

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

«Розробка вебсторінки ігрового порталу»

Виконав студент групи КН-19-2
КОПИЛОВ Владислав Русланович

Керівник роботи:
ЛЕВЧЕНКО Антон Юрійович

Консультант:
КУЗЬМЕНКО Олександр Вікторович

Рецензент:
ПЕТРОСЯН Руслан Валерікович

Житомир – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«05» січня 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: **КОПИЛОВ Владислав Русланович**

Керівник роботи: **ЛЕВЧЕНКО Антон Юрійович**

Тема роботи: **«Розробка вебсторінки ігрового порталу»**,

затверджена наказом закладу вищої освіти від **«05» січня 2023 р., №10/с**

Термін здачі закінченої роботи: **«20» червня 2023 р.**

Вихідні дані роботи: об'єктом дослідження є вебсторінка ігрового порталу на геймерську тематику. Предметом дослідження є веб-система, платформа розміщення публікацій, її побудова на сучасних технологіях розробки.

Консультанти випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Кузьменко О.В.	02.02.2023р.	02.02.2023р.
2	Кузьменко О.В.	03.04.2023р.	03.04.2023р.
3	Кузьменко О.В.	15.05.2023р.	15.05.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел.	2 лютого 2023 – 3 квітня 2023	Виконано
2	Проектування структури системи	3 квітня 2023 – 12 травня 2023	Виконано
3	Написання програмного коду	12 травня 2023 – 10 червня 2023	Виконано
4	Тестування системи	10 червня 2023 – 12 червня 2023	Виконано
5	Оформлення пояснювальної записки	12 червня 2023 – 16 червня 2023	Виконано

Здобувач вищої освіти **Владислав КОПИЛОВ**
Керівник **Антон ЛЕВЧЕНКО**

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «Розробка вебсторінки ігрового порталу» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на **77** сторінках друкованого тексту.

Пояснювальна записка має **3** сторінки додатків. Список використаних джерел містить **21** найменування і займає **2** сторінки. В роботі наведено **53** рисунки. Загальний обсяг роботи – **83** сторінок.

У першому розділі були визначені основні вимоги та критерії до проекту, були оглянуті альтернативні вебсторінки, було обрано стек технологій для розробки.

У другому розділі було проведено проектування вебсторінки, була розроблена серверна частина веб-додатку.

У третьому розділі був розроблений користувацький інтерфейс, було побудовано React застосунок, був описаний порядок роботи з вебсторінкою, були протестовані функціональні можливості вебсайту.

Висновок містить в собі результати виконаної роботи по розробці вебсторінки ігрового порталу.

					КН.КР.Б – 122 – 23 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		В.Р. КОПИЛОВ			Розробка вебсторінки ігрового порталу			Літ.	Арк.	Аркушів	
Керівник		А.Ю. ЛЕВЧЕНКО								3	83
Рецензент		Р.В. ПЕТРОСЯН						Житомирська політехніка, група КН-19-2			
Н. контр.											
Зав. каф.		М.С. ГРАФ									
					Пояснювальна записка						

ABSTRACT

The explanatory note to the qualification work on the topic "Development of a game portal web page" consists of a list of abbreviations, introduction, three chapters, conclusions, a list of references and an additions.

The text part is set out on 77 pages of printed text.

The explanatory note has 3 pages of appendices. The list of references includes 21 titles and occupies 2 pages. The paper contains 53 pictures. The total volume of the work is 83 pages.

In the first chapter, the main requirements and criteria for the project were defined, alternative web pages were reviewed, and a technology stack was selected for development.

In the second section, the web page was designed, the server side of the web application was developed.

In the third section, the user interface was developed, the React application was built, the procedure for working with the web page was described, and the functionality of the website was tested.

The conclusion contains the results of the work done on the development of the game portal web page.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1 Постановка задачі та технічне завдання на розробку вебсайту.....	9
1.2 Аналіз існуючих програмних продуктів за тематикою проекту	10
1.3 Обґрунтування вибору інструментальних засобів розробки	15
Висновки до першого розділу	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ВЕБСАЙТУ	20
2.1 Визначення варіантів використання та алгоритмів роботи системи.....	20
2.2 Структура бази даних, взаємодія компонентів системи.....	26
2.3 Реалізація функціоналу серверної частини вебсайту.....	31
Висновки до другого розділу.....	58
РОЗДІЛ 3. ПОБУДОВА ІНТЕРФЕЙСУ, ПОРЯДОК РОБОТИ З ВЕБСАЙТОМ, ТЕСТУВАННЯ	59
3.1 Реалізація користувацького інтерфейсу, побудова React-застосунку.....	59
3.2 Огляд роботи, інтерфейс вебсторінки	67
3.3 Тестування роботи системи	75
Висновки до третього розділу	76
ВИСНОВКИ	77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	78
ДОДАТОК.....	80

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface, використовується для взаємодії між фронтендом і бекендом, дозволяє отримувати та відправляти дані.

SPA - Single Page Application, веб-додаток, який завантажує лише одну HTML сторінку.

JWT - JSON Web Token, зберігає інформацію про сесію та дозволи, використовується для аутентифікації.

JSX – JavaScript XML, розширення синтаксису JavaScript, дозволяє використовувати HTML-подібні теги для побудови компонентів у React.

UI – User Interface, користувацький інтерфейс.

БД – база даних.

CRUD – Create, Read, Update, Delete, це основні методи створення, отримання, оновлення та видалення даних з БД.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Для всіх вже є очевидним той факт, що відеоігри та все, що з ними пов'язано давно переросло з гіковської культури у світову, з цього випливає неабияка увага від людей, як від поціновувачів провести час в цікавому та інтерактивному режимі, так і до професіоналів, які перетворили своє хоббі на роботу або бізнес.

Геймінг є однією з найбільш популярних галузей розваг у сучасному світі, яка все ще продовжує розвиватись, як в ширину, так і в глибину, охоплюючи все більшу аудиторію та рухаючи індустрію вперед, як в медійному плані, так і в плані розвитку технологій розробки сучасних ігор.

Ігри стають все більш високотехнологічними, емоційно насиченими, складнішими у розробці, але тим самим цікавішими, надаючи гравцям нові та унікальні ігрові механіки, для ще більш захоплюючого проведення часу.

Весь цей розвиток, тисячі ігор, які розроблюються та випускаються захоплюють величезну увагу, тим самим об'єднують мільйони гравців, які так чи інакше будуть шукати зручні для них джерела інформації, де можуть знайти ту саму новину про оновлення або випуск їх улюбленої гри.

З розширенням індустрії відеоігор, геймерська спільнота стає все більш активною, надійне джерело інформації може стати чудовим місцем для любителів геймерської тематики.

Саме таким місцем може бути вебсторінка, адже всі мають доступ до інтернету, немає необхідності завантажувати додаткові програми на свою систему, необхідно мати лише браузер.

