draftium – містить велику кількість готових шаблонів. Прототип збирається за принципом блокової системи, достатньо перетягнути потрібний елемент, задати йому параметри і буде на своєму місці. Не потрібно з нуля малювати всі функції. Але навіть якщо це знадобиться, це можна зробити за допомогою якісних інструментів [17];
- InVision – це професійна програма для роботи в команді. Воно дає доступ відразу кільком дизайнерам і розробникам, завдяки чому кожен може вносити свої зміни за наявності прав. Програма дозволяє створювати інтерактивні прототипи сайтів із повноцінним функціоналом та відображенням [17];
- Adobe XD – один з найпопулярніших інструментів серед UI/UX-дизайнерів. Дана програма функціонує в якості універсальної платформи для створення дизайну сайтів, мобільних додатків, голосових інтерфейсів і т. д. Adobe XD дозволяє командам колективно працювати між платформами в режимі реального часу, отримуючи цінні коментарі та відгуки безпосередньо від клієнтів [18].

Отже, для створення макету сайту буде використано інструмент Figma, це зумовлено не великою кривою входу для початку використання, достатньою кількістю функціональності та відсутності обмежень щодо створення та редагування проєктів.

Back-end (або серверна частина) – це програмна частина, що відповідає за усе що відбувається за лаштунками веб-сайту, веб-додатку чи програми, також відома як серверна сторона. Завдання бекенду полягає у з'єднані бази даних та фронтенду, який використовує відповідні дані для відображення у зручному форматі для користувача. А також для зворотного з'єднання вихідних даних з фронтенду з базою даних через сервер [19].

До складу серверної частини належать наступні елементи:

- Сервер – це комп’ютерна система, яка відповідає за доступність. Наприклад, щоразу, коли клієнт заходить на веб-сайт, його комп’ютери надсилають запит на сервер для завантаження веб-сайту. Отже, основне завдання сервера – слухати і відповідати на вхідні запити [20];
- Фреймворки – це фрагменти багаторазового коду, які забезпечують певну функціональність. Вони пропонують стандартизований спосіб створення програмних рішень, заощаджуючи при цьому значну кількість часу. Фреймворки слугують керівництвом для структурування коду, логіки та інших аспектів внутрішньої архітектури [20];
- Бази даних використовуються для ефективного зберігання, організації та управління великими обсягами даних. Сьогодні існує дві найпоширеніші моделі баз даних: реляційна та нереляційна. Реляційні бази даних використовують набір таблиць, які пов'язані між собою. Нереляційні бази даних можуть зберігати різні типи даних, які не пов'язані між собою [20];
- Логіка – описує послідовність операцій, які необхідно виконати для виконання певного завдання. В архітектурі бекенда логіка створюється для роботи майже виключно на серверах, інтерпретуючи вхідні дані та створюючи вихідні [20];
- Інтерфейс прикладного програмування (API) – це міст між клієнтською та серверною сторонами. Він працює так: клієнтська

сторона надсилає запит на сервер, який спочатку перевіряється API. Якщо запит дійсний, API передає його на сервер. Останній створює і надсилає відповідь, яка знову перевіряється API, а потім передається клієнту [20].

Відповідно до вимог можна сформувати наступний список фреймворків, що можуть вирішити поставленні задачі:

- Spring MVC – це фреймворк Java, який використовується для розробки веб-додатків. Він побудований на основі патерну Model-View-Controller (MVC) і володіє всіма основними функціями весняного фреймворку, такими як Dependency Injection, Inversion of Control. Архітектурний дизайн Spring MVC можна використовувати для розробки гнучких веб-додатків. Він в основному розділяє різні аспекти програми, такі як вхідна логіка, логіка інтерфейсу користувача та бізнес-логіка [21];
- Express – це JavaScript web-фреймворк, розроблений для використання в середовищі Node.js. Цей фреймворк відомий своєю швидкістю, гнучкістю і простотою, і вважається стандартом для розробки веб-додатків в середовищі Node.js. Express також має багато додаткових плагінів, які полегшують процес розробки і дозволяють легко вирішувати незвичайні задачі [22];
- Laravel – це кросплатформенний PHP-фреймворк для створення веб-додатків. Це серверна платформа, яка керує даними за допомогою шаблону проектування Model-View-Controller (MVC), що розділяє внутрішню архітектуру програми на логічні частини. Laravel містить набір бібліотек коду з попередньо запрограмованими цінними функціональними можливостями, такими як інструмент аутентифікації, інструмент вкорінення або HTML-шаблони, які дозволяють користувачам швидше створювати додатки. [23];

- Django – це фреймворк з відкритим вихідним кодом для бекенд додатків, заснований на Python – одній з найпопулярніших мов веб-розробки. Його основні цілі – простота, гнучкість, надійність та масштабованість [24].

Для розробки веб-серверу буде використано Express фреймворк. Перевагами цієї технології є єдина мова програмування для веб-додатку та веб-серверу, також простота використання, велика кількість розширень та пакетів для вирішення різних задач.

Із вибором технології для веб-серверу потрібно обрати сервіс, що буде надавати SDK рішення для взаємодії з БД та хмарним сховищем медіа даних та сервіс для відправлення електронних повідомлень.

Серед сервісів, що надають SDK рішення відповідно до вимог можна виділити:

- Google Firebase – це хмарна платформа Google для мобільної та веб-розробки. Це допомагає розробникам створювати, покращувати та масштабувати мобільні програми. Платформа пропонує інструменти, які можуть значно спростити робочі процеси: набір для навчання, хостинг, база даних у реальному часі тощо. Firebase добре підходить для створення MVP, оскільки зменшує час і зусилля, необхідні розробникам. Firebase дозволяє розробникам зосередитися на якості продукту замість зовнішніх проблем, які неминуче виникають під час розробки програми. Гнучкість платформи дозволяє розробляти програми для всіх мобільних операційних систем [25];
- AWS Amplify – це повнофункціональна платформа, розроблена для допомоги веб та мобільним розробникам у створенні повностекових та масштабованих додатків, що працюють під управлінням AWS. Платформа постачається з безліччю інструментів і сервісів, які дозволяють користувачам налаштовувати бекенд, підключати додатки, миттєво розгортати статичні веб-додатки та безперешкодно керувати

контентом поза консоллю AWS. Він також може містити бібліотеки коду, готові до використання компоненти та вбудований інтерфейс командного рядка [26];

- Microsoft Azure – це хмарна платформа, яка об'єднує рішення для інфраструктури як послуги (IaaS) – включаючи сервери, сховища даних, мережі та операційні системи – і набір інструментів та сервісів, які спрощують розробку та розгортання хмарних додатків (PaaS), хмарна платформа для покращення ефективності організації, управління, контролю виконання робочих процесів для середніх та малих підприємств. Це оптимальний варіант для розгортання віртуальної інфраструктури з використанням необхідних додатків. Можливість оперативного вирішення поточних бізнес-завдань [27, 28].

Для веб-серверу буде використано Firebase SDK рішення, це зумовлено тим, що вище згадані сервіси надають велику кількість функціональності, але мають різну криву входу для початку використання. Firebase в порівнянні з платформами AWS та Microsoft не вимагає додаткових знань у самій платформі та її інфраструктурі, що буде використовуватись.

Серед сервісів для відправлення електронних повідомлень можна виділити:

- Mailchimp – це платформа для автоматизації маркетингу, розроблена для компаній, які використовують електронну пошту для виходу на свої цільові ринки. Це універсальний інструмент, за допомогою якого можна керувати списками розсилки, створювати власні шаблони листів, а також підтримувати та автоматизувати всі ваші маркетингові кампанії [29];
- GetResponse – це інструмент для email-маркетингу та онлайн-кампаній. Використовуючи Getresponse, можна створювати потенційних клієнтів за допомогою форм реєстрації, цільових

сторінок і вебінарів. Ви можете надсилати розсилки, автоматизовані електронні листи та створювати рекламу на Facebook [30];

- Sendinblue – це платформа для автоматизації цифрового маркетингу. Вона допомагає розвивати ваш бізнес, дозволяючи створювати автоматизовані маркетингові воронки, які перетворюють відвідувачів на підписників, а підписників на клієнтів. Sendinblue використовується для створення автоматизованих маркетингових кампаній та дозволяє надсилати клієнтам автоматизовані електронні листи, SMS та повідомлення у WhatsApp [31].