Отже, у кваліфікаційній роботі, буде наведено процес розробки вебсторінки ігрового порталу, який спеціалізується на геймерській тематиці, для того, щоб надати гравцям та звичайним шанувальникам відеоігор чудову платформу для отримання актуальної інформації, широкий спектр якої буде пропонувати вебсайт, включаючи новини, статті та огляди про різні ігрові новинки, оновлення, гайди або просто цікаву інформацію.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

Актуальність теми: з високої популярності та широкої розповсюдженості по всьому світу культури відеоігор, маємо величезну спільноту геймерів. В цих умовах існує потреба в доступному та надійному джерелі інформації про ігри, актуальні останні новини, огляди цікавинок, освітлення різних подій та думок гравців.

Мета: надати шанувальникам ігор платформу з доступом до новин, оглядів і статей про їх улюблені відеогри. Дослідження можливостей розробки сучасних вебсайтів.

Об'єкт дослідження: вебсторінка для спільноти геймерів, дослідження їх можливих потреб у платформі, визначення на основі цього вимог та критеріїв для створення сайту, визначення зручних та ефективних способів розробки.

Предмет дослідження: розробка та оптимізація системи ігрового порталу, вивчення можливостей розробки сучасних веб-додатків.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі та технічне завдання на розробку вебсайту

Основна задача – розробити вебсторінку, яка буде включати в себе актуальну інформацію на геймерську тематику, а саме буде містити публікації, з розділенням на новини, статті та огляди. Дані пости можуть в себе включати посібники, цікаву та актуальну інформацію про ігри, сторінки де можна буде побачити розбір новинок або інформацію про оновлення та патчі вже існуючих. Зміст самих публікацій загалом буде залежити від автора поста.

Ігрова індустрія стрімко розвивається, можна просто порівняти теперішній рівень і те, що було 10 років назад, різниця очевидна. Ця сфера охоплює величезну базу людей, що вже казати, про окреми ігрові серії, які давно перейшли межу просто відеігор та йдуть нога в ногу з кіноіндустрією, по іграм знімають фільми та серіали, пишуть книги, малюють комікси. Теж саме можна сказати і навпаки, багато фільмових саг, звичайно, що хочуть розробляти ігри з їх персонажами і світом.

З цього випливає, що вебсторінка ігрового порталу буде чудовим місцем для поціновувачів ігрової тематики, тож подібна система має буде добре продумана і пропрацьована.

Розробка вебсайту як і будь якої програмної системи включає в себе чимало етапів та охоплює велику кількість задач для їх вирішення.

З огляду на сучасні технології для розробки, виділимо чотири основних етапи: проектування вебсайту, розробка серверної частини, розробка клієнтської частини, тестування та налагодження системи.

Розглянемо вимоги до системи:

- Перше з чим стикається користувач це інтерфейс, тож він має бути інтуїтивно зрозумілий, не нагромаджений, приємний на вид дизайн;
- Контент головної сторінки має одразу давати користувачу те, що він хоче, одразу має бути присутній блок публікацій, адже заради цього він і зайшов на сайт;

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

- Навігація по вебсайту має бути зрозуміла, повинно бути розділення на певні розділи публікацій, наприклад новини, статті, огляди;
- Має бути допоміжний функціонал для користувача, якщо він зайшов з на сайт з певною метою. Це наявність зручного пошуку на вебсторінці.
- Зміст контенту має відповідати зазначеній тематиці;
- Швидкість роботи сайту це один з ключових моментів, тож перфоманс має бути пропрацьований. Зайві перерендери або підвантаження нікому не потрібні;
- Весь запланований функціонал має коректно працювати.

1.2 Аналіз існуючих програмних продуктів за тематикою проекта

Для пошуку та аналізу аналогічних вебсторінок, спочатку потрібно виділити основні критерії за якими будуть визначені відповідні вебсайти для дослідження.

Критерії для пошуку існуючих вебсторінок:

- Наявність блоку публікацій з можливістю переходу на сторінку відповідної новини, статті тощо;
- Можливість пошуку на сайті;
- Приємний на вид та зручний інтерфейс, інтуїтивно зрозумілий для користувача, має бути адаптивним;
- Сама публікація має включати в себе не тільки текст, має бути присутня текстова стилізація (абзаци, виділення кольорів, різні шрифти), наявність зображень тощо;
- Можливість подивитися автора публікації, перейти на сторінку з його іншими роботами;
- Наявність пагінації або «нескінченного» скролу публікацій;
- Хороший перфоманс вебсайту, все має працювати швидко;
- (опціонально) наявність авторизації, що додасть можливість оцінювати публікації, залишати коментарі, редагувати свій профіль тощо.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

Отже було виділено три приклади аналогічних програмних програмних продуктів:

1) Game Rant – вебсайт новин, оглядів, посібників на ігрову тематику [1].
Опис від самих власників сайту – «надаємо контент, написаний геймерами для геймерів», повністю описує призначення сайту та його можливості.

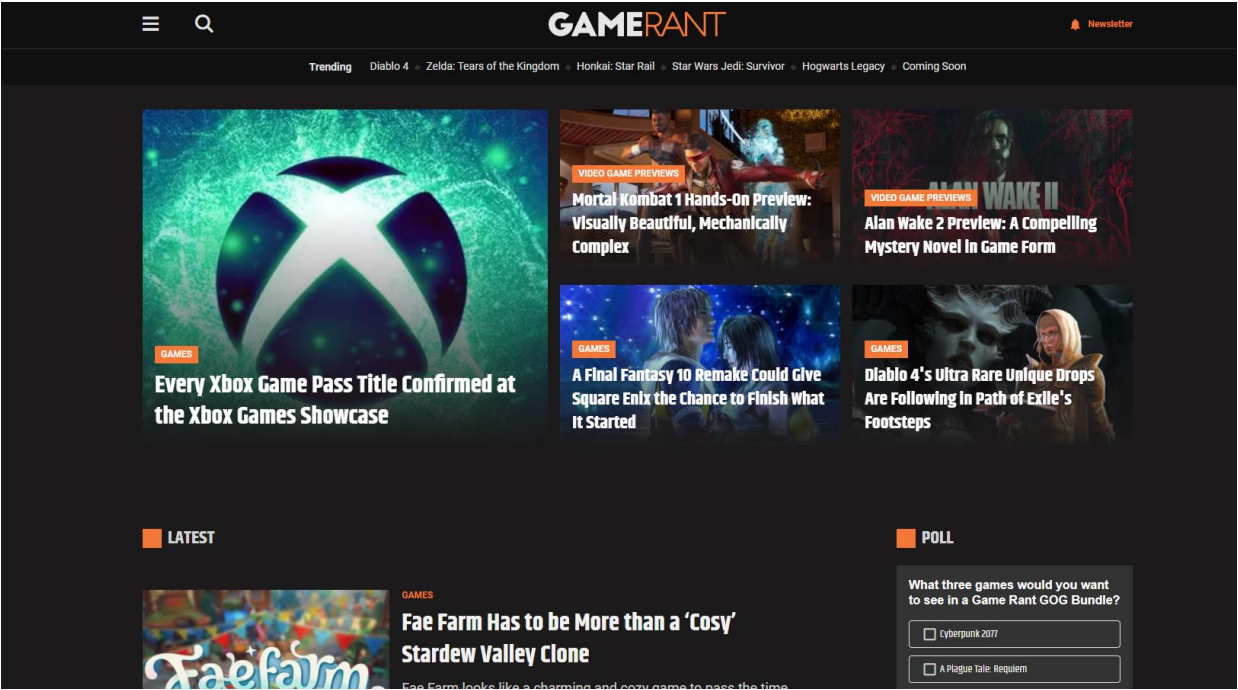


Рис.1.1. Головна сторінка сайту Game Rant

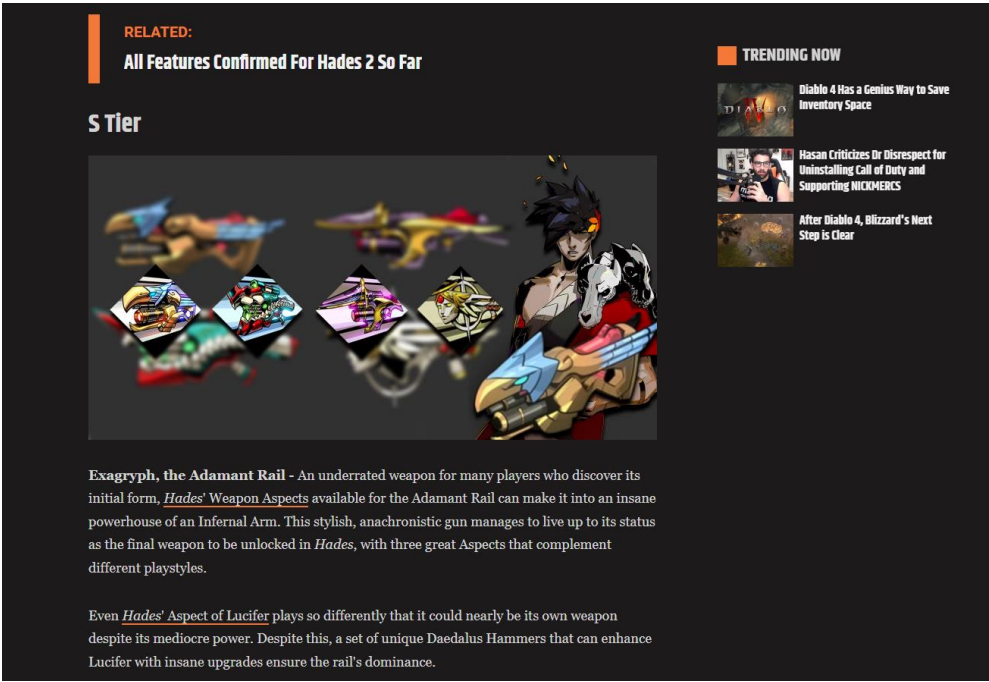


Рис.1.2. Приклад оформлення публікації

Переваги:

- Приємний, зручний інтерфейс;
- Наявна адаптивність;
- Присутній пошук;
- Є розділення публікацій на категорії;
- Присутня можливість переглянути автора публікації;
- Хороший перформанс вебсторінки.

Недоліки:

- Немає можливості оцінити публікацію;
- Відсутні коментарі;
- При перегляді конкретної публікації, ми знаходимось на відповідній сторінці, але в кінці публікації підвантажується одразу наступна, це трошки збиває і не є доречним.

2) IGN – популярний вебсайт, який охоплює велику частину контенту пов'язну з іграми, фільмами, ігровою технікою, містить новини, огляді, статті, інформацію про ігри, фільми тощо [2].

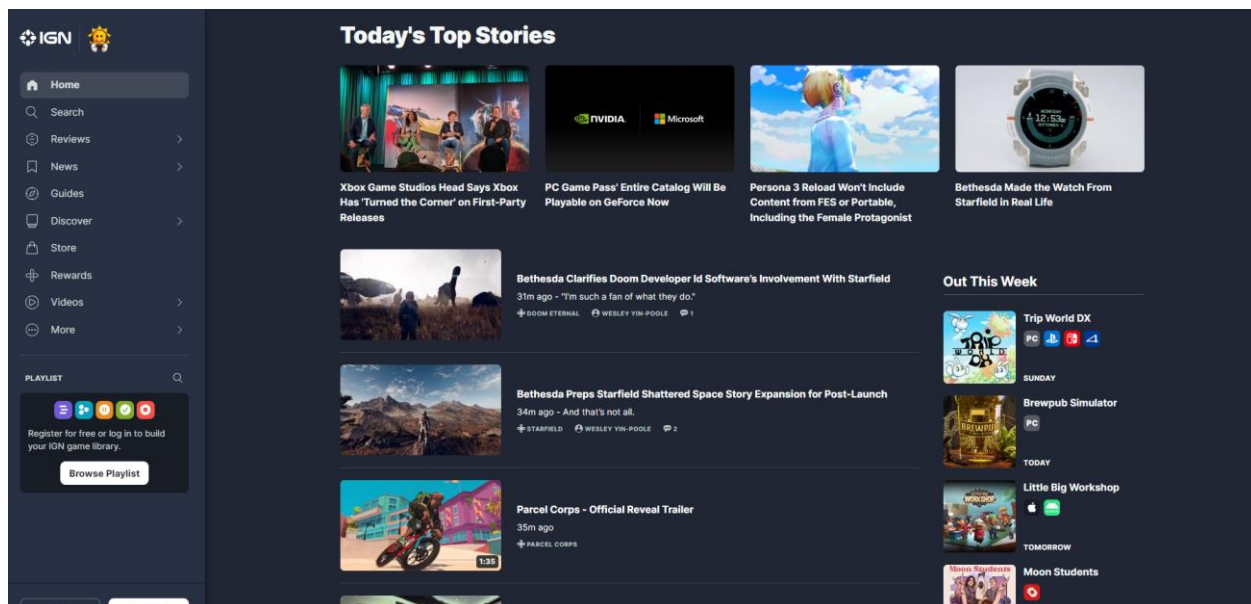


Рис.1.3. Головна сторінка сайту, блок з популярними публікаціями

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

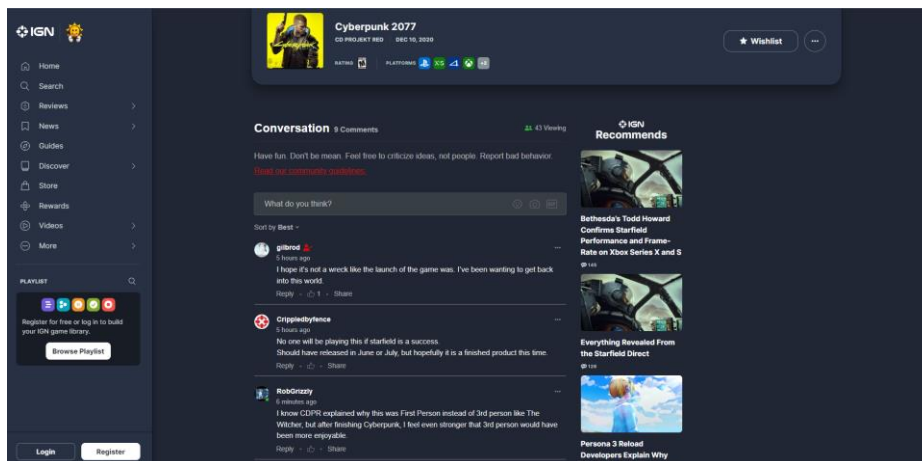


Рис.1.4. Новина, блок з коментарями до публікації

Переваги:

- Приємний, зручний інтерфейс, швидко працює, є адаптивність;
- Великий функціонал;
- Багато контенту, велика кількість публікацій на різні тематики;
- Присутній хороший пошук з розділенням на категорії пошуку;
- Є можливість залишати коментарі під публікацією;

Недоліки:

- Немає, весь зазначений функціонал наявний.