Веб-сервер буде використовувати сервіс Sendinblue для відправлення електронних повідомлень, це зумовлено легкістю у використані сервісу та найбільшою кількістю повідомлень наданих у безплатному тарифі.

У плані проєктного рішення зазначена вимога для створення API документації для веб-серверу. Для вирішення цієї задачі можна використати одну із наступних технологій для документування API:

- Postman – це інструмент для перевірки та тестування API, що дозволяє створювати, надсилати та перевіряти HTTP-запити та аналізувати їх відповіді [32];
- Swagger – це інструмент для документування та тестування API, який автоматично створює документацію API на основі опису його структури у форматі YAML або JSON файлу [32];
- SoapUI – це інструмент для тестування веб-сервісів, що дозволяє створювати та виконувати тести на основі SOAP-протоколу [32];
- Fiddler – це інструмент для аналізу та налагодження HTTP-трафіку між веб-браузером і веб-сервером. З точки зору тестування API, Fiddler дозволяє перехоплювати, аналізувати та змінювати HTTP-запити та відповіді, що передаються між клієнтом та сервером [32];
- JMeter – це інструмент для тестування продуктивності та функціональності програмного забезпечення, який може бути

використаний для тестування API. З точки зору тестування API, JMeter дозволяє створювати запити до API, аналізувати відповіді та оцінювати продуктивність та функціональність API [32].

Для документування API буде використано Swagger, оскільки з усіх інших технологій Swagger має більше можливостей для документування API.

Front-end (або клієнтська частина) – це та частина програми або веб-додатку, яка відповідає за відображення та взаємодію з користувачем через його веб-браузер. Це включає в себе створення та розміщення елементів інтерфейсу, таких як кнопки, текстові поля, зображення та інші елементи, а також обробку взаємодії користувача, якість візуального вигляду та забезпечення зручності використання. [19].

Для розробки веб-додатку можна обрати один із двох варіантів, а саме готове рішення у вигляді фреймворку або використати бандлер із власною конфігурацією проєкту. У випадку із фреймворком можна обрати одну із вже згаданих веб-технологій в пункті 1.2, а саме React, Angular та Vue, або одну із наступних:

- Solid – це декларативна бібліотека JavaScript, призначена для створення користувацьких інтерфейсів. На відміну від інших фреймворків, вона не використовує віртуальний DOM. Замість цього він компілює свої шаблони в реальні вузли DOM і оновлює їх за допомогою вбудованої реактивності. Це означає, що повторно виконується лише код, який залежить від зміни стану [33];
- Svelte – це потужний інструмент, який дозволяє швидко розробляти веб-додатки. Він також фокусується на створенні динамічних та цікавих користувацьких інтерфейсів. Однак, на відміну від інших фреймворків, Svelte не використовує віртуальний DOM або бібліотеки часу виконання. Натомість він компілює ваші компоненти у невеликий та ефективний JavaScript-код, який оновлює реальний DOM напряму. Це робить додатки Svelte швидшими, меншими та простішими в обслуговуванні [33];

- Qwik – це DOM-орієнтований JavaScript-фреймворк, який має на меті забезпечити найшвидший "час до інтерактиву", зосереджуючись на відновлюваності для рендерингу HTML на стороні сервера та оптимізованому лінивому завантаженні коду [34].

У випадку із використанням бандлеру, для реалізації веб-додатку можна використати наступні технології:

- Webpack – це потужний пакувальник модулів для JavaScript-додатків. Він пакує всі модулі вашого додатку в один або декілька пакетів (часто лише в один) і подає їх браузеру. Однак Webpack – це більше, ніж просто пакувальник модулів. За допомогою завантажувачів і плагінів він може трансформувати, мінімізувати і оптимізувати всі типи файлів перед тим, як відправити їх браузеру у вигляді одного пакета. Він приймає різні ресурси, такі як JavaScript, CSS, шрифти, зображення та HTML, а потім перетворює їх у формат, зручний для використання браузером [35];
- Gulp – це система потокової збірки, яка допомагає автоматизувати кілька завдань. До завдань які може виконувати Gulp відносять: мінімізація CSS або JS коду, компіляція SASS або LESS файлів, перетворення SVG іконок у шрифти, перегляд файлів на предмет змін та інші дії. Іншими словами Gulp використовується для виконання типових дій і завдань, які в інших випадках є важливими і майже завжди виконуються, але повторне виконання одних і тих же дій забирає багато часу [36];
- Parcel – це пакет для створення веб-додатків з нульовою конфігурацією. Parcel підтримує багато мов і типів файлів з коробки, від веб-технологій, таких як HTML, CSS і JavaScript, до ресурсів, таких як зображення, шрифти і відео. Якщо тип файлу не включено за замовчуванням, Parcel автоматично встановить всі необхідні плагіни та залежності для користувача [37].

Для створення веб-додатку буде використовуватись бандлер Webpack, це зумовлено можливістю повного контролю над процесом конфігурації збірки та легкістю у розширенні при потребі додати веб-фреймворк або іншу технологію.

Відповідно до вимоги для веб-серверу в використанні підходу SSR та технологічного стеку веб-додатку і веб-серверу, для створення динамічних та статичних HTML сторінок буде використовуватись Handlebars. Ця технологія дозволить зменшити кількість повторюваного коду можливістю розділення на окремі частинки шаблону та їх перевикористанням, а також технологія наддасть єдине рішення для клієнтської та серверної частини.

Для стилізації сторінок буде використовуватись SCSS, це мова сценаріїв препроцесора, яка є надмножиною CSS. Вона надає додаткові можливості та функції, які недоступні у звичайному CSS. Синтаксис SCSS дуже схожий на CSS, але дозволяє використовувати змінні, вкладеність, міксини та інші конструкції програмування [38].

3.2 Програмна реалізація системи обміну контентом

Відповідно до пунктів успішної реалізації веб-додатку (див. пункт 3.1), першим етапом реалізації системи обміном контентом є створення макету в Figma. Відповідно до вимог та технологічного стеку було створено макет веб-додатку, на Рис. 3.1-3.3 зображено кілька сторінок макету веб-додатку. При розробці макету сайту розроблено наступні сторінки:

- Головна сторінка – містить коротку інформацію про сервіс. На сторінці наведено мінімальний текст розміром до 100 слів, чорним кольором. Всі навігаційні елементи та зображення, що на сторінці є інтуїтивно простими для користувача;
- Сторінка «Тайват через фото» – реалізує вимогу «фільтрація фото контенту за критеріями». На сторінці з правої сторони розміщено

список фото з тематичним контентом, з лівої сторони – функціонал для безпосередньої фільтрації;

- Сторінка авторизації – містить форму із полями для введення пошти та паролю, а також посилання на сторінку реєстрації;
- Сторінка реєстрації – мітить форму із полями для ведення пошти, паролю, ім’я, вибір ігрового елементу (вода, вогонь, земля, повітря, та інші) та ігрового міста із запропонованого списку;
- Сторінка профілю користувача – на цій сторінці розміщено список фото, що відносяться до категорії «улюблених персонажів» користувача та можливість редагувати список видаляючи певні фото. Також на сторінці є можливість редагувати особисту інформацію користувача, оновлювати список «улюблених персонажів»;
- Сторінка відправлення повідомлення – ця сторінка реалізує вимогу по відправлені електронного повідомлення розробнику, на ній розміщено форма із полем для пошти та тесту повідомлення;
- Сторінка з списком персонажів – містить список всіх персонажів та посилання на детальну сторінку з персонажем;
- Сторінка з персонажем – містить детальну інформацію про персонажа у вигляді текстової інформації, відео та фото контенту, а також має функціональність по додаванню фото до категорії «улюблені фото»;
- Сторінка про сайт – містить короткий текст від розробника системи обміном контентом, не більше 50 слів.