3) PlayUA – українське розважальне інтернет-видання, пишуть про відеоігри, кіноіндустрію, мальовані історії, технології та медіа загалом [3].

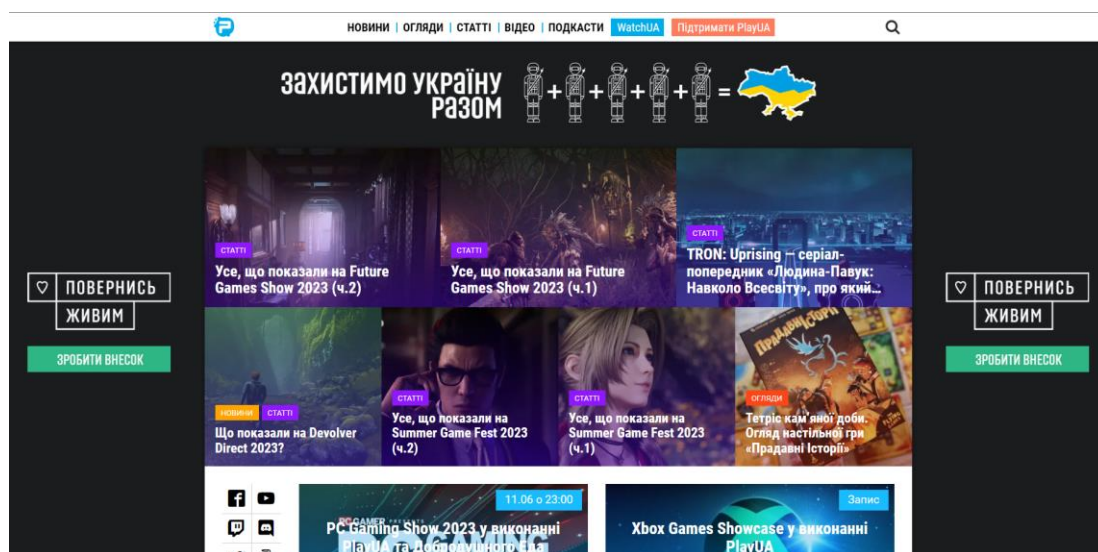


Рис.1.5. Головна сторінка сайту PlayUA

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

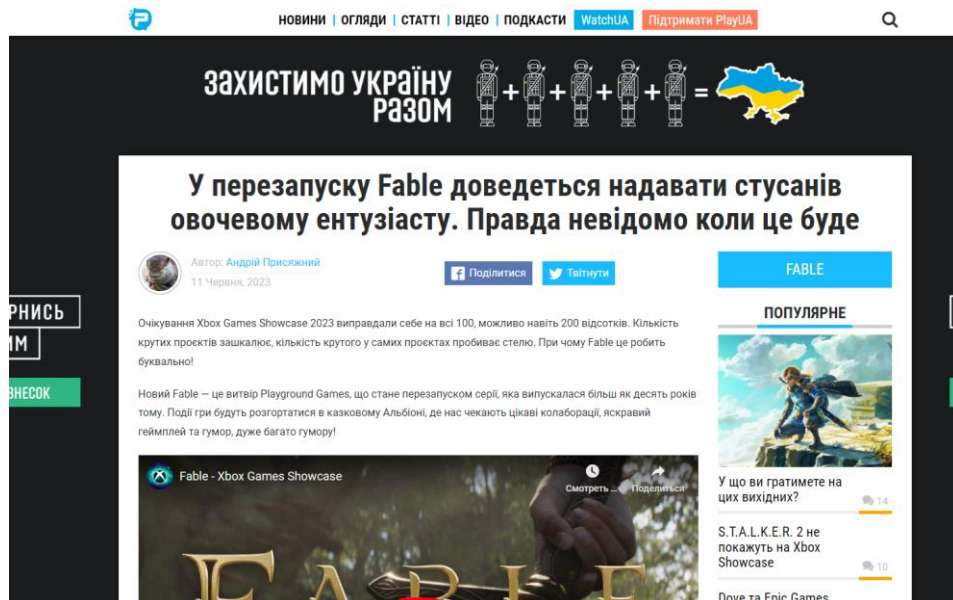


Рис.1.6. Публікація на вебсторінці

Переваги:

- Хороший дизайн сайту, зручний та швидкий інтерфейс, адаптивний;
- Є розділення публікацій на категорії;
- Присутній пошук;
- Є можливість оцінювати статтю та залишати коментарі під публікацією.

Недоліки:

- Немає, весь зазначений функціонал наявний.

Отже, як бачимо, майже всі вебсторінки мають схожий функціонал та загалом мають все, що потрібно, все що відповідає тематиці. Тож можемо брати критерії, за якими здійснювався пошук та аналіз аналогів, як основу для розробки власного вебсайту.

Тож основні риси проекту це: зручний та швидкий інтерфейс, сторінки публікацій мають текстову стилізацію, присутні зображення, є розділи на категорії публікацій, присутній пошук, авторизація, можливість оцінювати пости та залишати коментарі, перегляд автора поста, його інші публікації, не нагромадження головна сторінка, користувачу одразу зрозуміло що і до чого, зручна навігація по вебсайту.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

1.3 Обґрунтування вибору інструментальних засобів розробки

Веб технології існують дуже давно, тож не дивно, що вони постійно розвиваються, завжди виходять оновлення, покращуються різні фреймворки та бібліотеки. Наразі існує багатів підходів до розробки сайту, можуть використовуватися різні типи архітектур, виділимо декілька з них:

- Монолітна. Все розгортається як один цілий додаток, який містить усі компоненти, такі як база даних, частина серверного коду, фронтенд і тд;
- Клієнт-серверна. Основа ідея – розподіл серверним компонентів і клієнтських, тобто браузерних [4]. Добре підходить для проектів, що масштабуються, зручна у використанні;
- Мікросервісна. Вебсайт розбивається на невеликі, автономні та взаємодіючі сервіси, вони працюють окремо. Кожен сервіс відповідає за свою конкретну задачу, виконує тільки свій функціонал.

Тож існує чимало видів архітектур, від простих до більш складних [5]. Вибір залежить від масштабу та складності проекту, від кількості людей які над ним працюють тощо.

У випадку розробки кваліфікаційної роботи, з огляду на вимоги до вебсайту, його функціонала та можливостей, найбільш доречним буде вибір клієнт-серверної архітектури.

Слід визначити архітектурний підхід до розробки вебсайту. Їх також існує досить багато REST API, GraphQL, SOAP, gRPC, OData тощо. Але зупинюсь на перших двох, вони є найбільш популярними, оберемо один з них.

REST API – використовує протокол HTTP для обміну даними між сервером та клієнтом, найбільш поширені це GET, POST, PUT, PATCH, DELETE запити [6]. Використовує ресурси, які можна назвати сутностями, можуть бути адресовані та підлягають взаємодії. Ідентифікація ресурсів складається з базового шляху API та шляху до конкретного ресурсу, наприклад /api/posts/225. Даний підхід забезпечує простий та розширюваний спосіб взаємодії між клієнтом та сервером, якісно та швидко дозволяє працювати з даними.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

GraphQL – є мовою та середовищем виконання запитів, по суті є альтернативою традиційному REST API, надає зручний та потужний спосіб отримання даних з сервера. Одна з характеристик, це можливість гнучкого вибору даних, тобто можна чітко вказувати, яку саме інформацію потрібно отримати, це дозволяє уникати перевантаження мережі та сервера [7].

Таблиця 1.1.

Порівняння архітектурних підходів до розробки вебсайту

REST	GraphQL
Шлях запити + http метод (ендпоінт, кінцева точка) ідентифікують ресурс (дію на сервері)	Запит формується квері-виразом, який ідентифікує ресурс\дію
API відокремлений від фронтенду	API відокремлений від фронтенду
Найпоширеніший тип API, простий та зручний у використанні	Популярний підхід, але менш поширений, оскільки треба знати мову запитів яка використовується

Зважаючи на потреби і вимоги вебсайту та орієнтуючись на дослідження і аналіз, оберемо REST API архітектурним підходом до розробки ігрового порталу.

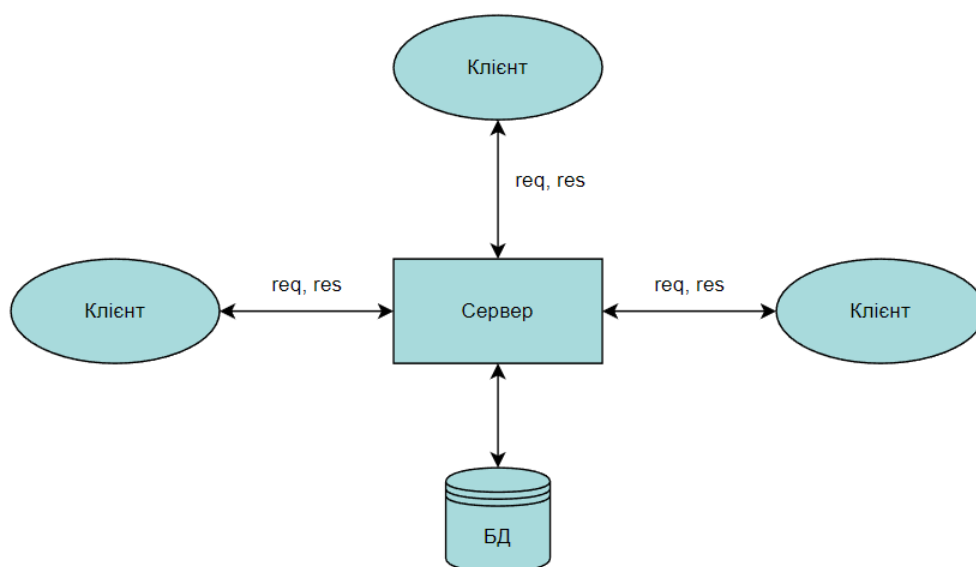


Рис.1.7. Діаграма принципу роботи клієнт-серверної архітектури

Тепер потрібно обрати технології розробки. Почнемо з вибору бази даних, їх також існує купа, основна їх різниця це використання SQL чи ні, та гнучкість схем або таблиць [8]. В мене одразу падає вибір на MongoDB.

MongoDB – документорієнтована, NoSQL база даних, використовує гнучкі схеми, не вимагає фіксованих таблиць(колекцій), кожен документ може мати свою власну структуру, що дозволяє динамічно змінювати схему без внесення змін у всі запити [9]. Якщо даний аспект використовувати правильно, то це абсолютна перевага над іншими базами даних, у випадку розробки вебсайту. MongoDB також надає великий набір можливостей запитів для фільтрації даних, наприклад одна з надзручних можливостей це вбудований пошук та агрегація даних. Також можна виділити зручність масштабування, можливість розподілення даних по кластеру серверів.

Керування базою даних буде здійснюватись через MongoDB Atlas – керувний сервіс хмарної бази даних, надає простий і зручний спосіб управління БД [10].

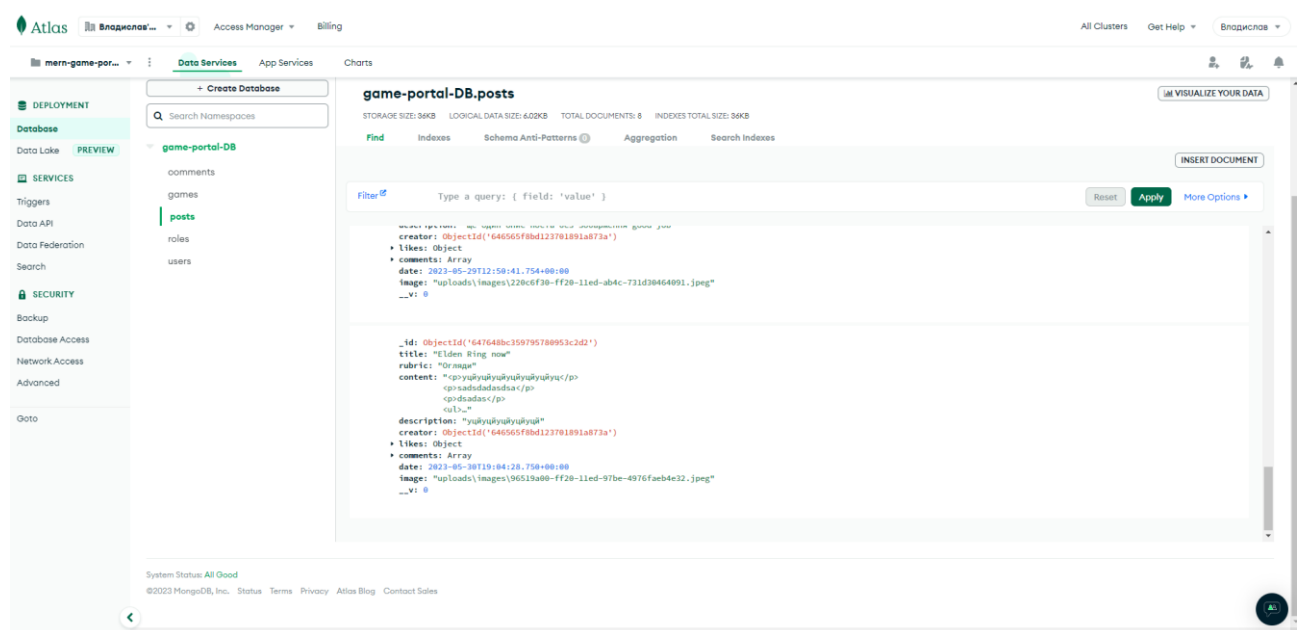


Рис.1.8. MongoDB Atlas

Розробка серверної частина – ключовий момент. Оскільки було обрано архітектурний підхід REST, то доречним буде використання Node.js в поєднанні з Express.js.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

Node.js – крос-платформне середовище, яке дозволяє виконання JavaScript коду на стороні сервера [11]. Основні особливості і переваги це асинхронна модель програмування, дозволяє виконувати багато операцій без блокування потоку. Висока продуктивність, двигун V8 забезпечує швидку компіляцію. Широка екосистема, існує багато модулів, бібліотек і фреймворків, які добре поєднуються з Node.js.