Посилання на ввесь макет для веб-додатку додається: https://www.figma.com/file/8gxkL94XyHiuedslxtSG9y/Keqing-site?type=design&node-id=0%3A1&t=tiGObCvH8Wjlwi2p-1.

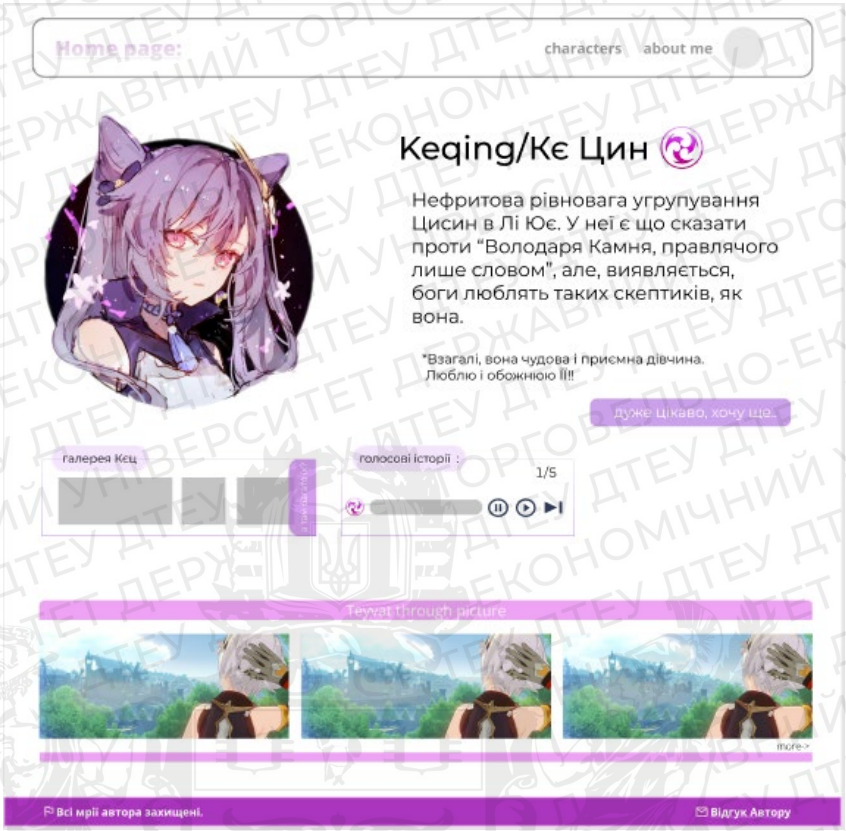

Рис. 3.1. Макет головної сторінки

Рис. 3.2. Макет сторінки авторизації

Рис. 3.3. Макет сторінки, що реалізує вимогу «Фільтрація фото контенту за критеріями»

На другому етапі реалізується безпосередньо веб-сервер із використанням сторонніх сервісів та створення API документації.

Серед вимог до веб-серверу також зазначено використання архітектурного шаблону MVC та CSRF захисту.

MVC (Model View Controller) – це патерн проектування програмного забезпечення для організації коду програми у три взаємопов'язані частини (рис. 3.4) [39]:

- Модель (Model) – це логіка взаємодії з базою даних [39];
- Представлення (View) – це користувацький інтерфейс, з яким взаємодіє користувач [39];
- Контролер (Controller) – це є посередником між представленням і моделлю [39].

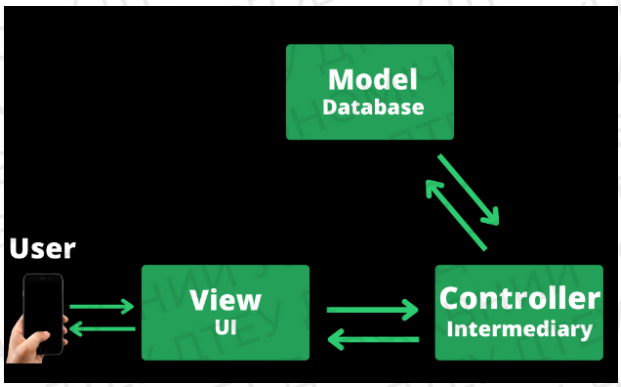

Рис. 3.4. Схема роботи шаблону MVC [39]

На рис. 3.5 зображено архітектуру веб-серверу відповідно до шаблону MVC.

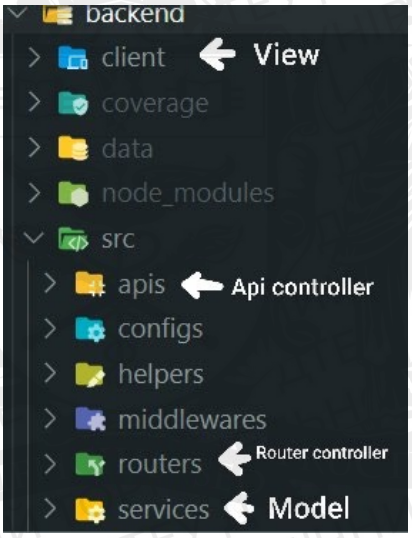

Рис. 3.5. Архітектура веб-серверу

CSRF (Cross-site request forgery) – це міжсайтові атаки підробки запитів використовуються зловмисно для надсилання запитів від автентифікованого користувача до веб-програми. Жертва не може бачити відповіді на підроблені запити, що робить це особливо складним методом атаки, який працює за лаштунками, щоб викликати збої в роботі. Атаки CSRF зосереджені на змінах стану, а не на початковій крадіжці даних [40]. Для реалізації CSRF захисту використано NPM пакет «csurf». На рис. 3.6 зображено ініціалізацію CSRF захисту в веб-сервері.

```
const csurf = require('csurf')
...
const csrufSecure = csurf({ cookie: true })
const csrufSecureAPI = csurf({
  cookie: true,
  ignoreMethods: ['HEAD', 'OPTIONS'],
})
...
app.use(csrufSecure)
...
app.all('*', (req, res, next) => {
  res.cookie('XSRF-TOKEN', req.csrfToken())
  next()
})
app.use(site_router)
if (process.env.NODE_ENV === 'test') {
  app.use('/api/v1', api_v1)
} else {
  app.use('/api/v1', csrufSecureAPI, api_v1)
}
```

Рис. 3.6. CSRF захист у веб-сервері

Відповідно до визначеного стеку технологій для реалізації:

- Веб-серверу використано фреймворк Express;
- Роботи із базою даних та хмарним сховищем для медіа контенту використано Firebase SDK;
- Надсилання електронних повідомлень використано Sendinblue;
- SSR підходу використано Handlebars;
- API документації використано NPM пакет «swagger-ui-express».