Express.js – один з тих самих фреймворків, який добре поєднується з Node.js. Надає простий та ефективний спосіб створення API, є чудовим інструментом для побудови маршрутизації, легко розширюється, базується на концепції middleware, що дозволяє додавати функціонал через обробку запитів [12].

Фронтенд. При побудові сучасних вебсайтів маємо три стовпа, три основних технології які найчастіше використовуються. Це Vue, React та Angular. Так чи інакше вони виконують одну функцію – побудову веб-інтерфейсів, надаючи зручні інструменти та підходи для цього [13]. Вибір залежить від специфіки проекту та від самих розробників. Тож мій вибір падає на React.

React – відкрита JavaScript бібліотека для розробки користувацьких інтерфейсів. Використовує віртуальний DOM для ефективного оновлення відображення, реакт порівнює віртуальне DOM дерево зі старим станом та вносить тільки необхідні зміни [14]. Одна з характеристик реакту – це компонента модель, можна розбивати потрібні елементи на компоненти, які будуть самостійні і можуть бути повторно використанні де потрібно, це спрощує розробку та і в цілому є зручним підходом.

Величезний плюс реакта це його екосистема, як і в Node.js. Існує безліч бібліотек та надбудов, які прекрасно поєднуються з реактом. Вони спрощують та пришвидшують розробку. Це можуть бути фундаментальні бібліотеки, наприклад react-router-dom – забезпечує маршрутизацію при цьому зберігаючи принцип SPA (Single Page Application) [15]. Бібліотеки з готовими дизайнами, просто готовими компонентами, наприклад карусель зображень тощо.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

Таким чином складається стек MERN. Досить популярний та ефективний для розробки сучасних фулстак веб-додатків.



Рис.1.9. Стек технологій MERN

Висновки до першого розділу

У ході виконання першого розділу була поставлена задача на виконання кваліфікаційної роботи, були визначені основні вимоги до розробки проекту.

Були досліджені та проаналізовані аналоги вебсайтів по поставленим критеріям. Це допомогло визначитись з основними цілями розробки, який має бути мінімальний функціонал, як має бути побудовано користувацький інтерфейс, як стилістично має подаватись основний контент у вигляді публікацій на вебсторінці.

Маючи вимоги та бачення вебсайту, було обрано потрібну та оптимальну архітектуру для розробки проекту, було визначено і пояснено вибір стеку технологій для реалізації поставлених задач – MongoDB, Express.js, React, Node.js.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ВЕБСАЙТУ

2.1 Визначення варіантів використання та алгоритмів роботи системи

У даному розділі буде описано проектування та розробка програмного продукту, відповідно до поставлених задач та вимог з першого розділу кваліфікаційної роботи.

Як було визначено раніше, один з етапів розробки це проектування. Надзвичайно важливий етап, адже він дає чіткий напрямок для подальших дій. На етапі проектування ми визначаємо можливі варіанти використання системи, її алгоритми роботи та загалом майбутню модель, це дозволить більш точно і коректно будувати систему вебсайту. Вже на даному етапі ми можемо побачити як має все працювати задля уникнення проблем у подальшій розробці.

Почнемо з діаграми варіантів використання (Use Case Diagram) – UML діаграма, яка зображає вимоги до розроблюваної системи, ідентифікує зв'язки та відношення між акторами і прецедентами в системі.

На даній діаграмі виділено 4 актори – користувач, зареєстрований користувач, адміністратор та власник. Різниця між акторами - це їх ролі та доступ до певних прецедентів, наприклад звичайний користувач немає ніяких зв'язків з «Діями над постами».

Тобто таким чином, ми бачимо усі можливі варіанти використання системи для відповідних акторів.

Можливості акторів:

- Користувач: може переглядати публікації, дані про ігри, здійснювати пошук, може зареєструватись;
- Зареєстрований користувач: перегляд, пошук публікацій та ігор, можливість ставити лайк та писати коментарі під постами;
- Адміністратор: такі ж можливості як зареєстрований користувач + має доступ до створення, редагування, видалення постів та даних про ігри;
- Власник: можливості адміністратора + перегляд списку користувачів, доступ до зміни їх ролей.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

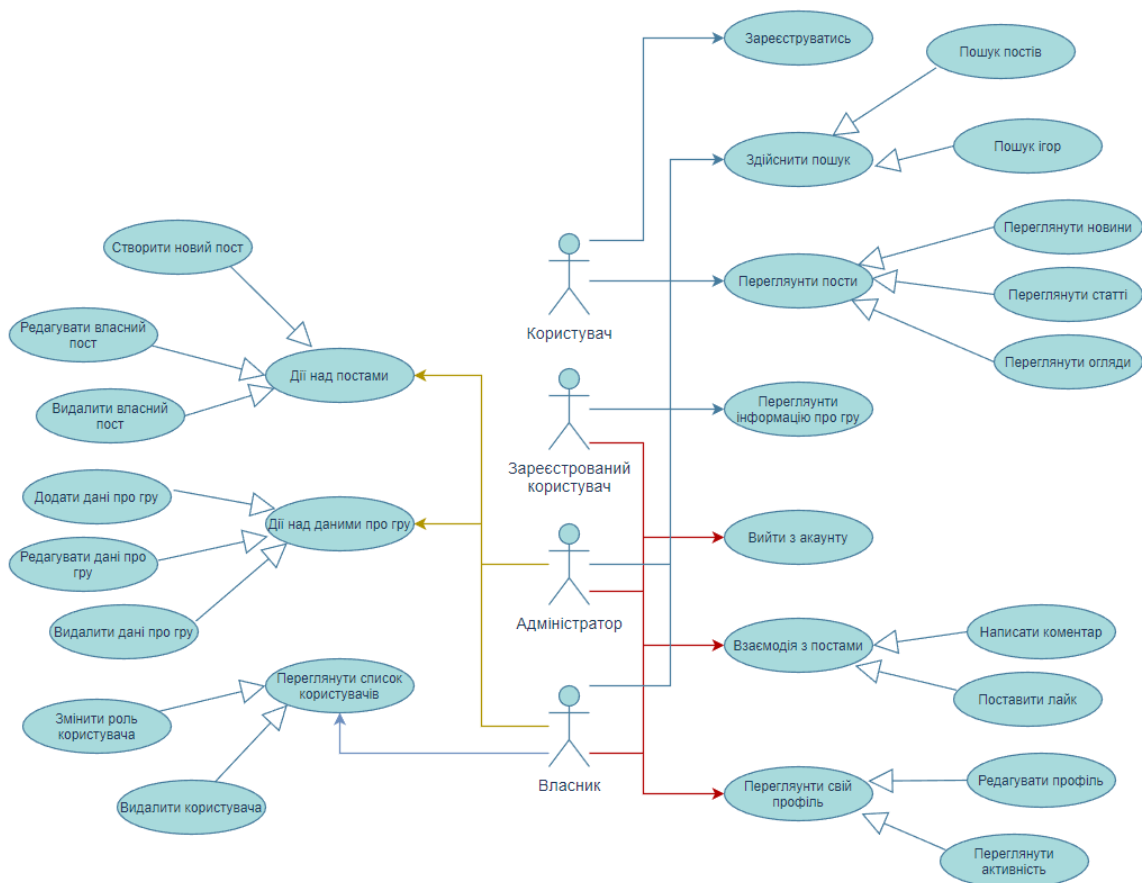


Рис.2.1. Діаграма варіантів використання

Тепер слід визначити основні алгоритми роботи та стани системи вебсайту. Це можна зробити за допомогою діаграми діяльності (Activity diagram) – візуальне представлення та моделювання послідовності дій, процесів у системі. Дана діаграма дозволяє моделювати послідовність дій у процесі, відображати шляхи виконання, умови й альтернативні процеси.

Діаграма діяльності складається з певних дій:

- Дія (action): етап діяльності, конкретне завдання, певні кроки або дії;
- Рішення (decision node): умовний шлях, який має один або декілька виходів, мають певну умову;
- Керування (control flows): з'єднувачі між кроками, вказують порядок дій;
- Початкова точка (start node): початковий вузол, початок діаграми;
- Кінцева точка (end node): кінцевий вузол, останній крок, дія, завершення активності.

Можна зробити одну велику діаграму діяльності для показу алгоритмів роботи вебсайту, але більш доречно буде зробити декілька, з різними початковими точками, тим самими спростити сприйняття даних діаграм.

Почнемо з взаємодії з публікаціями звичайного користувача.

Початковий стан – відкриття вебсайту. Далі користувач має вибір, це буде розділення публікацій на категорії. Обираючи певну категорію та відповідну публікацію, користувач має дії. Перша з них це натиск лайка, після якого йде рішення – перевірка авторизації користувача, бо ставити\прибирати лайки, може лише авторизований користувач. У випадку, якщо користувач авторизований, далі йде рішення, прибрати лайк після натиску чи поставити його, після чого активність завершується. Принцип роботи коментаря аналогічний. Перехід на сторінку з іншими постами автора також завершує активність.

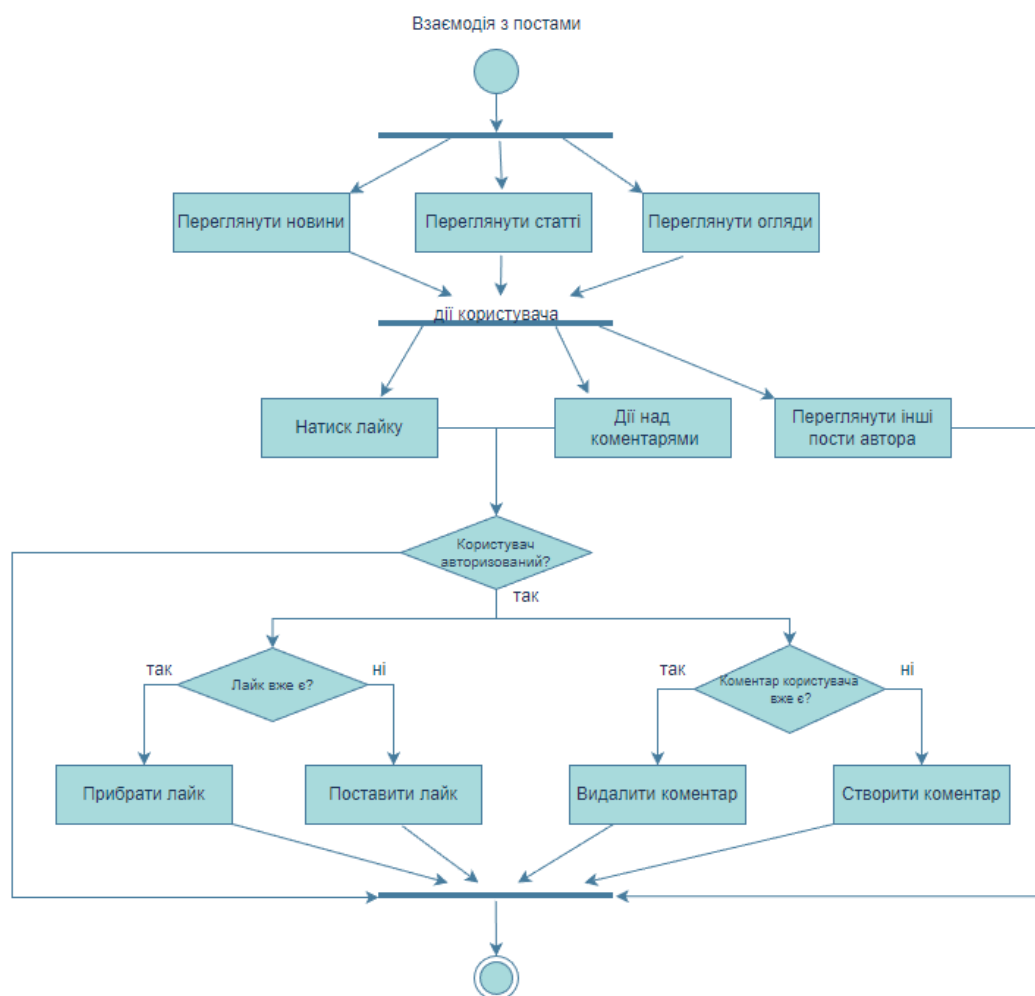


Рис.2.2. Діаграма діяльності, взаємодія з постами

Розглянемо діаграму діяльності дій над постами у разі авторизованого користувача - автора публікації.

Початковий вузол – відкриття сайту. Після вибору публікації за категорією, адміністратор має такі активності як редагування або видалення публікації. Вони будуть працювати за схожим принципом, після оновлення даних або натиску видалення – треба підтвердити свою дію, у разі підтвердження – переходимо у кінцеву точку, в іншому випадку активність продовжується. Також адміністратор одразу після початкового стану має можливість створення нової публікації, після її вибору йде вже знайоме рішення підтвердження дії.