На рис. 3.7-3.9 наведено деталі імплементації реалізації веб-серверу відповідно до зазначених пунктів.

```
const { getFirestore } = require('firebase-admin/firestore')
...
const db = getFirestore()

class UserData {
  async #get_user_by_id(uid) {
    const userRef = db.collection('users').doc(uid)

    return {
      user: (await userRef.get()).data(),
      userRef,
    }
  }

  async get_user(uid) {
    const { user } = await this.#get_user_by_id(uid)

    return user
  }

  async update_user(data, uid) {
    const { user, userRef } = await this.#get_user_by_id(uid)

    await userRef.set({
      ...user,
      ...data,
    })
  }
```

Рис. 3.7. Використання Firebase SDK для роботи з базою даних

```
const Sib = require('sib-api-v3-sdk')

const send_mail = async ({ message, email }) => {
  const client = Sib.ApiClient.instance
  const apiKey = client.authentications['api-key']
  apiKey.apiKey = process.env.SENDBLUE_API_KEY

  const tranEmailApi = new Sib.TransactionalEmailsApi()
  const sender = {
    email: process.env.SENDBLUE_EMAIL,
    name: 'Keqing-site 😊',
  }
  const receivers = [
    {
      email: process.env.SENDBLUE_EMAIL,
    },
  ]

  await tranEmailApi.sendTransacEmail({
    sender,
    to: receivers,
    subject: 'New message from keqing-site',
    textContent: `
        <h3>It's message from keqing-site!</h3>
        <p>Email sended from: {{params.uEmail}}</p>
        <p>Message context: {{params.uMessage}}</p>
      `,
    params: {
      uMessage: message,
      uEmail: email,
    },
  })
}
```

Рис. 3.8. Використання сервісу Sendinblue

```
...
const {
  get_character_images,
} = require('../../services/firebase/storage.service')
const {
  store_data,
  user_data,
} = require('../../services/firebase/store.service')

const character = async (req, res) => {
  try {
    const { name } = req.params
    const { uid } = req.query

    const user = await user_data.get_user(uid)
    if (!user) throw Error('user not defined')

    const data = await Promise.all([
      store_data.get_data('characters', name),
      get_character_images(name),
    ])

    res.render(path.resolve(__dirname, '../../../client/views/character.hbs'), {
      layout: 'layout-character',
      name,
      images: data[1],
      ...data[0],
    })
  } catch (error) {
    res.redirect('/registration')
  }
```

Рис. 3.9. Використання SSR підходу з використанням Handlebars

Для опису структури API використано YAML файл (рис. 3.10). Також Swagger відповідно до конфігураційного файлу автоматично генерує Swagger-UI, що представляє собою повноцінну інтерактивну веб-сторінку з якою користувач може взаємодіяти (рис. 3.11). Веб-сервер надає доступ до сторінки API документації за наступним маршрутом «[URL]/api-docs».

```
paths:
  /images:
    get:
      tags:
        - Images
      summary: Get images for home page
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Images'

        '500':
          description: Error
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
  /images/login:
    get:
      tags:
        - Images
      summary: Get login for character page
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
```

Рис. 3.10. YAML файл для API документації

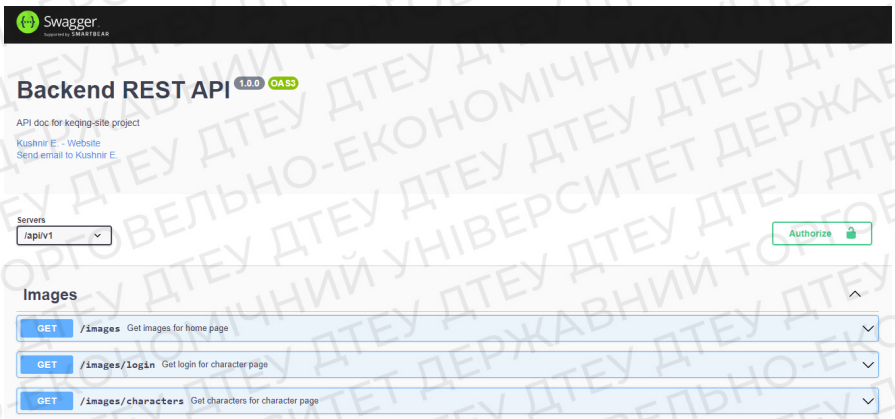

Рис. 3.11. Згенерована веб-сторінка для API документації

У додатку А наведено деталі імплементації для налаштування веб-серверу на Express, а саме: виконання різноманітних налаштувань, створення екземпляру серверу, створення маршрутів та підключення потрібних модулів.

На третьому етапі відбувається реалізація веб-додатку. Відповідно до визначеного стеку технологій на цьому етапів:

- Використано бандлер Webpack із власною конфігурацією;
- Зверстано веб-сторінки відповідно до макету.

Webpack – це інструмент, який використовується для збирання та пакування модулів у сучасних JavaScript-додатках. При роботі з додатком, webpack створює внутрішній граф залежностей з однієї або кількох початкових точок, а потім об'єднує всі необхідні модулі проекту у один або кілька пакетів [41].

Для створення статичних веб-сторінок потрібно сформувати файл конфігурації. Відповідно до конфігурації webpack буде формувати бандл для кожної сторінки у вигляді: HTML файлу, CSS стилів, JavaScript файлів та інших потрібних медіа ресурсів. Щоб сформувати файл конфігурації webpack потрібно розібратись у основних концепціях бандлеру:

- Entry (точка входу) – вказує, з якого модуля webpack повинен почати будувати свій внутрішній граф залежностей. Webpack з'ясує, від яких інших модулів і бібліотек залежить ця точка входу (прямо чи опосередковано) [41];

- Output (точка виходу) – вказує webpack, куди помістити створені ним файли і як їх називати [41];
- Loaders (завантажувачі) – з коробки webpack розуміє лише JavaScript та JSON файли. Завантажувачі дозволяють webpack обробляти інші типи файлів і перетворювати їх на коректні модулі, які можуть бути використані вашим додатком і додані до графа залежностей [41];
- Plugins (плагіни) – у той час як завантажувачі використовуються для перетворення певних типів модулів, плагіни можна використовувати для виконання ширшого кола завдань, таких як оптимізація бандлів, ін'єкція змінних середовища [41];
- Mode (режим) – встановивши для параметра mode значення «development», «production» або «none», можна увімкнути вбудовані оптимізації webpack, які відповідають кожному середовищу [41].

Відповідно до концепції webpack сформовано конфігураційний файл (див. дод. В).

Також однією із вимог для створення веб-додатку є використання архітектурного шаблону MVVM.

Model-View-ViewModel (MVVM) – це патерн проектування програмного забезпечення, який структурований таким чином, щоб розділити логіку програми та елементи керування користувацьким інтерфейсом (рис. 3.12) [42]. Елементами шаблону є:

- Модель (Model) – містить логіку програми, яку отримує модель вигляду при отриманні вхідних даних від користувача через View [42];
- Представлення (View) – це сукупність видимих елементів, які також отримують вхідні дані від користувача. Сюди входять користувацькі інтерфейси (UI), анімація і текс [42];
- Модель вигляду (ViewModel) – тут розміщуються елементи керування для взаємодії з представленням, а зв'язування використовується для

з'єднання елементів інтерфейсу користувача у представлені з елементами керування у ViewModel [42].

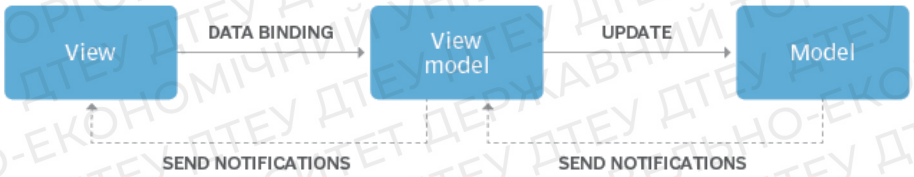

Рис. 3.12. Схема роботи шаблону MVVM [42]

Архітектурний шаблон MVVM зручно використовувати для розробки програмного забезпечення коли потрібно реалізувати «прив'язку даних». Тобто встановити зв'язок між інтерфейсом додатку та даними, що він відображає.

Для зручності імплементації шаблону MVVM для веб-додатку використано патерн проєктування Observer (спостерігач) – це поведінковий патерн проектування, який дозволяє визначити механізм підписки для сповіщення декількох об'єктів про будь-які події, що відбуваються з об'єктом, за яким вони спостерігають [43]. У додатку Б зображено деталі імплементації патерну спостерігач. Також для його реалізації використано Proxy object (об'єкти-проксі) – обгортає інший об'єкт і перехоплює операції, такі як читання/запис властивостей та інші, за бажанням обробляючи їх самостійно, або прозоро дозволяючи об'єкту обробляти їх [44].

На рис. 3.13 зображено архітектуру веб-додатку з використанням шаблону MVVM та патерну спостерігача для головної сторінки.

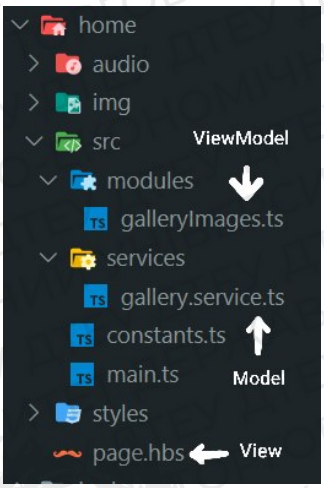

Рис. 3.13. Архітектура веб-додатку

Відповідно після створення конфігураційного файлу для бандлера зверстано сторінки для веб-додатку (Рис. 3.14-3.16).

Рис. 3.14. Головна сторінка веб-додатку

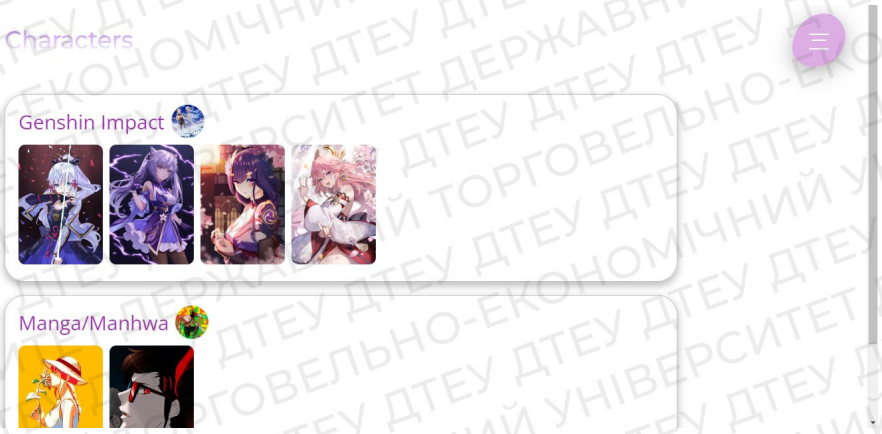

Рис. 3.15. Сторінка із списком персонажів

Рис. 3.16. Сторінка із персонажем

3.3 Тестування системи обміну контентом

Відповідно до етапу тестування в пункті 2.3, щоб протестувати систему обміном контентом потрібно написати Unit тести для сервісів веб-серверу та написати End to End тести для веб-додатку.

Unit тестування (модульне тестування) – це процес, в якому кожна незалежна функціональність додатку перевіряється окремо, в контрольованому середовищі, що штучно створюється [45]. Перевагами такого виду тестування є:

- Легкість перевірки працездатності розглянутого компоненту [45];
- Розділення реалізації від інтерфейсу є необхідним, особливо з урахуванням модульного тестування, оскільки це допомагає зменшити залежності в системі [45];
- Документація модульного тестування може слугувати як «живий документ», що надає приклади використання для кожного тестового класу. Модульне тестування допомагає краще розуміти роль кожного класу у контексті всієї програмної системи [45];
- У методології «розробка через тестування», яка активно застосовується у екстремальному програмуванні, модульне

тестування є одним з основних інструментів, що дозволяє розробляти продукт відповідно до вимог до даного модуля [45].

Для написання модульних тестів буде використовуватись Jest – це фреймворк для "чудового тестування JavaScript". Він побудований на основі Jasmine і є популярним фреймворком для тестування, спеціально розробленим для виконання переважно модульного тестування [46].

На Рис. 3.17 показано деталі імплементації модульного тесту сервісу для роботи з БД. Щоб запустити процес виконання тестів локально потрібно виконати наступну команду в терміналі корінної папки «npm run "be:test"», на Рис. 3.18 показано результат виконання всіх модульних тестів.

```
it('#get_user', async () => {
  const res = await user_data.get_user(mockUserData.uid)

  expect(res).toEqual(mockUserData)
  expect(mockStoreCollection).toBeCalledWith('users')
})

it('#update_user', async () => {
  const newData = { newData: 'test' }

  await user_data.update_user(newData, mockUserData.uid)

  expect(mockStoreSet).toBeCalledWith({ ...mockUserData, ...newData })
})

it('#update_user_stand', async () => {
  const newData = { stand: 'test' }

  await user_data.update_user_stand(newData, mockUserData.uid)

  expect(mockStoreSet).toBeCalledWith({ ...mockUserData, stand: newData })
})

it('#update_user_favorite_new_item', async () => {···
```

Рис. 3.17. Unit тест сервісу для роботи з БД

```
> keqing_site_backend@1.0.0 test
> jest

 PASS  src/services/firebase/auth.service.test.js (7.233 s)
 PASS  src/services/firebase/storage.service.test.js (7.243 s)
 PASS  src/services/firebase/store.service.test.js (7.39 s)
 PASS  src/services/local_data.service.test.js
 PASS  src/services/mail.service.test.js

Test Suites: 5 passed, 5 total
Tests:       15 passed, 15 total
Snapshots:   0 total
Time:        14.167 s
Ran all test suites.
```

Рис. 3.18. Результат виконання Unit тестів

End to End (або E2E, наскрізне тестування) – це метод тестування програмного забезпечення, який перевіряє функціональність і продуктивність всього програмного додатку від початку до кінця, імітуючи реальні сценарії користувачів і реплікуючи реальні дані. Його мета – виявити помилки, які виникають при інтеграції всіх компонентів, гарантуючи, що додаток забезпечує очікуваний результат як єдине ціле [47]. До переваг наскрізного тестування відносять:

- Управління якістю на різних рівнях додатків: сучасні додатки побудовані на складній архітектурі, що складається з декількох рівнів із взаємопов'язаними робочими процесами. Ці шари можуть добре працювати окремо, але конфліктувати один з одним, коли вони з'єднані. E2E-тестування може перевірити взаємодію між цими окремими рівнями та компонентами [47];
- Backend тестування: E2E-тестування спочатку перевіряє бекенд-рівні, особливо базу даних додатку, яка передає критично важливу інформацію іншим рівням для роботи додатку [47];
- Забезпечте стабільну якість додатків у різних середовищах: наскрізне тестування перевіряє фронтенд, гарантуючи, що додаток працює належним чином у широкому діапазоні браузерів, пристроїв і платформ [47];
- Тестування сторонніх додатків: у додаток інтегровані зовнішні системи для виконання вузько специфічних завдань. Наскрізне тестування забезпечує сумісність між зовнішніми та внутрішніми системами, а також обмін даними між ними [47].

Для написання та виконання E2E тестів використано Cypress – це фреймворк для наскрізного інтеграційного та модульного тестування, спрямований на уніфікацію тестування веб-додатків незалежно від мови програмування, що використовується. Він працює на всіх платформах і в усіх

браузерах. Cypress також надає інтегроване середовище розробки, яке завантажуєте у окремому браузері [48].

На Рис. 3.19 зображено деталі імплементації E2E тестів для сторінки персонажа. Щоб запустити процес виконання тестів локально потрібно виконати наступну команду в терміналі корінної папки «npm run "fe:test"», на Рис. 3.20 показано результат E2E тесту для сторінки персонажа.

```
describe('Test character page', () => {
  beforeEach(() => {
    cy.visit('/login?continuePath=/characters/ayaka')
    cy.fixture('user-credentials.secret').then((user) => {
      cy.get('#email').type(user.email, { force: true })
      cy.get('#password').type(user.password, { force: true })
      cy.get('form button[type="submit"]').click()
    })
  })

  beforeEach(() => {
    cy.get('.k-header-controls-item.k-images-btn_open').as('openImages')
    cy.get('.k-images').as('listImages')
    cy.get('.k-header-controls-item.k-video-btn_open').as('openVideo')
    cy.get('.k-video').as('video')
  })

  it('#open character images should work', () => {
    cy.get('@openImages').click()
    cy.get('@listImages').should('be.visible')
  })

  it('#open character video should work', () => {
    cy.get('@openVideo').click()
    cy.get('@video').should('be.visible')
  })
})
```

Рис. 3.19. E2E тест для сторінки персонажа

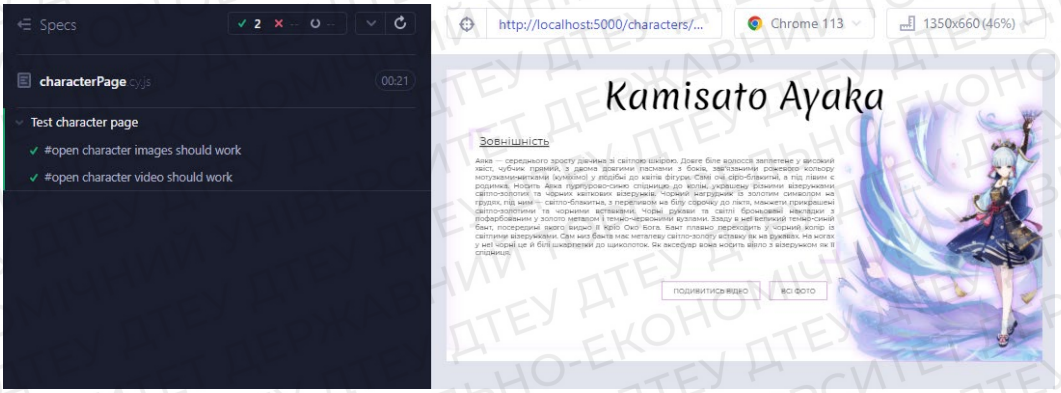

Рис. 3.20. Результат виконання тесту для сторінки персонажа

Висновки до розділу 3

У процесі реалізації системи обміном контентом було здійснено ретельний аналіз різних технологічних стеків. Враховуючи всі вимоги до реалізації було обрано певний стек технологій. Вибір стеку технологій включав фреймворк для написання веб-серверу, SDK рішення для взаємодії з БД та хмарним сховищем медіа ресурсів, сервіс для розробки макету веб-додатку, вибір способу реалізації веб-додатку між фреймворком та використанням бандлеру, вибір сервісу для відправлення електронних повідомлень та технології для документування API.

На основі обраного стеку було розроблено програмну реалізацію системи обміном контентом використовуючи найкращі практики для структурування коду. Реалізація системи включає всі функціональні вимоги, крім того було враховано і не функціональні вимоги для створення системи обміном контентом як CSRF захист веб-серверу.

Щоб забезпечити програмну якість коду було створено Unit тести для сервісів веб-серверу та документування API, це забезпечить стабільність та надійність окремих частин системи. А також розроблено E2E тести для веб-додатку, щоб перевірити повний цикл взаємодії користувача із системою.

ВИСНОВКИ

У випускній кваліфікаційній роботі представлено результати теоретичного дослідження та практичної реалізації, що полягають у розробці WEB-орієнтованої системи обміном контентом. До результатів теоретичного дослідження відноситься дослідження способів побудови веб-додатків, формування функціональних, не функціональних вимог та проєктного рішення для системи, а також створення UML-діаграм. Результати теоретичного дослідження стали основою для практичної реалізації WEB-орієнтованої системи обмін контентом. В результаті практичної реалізації системи обміном контентом отримані наступні **висновки:**

1. Здійснено реалізацію макету для веб-додатку відповідно до функціональних вимог, це дозволило виявити не точності у вимогах, заздалегідь отримати основні особливості веб-додатку та скорегувати процес розробки.

2. Здійснено програмну реалізацію клієнтської частини системи обміном контентом відповідно до створеного макету, визначених вимог та обраного стеку технологій у вигляді багатосторінкового веб-додатку.

3. Здійснено програмну реалізацію веб-серверу для системи обміном контентом у відповідності до вимог зі використанням сторонніх сервісів, обраного стеку технологій та забезпечення захисту веб-серверу від підробки міжсайтових запитів.

4. Здійснено тестування системи обміном контентом з написанням Unit тестів для веб-серверу та End to End тестів для веб-додатку. Тестування забезпечує правильність роботи системи та безпосередньо допомагає виявити помилки у разі їх появи при зміні певної часини системи.

5. Для розширення можливості підтримки проєкту початковий код системи обміном контентом розміщено на сервісі для зберігання репозиторіїв GitHub, посилання додається: https://github.com/zenia369/keqing-site. Також створено дошку Kanban для розподілення завдань для розробки та створення ного функціоналу на сервісі Trello, посилання додається:

https://trello.com/invite/b/g9Rau0Ln/ATTI86e11be8abca39d919f930557754935eBC6B8957/keqing-site. Ці дії допоможуть забезпечити можливість участі інших розробників у розвитку системи обміном контентом.

WEB-орієнтована система для обміном контентом розміщена в Інтерне та доступна за наступним посланням: https://keqing-site.onrender.com/.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке Веб додаток? [Електронний ресурс] // Webcase. – Режим доступу: https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/. – Назва з екрану.
2. Webpack [Електронний ресурс] // Документація Webpack. – Режим доступу: https://webpack.js.org/. – Назва з екрану.
3. Що таке CMS сайту [Електронний ресурс] // Hostiq. – Режим доступу: https://hostiq.ua/wiki/ukr/cms/. – Назва з екрану.
4. Usage statistics of content management systems [Електронний ресурс] // W3techs. – Режим доступу: https://w3techs.com/technologies/overview/content_management. – Назва з екрану.
5. Що таке CMS і які види CMS для сайтів бувають [Електронний ресурс] // Cityhost. – Режим доступу: https://cityhost.ua/uk/blog/chto-takoe-cms-i-kakie-vidy-cms-dlya-saytov-byvayut.html. – Назва з екрану.
6. How to Write a Business Requirements Document: Guidelines, Templates, and Useful Tips [Електронний ресурс] // Altexsoft. – Режим доступу: https://www.altexsoft.com/blog/business-requirements-document/. – Назва з екрану.
7. Functional and Nonfunctional Requirements: Specification and Types [Електронний ресурс] // Altexsoft. – Режим доступу: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/. – Назва з екрану.
8. Brennan K. A Guide to the Business Analysis Body of Knowledge (BABOK Guide) 3rd edition, International Institute of Business Analysis, 2015p. – 521c.
9. Простий посібник зі схем UML і моделювання баз даних [Електронний ресурс] // Microsoft, Cor. – Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling. – Назва з

екрану.

10. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти [Електронний ресурс] // Dou. – Режим доступу: https://dou.ua/forums/topic/40575/. – Назва з екрану.

11. UML для бізнес-моделювання: для чого потрібні діаграми процесів [Електронний ресурс] // Evergreens. – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html. – Назва з екрану.

12. Пушкар Т. А. Техніко-економічне обґрунтування проєктних рішень, Економіка та суспільство, (випуск 28), 2021р. – 6с.

13. What is a Tech Stack and How Do They Work? [Електронний ресурс] // Mongodb, Inc. – Режим доступу: https://www.mongodb.com/basics/technology-stack. – Назва з екрану.

14. Top 8 Tech Stacks: Choosing the Right Tech Stack [Електронний ресурс] // Fullscale. – Режим доступу: https://fullscale.io/blog/top-5-tech-stacks/. – Назва з екрану.

15. Технічний стек. Що це таке? [Електронний ресурс] // Azbyka. – Режим доступу: https://azbyka.com.ua/uk/tehnicheskij-stek1/. – Назва з екрану.

16. Що таке макет сайту та навіщо він потрібен [Електронний ресурс] // Patprofi. – Режим доступу: https://patprofi.com/shho-take-maket-sajtu-ta-navishho-vin-potriben/. – Назва з екрану.

17. Прототип сайту – як він допомагає оцінці та розробці [Електронний ресурс] // Wezom. – Режим доступу: https://wezom.com.ua/ua/blog/prototip-sajta-kak-on-pomogaet-ocenke-i-razrabotke. – Назва з екрану.

18. Створення прототипу сайту у 2021: 7 інструментів, які вам допоможуть [Електронний ресурс] // Uaspectr. – Режим доступу: https://uaspectr.com/2022/11/08/instrumenty-prototypuvannya-v-2022-rotsi/. – Назва з екрану.

19. Що таке Frontend і Backend? [Електронний ресурс] // Kr-labs. – Режим доступу: https://kr-labs.com.ua/blog/shho-take-frontend-i-backend/. –

Назва з екрану.

20. A Down-to-Earth Guide to Choosing Backend Technologies for Business Owners [Електронний ресурс] // Visartech. – Режим доступу: https://www.visartech.com/blog/backend-technologies-guide/. – Назва з екрану.

21. Introduction To Spring MVC Flow Diagram & Advantages [Електронний ресурс] // Upgrad. – Режим доступу: https://www.upgrad.com/blog/spring-mvc-flow-diagram/. – Назва з екрану.

22. Створення веб-додатків на EXPRESS.JS [Електронний ресурс] // Brander. – Режим доступу: https://brander.ua/technologies/expressjs. – Назва з екрану.

23. A brief guide through Laravel [Електронний ресурс] // Mdevelopers. – Режим доступу: https://mdevelopers.com/blog/a-brief-guide-through-laravel. – Назва з екрану.

24. Why We Use Django Framework & What Is Django Used For Laravel [Електронний ресурс] // Djangostars. – Режим доступу: https://djangostars.com/blog/why-we-use-django-framework/. – Назва з екрану.

25. What is Firebase [Електронний ресурс] // Flipabit. – Режим доступу: https://flipabit.dev/glossary/firebase/. – Назва з екрану.

26. What is AWS Amplify? [Електронний ресурс] // Back4app. – Режим доступу: https://blog.back4app.com/what-is-aws-amplify/. – Назва з екрану.

27. Microsoft Azure [Електронний ресурс] // Datagroup. – Режим доступу: https://www.datagroup.ua/smb/dlya-viddalenoyi-roboti/microsoft-azure. – Назва з екрану.

28. Платформа Microsoft Azure [Електронний ресурс] // Techexpert. – Режим доступу: https://techexpert.ua/it-products/platforma-microsoft-azure/. – Назва з екрану.

29. So, What Exactly is Mailchimp and Why Do You Need It for Email Marketing [Електронний ресурс] // Cospark. – Режим доступу: https://cospark.com/blog/what-is-mailchimp/. – Назва з екрану.

30. What is GetResponse and why do you need it? [Електронний

ресурс] // Emailaudience. – Режим доступу: https://www.emailaudience.com/what-is-getresponse/. – Назва з екрану.

31. What is Sendinblue [Електронний ресурс] // Websiterating. – Режим доступу: https://www.websiterating.com/email-marketing/what-is-sendinblue/. – Назва з екрану.

32. Тестування API за допомогою Swagger: особливості та переваги [Електронний ресурс] // Ithillel. – Режим доступу: https://blog.ithillel.ua/articles/api-testing-with-swagger. – Назва з екрану.

33. SolidJS vs Svelte: The Ultimate Comparison of Two Innovative Web Frameworks [Електронний ресурс] // Frontendmag. – Режим доступу: https://www.frontendmag.com/insights/solidjs-vs-svelte/. – Назва з екрану.

34. Qwik, a Resumable Javascript Framework [Електронний ресурс] // Infoq. – Режим доступу: https://www.infoq.com/news/2021/09/qwik-javascript-framework/. – Назва з екрану.

35. Webpack: A Gentle Introduction to the Module Bundler Framework [Електронний ресурс] // Auth0. – Режим доступу: https://auth0.com/blog/webpack-a-gentle-introduction/. – Назва з екрану.

36. Gulp.js and Webpack – Which One to Use and When? [Електронний ресурс] // Buddy. – Режим доступу: https://buddy.works/tutorials/gulp-js-and-webpack-which-one-to-use-and-when. – Назва з екрану.

37. What is Parcel.js? [Електронний ресурс] // Trustradius. – Режим доступу: https://www.trustradius.com/products/parcel-js/reviews. – Назва з екрану.

38. What is the difference between CSS and SCSS? [Електронний ресурс] // Geeksforgeeks. – Режим доступу: https://www.geeksforgeeks.org/what-is-the-difference-between-css-and-scss/. – Назва з екрану.

39. MVC in Computer Science – The MVC Model [Електронний ресурс] // Freecodecamp. – Режим доступу: https://www.freecodecamp.org/news/what-does-mvc-mean-in-computer-science/. – Назва з екрану.

40. What Is CSRF And How Do You Prevent It? [Електронний ресурс] // Guardrails. – Режим доступу: https://www.guardrails.io/blog/what-is-csrf-and-how-do-you-prevent-it/. – Назва з екрану.

41. Webpack Concepts [Електронний ресурс] // Документація Webpack. – Режим доступу: https://webpack.js.org/concepts/. – Назва з екрану.

42. Model-View-ViewModel (MVVM) [Електронний ресурс] // Techtarget. – Режим доступу: https://www.techtarget.com/whatis/definition/Model-View-ViewModel. – Назва з екрану.

43. Observer [Електронний ресурс] // Refactoring. – Режим доступу: https://refactoring.guru/design-patterns/observer. – Назва з екрану.

44. Proxy та Reflect [Електронний ресурс] // Документація Javascript. – Режим доступу: https://uk.javascript.info/proxy. – Назва з екрану.

45. Модульне тестування [Електронний ресурс] // Qalight. – Режим доступу: https://qalight.ua/baza-znaniy/modulne-testuvannya/. – Назва з екрану.

46. Jest Tutorial: Complete Guide to Jest Testing [Електронний ресурс] // Lambdatest. – Режим доступу: https://www.lambdatest.com/jest. – Назва з екрану.

47. What is End-to-End Testing? E2E Testing Full Guide Testing [Електронний ресурс] // Katalon. – Режим доступу: https://katalon.com/resources-center/blog/end-to-end-e2e-testing. – Назва з екрану.

48. What Is Cypress Testing and 4 Steps to Get Started Testing [Електронний ресурс] // Snipcart. – Режим доступу: https://snipcart.com/blog/what-is-cypress. – Назва з екрану.

ДОДАТКИ

Додаток А

Програмний код для налаштування веб-серверу на Express

```
const path = require('path')
const { APP_CONFIG } = require('./app_paths')
require('dotenv').config({ path: APP_CONFIG })
const express = require('express')
const csurf = require('csurf')
const cookieParser = require('cookie-parser')
const morgan = require('morgan')
const handlebarsConfig = require('./configs/handlebars')
const firebaseConfig = require('./configs/firebase')
const swaggerConfig = require('./configs/swagger')
const app = express()
const csrufSecure = csurf({ cookie: true })
const csrufSecureAPI = csurf({
  cookie: true,
  ignoreMethods: ['HEAD', 'OPTIONS'],
})
firebaseConfig()
handlebarsConfig(app)
swaggerConfig(app)
const site_router = require('./routers/pages.route')
const error_router = require('./routers/error.route')
const api_v1 = require('./apis')
app.use(express.static(path.join(__dirname, '../client/public')))
app.use(express.json())
app.use(cookieParser())
app.use(csrufSecure)
app.use(morgan('tiny'))
```

```
app.all('*', (req, res, next) => {
 res.cookie('XSRF-TOKEN', req.csrfToken())
 next()
})
app.use(site_router)
if (process.env.NODE_ENV === 'test') {
 app.use('/api/v1', api_v1)
} else {
 app.use('/api/v1', csrufSecureAPI, api_v1)
}
app.use(error_router)
app.use((err, req, res, next) => {
 if (err.code === 'EBADCSRFTOKEN') {
  return res.status(403).send('try go to home page')
 }
 return res
  .status(400)
  .sendFile(
   path.resolve(__dirname, '../client/public/pages/error/index.html')
  )
})
module.exports = app
```

Додаток Б

Програмний код реалізації патерну спостерігач для веб-додатку

```
function createReactiveObject<T extends object>(
  _value: T,
  data: T,
  subsribesMap: SubsribesMapType<T>,
  observerId: string,
  tree: SubscribeWatchersType = []
) {
  const getPath = (key: string) => tree.concat(key).join('.')
  const proxyHelper: ProxyHandler<T> = {
    set(...args) {
      const path: string = getPath(args[1] as string)
      const success = Reflect.set(...args)
      if (success) {
        const isWatched = subsribesMap.has(path)
        try {
          if (isWatched) {
            subsribesMap.get(path)?.forEach((s) => {
              if (typeof s === 'function') {
                requestAnimationFrame(() => s(data))
              } else if (typeof s === 'object') {
                requestAnimationFrame(() => s.complete(data))
              }
            })
          }
          subsribesMap.get(observerId)?.forEach((s) => {
            if (typeof s === 'function') {
              requestAnimationFrame(() => s(data))
            } else if (typeof s === 'object') {
```

```
          requestAnimationFrame(() => s.complete(data))
        }
      })
    } catch (error: any) {
      subsribesMap.forEach((set) => {
        set.forEach((s) => {
          if (typeof s === 'object') {
            requestAnimationFrame(() => s.error(error, data))
          }
        })
      })
    }
  }
  return success
},
get(...args) {
  const value = Reflect.get(...args)
  if (
    value &&
    typeof value === 'object' &&
    ['Array', 'Object'].includes(value.constructor.name)
  ) {
    return createReactiveObject<T>(
      value as T,
      data,
      subsribesMap,
      observerId,
      tree.concat(args[1] as string)
    )
  }
```

```
    return value
   },
  }

  return new Proxy(_value, proxyHelper)
 }

 export function createObserver<T extends object>(_value: T) {
  const data: T = structuredClone(_value)
  const observerId: string = crypto.randomUUID()
  const subsribesMap: SubsribesMapType<T> = new Map()
  const addSubscriberHelper = (
   handler: SubscribeHandlerType<T>,
   watcher = observerId
  ) => {
   const subSet = subsribesMap.get(watcher)
   if (subSet) {
    subSet.add(handler)
   } else {
    subsribesMap.set(watcher,                                    new
Set<SubscribeHandlerType<T>>().add(handler))
   }
  }
  const subject = createReactiveObject(
   data,
   data,
   subsribesMap,
   observerId
  ) as CreateObserverInitialValueType<T>
```

```
  const subscribe = (
    handler: SubscribeHandlerType<T>,
    watchers?: SubscribeWatchersType
  ): (() => void) => {
    if (watchers === undefined || watchers.length === 0) {
      addSubscriberHelper(handler)
    } else if (watchers.length > 0) {
      watchers.forEach((w) => {
        addSubscriberHelper(handler, w as string)
      })
    }
    return () => {
      if (watchers === undefined || watchers.length === 0) {
        subsribesMap.get(observerId)?.delete(handler)
      } else if (watchers.length > 0) {
        watchers.forEach((w) => {
          subsribesMap.get(w as string)?.delete(handler)
        })
      }
    }
  }
  return { subject, subscribe }
}
```

Додаток В

Програмний код для файлу конфігурації webpack

```
const path = require('path')
const MiniCss = require('mini-css-extract-plugin')
const CopyPlugin = require('copy-webpack-plugin')
const TsconfigPathsPlugin = require('tsconfig-paths-webpack-plugin')
const WebpackBar = require('webpackbar')
const generateEntries = require('./generateEntries')
const generateHtmlTemplatePlugin = require('./generateHtmlTemplatePlugin')
const OUTPUT_PATH = path.resolve(__dirname, '../../../backend/client/public')
module.exports = (mode) => {
 const isDev = mode === 'development'
 return {
  mode,
  entry: generateEntries(),
  output: {
   filename: isDev
    ? 'static/js/main.[name].js'
    : 'static/js/main.[name].[hash:6].bundle.js',
   chunkFilename: isDev
    ? 'static/js/chunk.[name].js'
    : 'static/js/chunk.[name].[hash:6].js',
   path: OUTPUT_PATH,
   assetModuleFilename: 'static/assets/[hash][ext][query]',
  },
  devtool: isDev ? 'source-map' : false,
  plugins: [
   new MiniCss({
    filename: ({ chunk }) => {
     if (chunk.filenameTemplate) {
```

```
        return 'static/styles/style.[name].css'
      }
      return isDev
        ? 'static/styles/style.[name].css'
        : 'static/styles/style.[name].[hash:6].min.css'
    },
    chunkFilename: () => {
      return isDev
        ? 'static/styles/style.chunk.[name].css'
        : 'static/styles/style.chunk.[name].[hash:6].min.css'
    },
  }),
  new CopyPlugin({
    patterns: [path.resolve(__dirname, '../../public')],
  }),
  new WebpackBar({
    name: 'Keqing-site_FE',
    color: '#41b883',
    profile: true,
    reporter: {
      changeOverTime: true,
      done: () => console.log('🎉 My Awesome Build compiled successfully!'),
      allDone: () => console.log('🎉 All builds done!'),
    },
  }),
  ...generateHtmlTemplatePlugin(),
],
module: {
  rules: [
```

```
{ test: /\.ts$/, loader: 'ts-loader' },
{
 test: /\.s[ac]ss$/i,
 use: [
  MiniCss.loader,
  {
   loader: 'css-loader',
   options: {
    sourceMap: isDev,
   },
  },
  {
   loader: 'sass-loader',
   options: {
    sourceMap: isDev,
   },
  },
 ],
},
{
 test: /\.html$/i,
 use: 'html-loader',
},
{
 test: /\.(png|jpe?g|gif|svg|jfif|mp3)$/i,
 type: 'asset/resource',
},
{
 test: /\.hbs$/,
 loader: 'handlebars-loader',
```

```
        options: {
          inlineRequires: '/(img|audio)/',
          partialDirs: [path.resolve(__dirname, '../../src/shared/views')],
        },
      },
    ],
  },
  resolve: {
    extensions: ['.js', '.ts'],
    plugins: [
      new TsconfigPathsPlugin({
        configFile: path.resolve(__dirname, '../../tsconfig.json'),
      }),
    ],
    alias: {
      '@': path.resolve(__dirname, '../../'),
      '@Styles': path.resolve(__dirname, '../../src/shared/styles'),
      '@Shared': path.resolve(__dirname, '../../src/shared/js'),
      '@Lib': path.resolve(__dirname, '../../src/lib'),
      '@Util': path.resolve(__dirname, '../../src/shared/utils'),
      '@UI': path.resolve(__dirname, '../../src/lib/ui'),
      '@Public': path.resolve(__dirname, '../../src/shared/icons'),
    },
  },
}
```