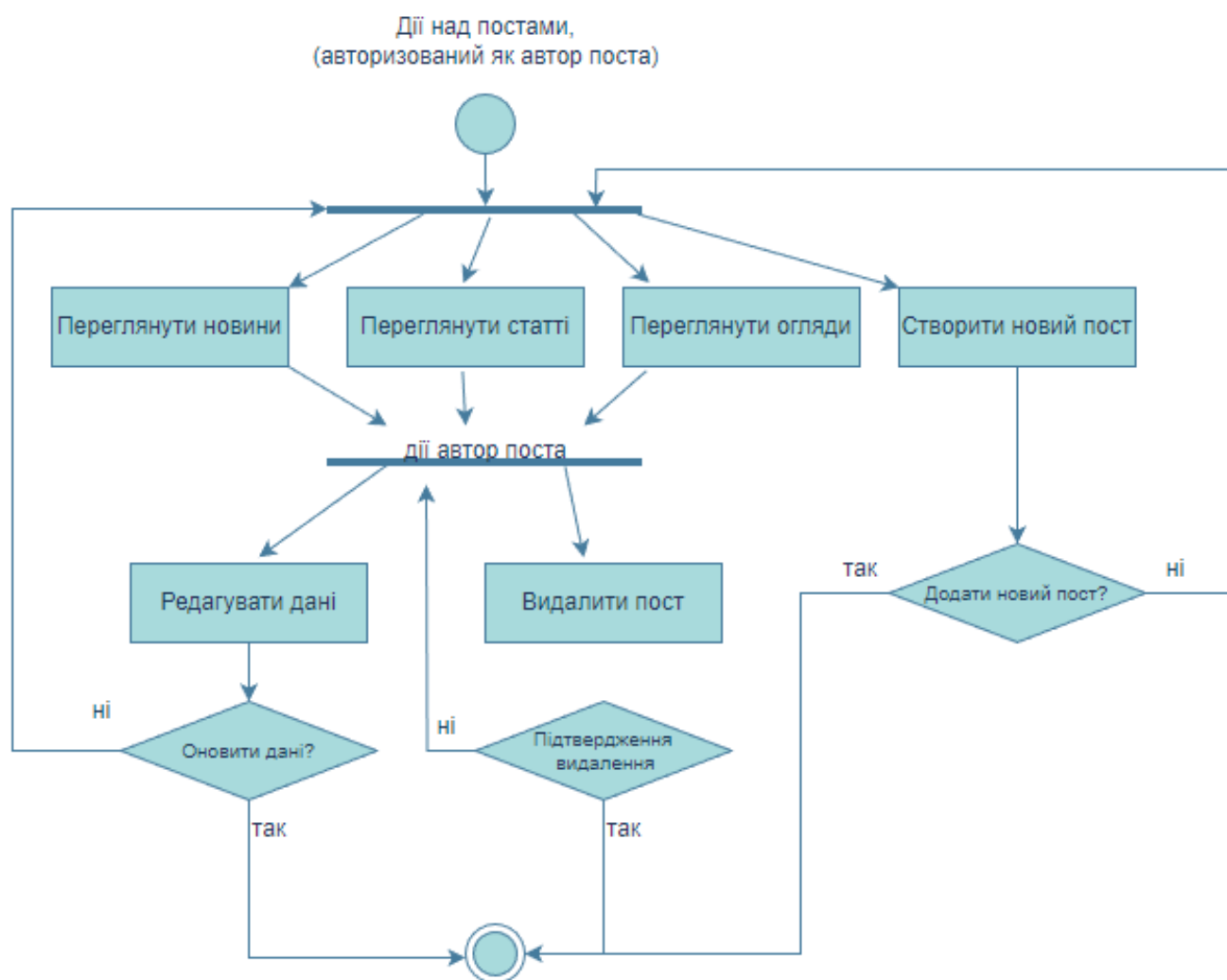


Рис.2.3. Діаграма діяльності, дії над постами

На вебсайті планується, крім публікацій, додавання інформаційного блоку про ігри. Діаграма діяльності буде такою самою як попередня. Початковий стан – користувач авторизований як адміністратор, єдина відмінність, йому не обов’язково бути тим самим адміністратором, який додав дані про гру, це може робити будь який адмін.

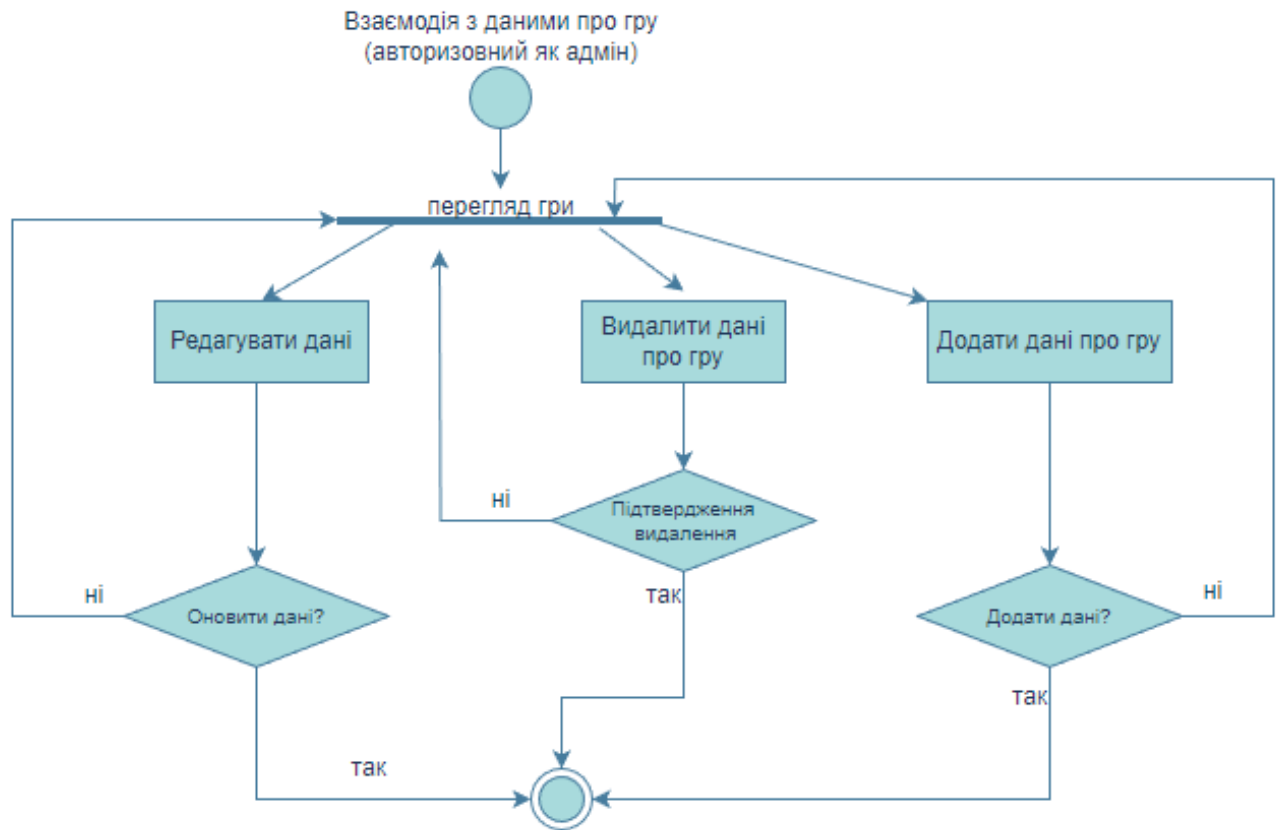


Рис.2.4. Діаграма діяльності, дії над даними про гру

Далі розглянемо невелику діаграму можливостей користувача, який має роль власника, а саме його взаємодію з іншими користувачами.

Особливості даної ролі полягають в тому, що власник може змінювати ролі інших користувачів, надавати права адміна користувачу, який це потребує. В рамках вебсайту ігровго порталу, адміністратор це учасник команди авторів – які забезпечують контентом вебсторінку, тобто роблять публікації.

Після початку активності, власник може перейти до списку користувачів, де має можливість надати\забрати певну роль. Також є можливість видалення

користувача, після якого йде рішення – підтвердження видалення, у разі згоди активність завершується, в іншому – продовжується.

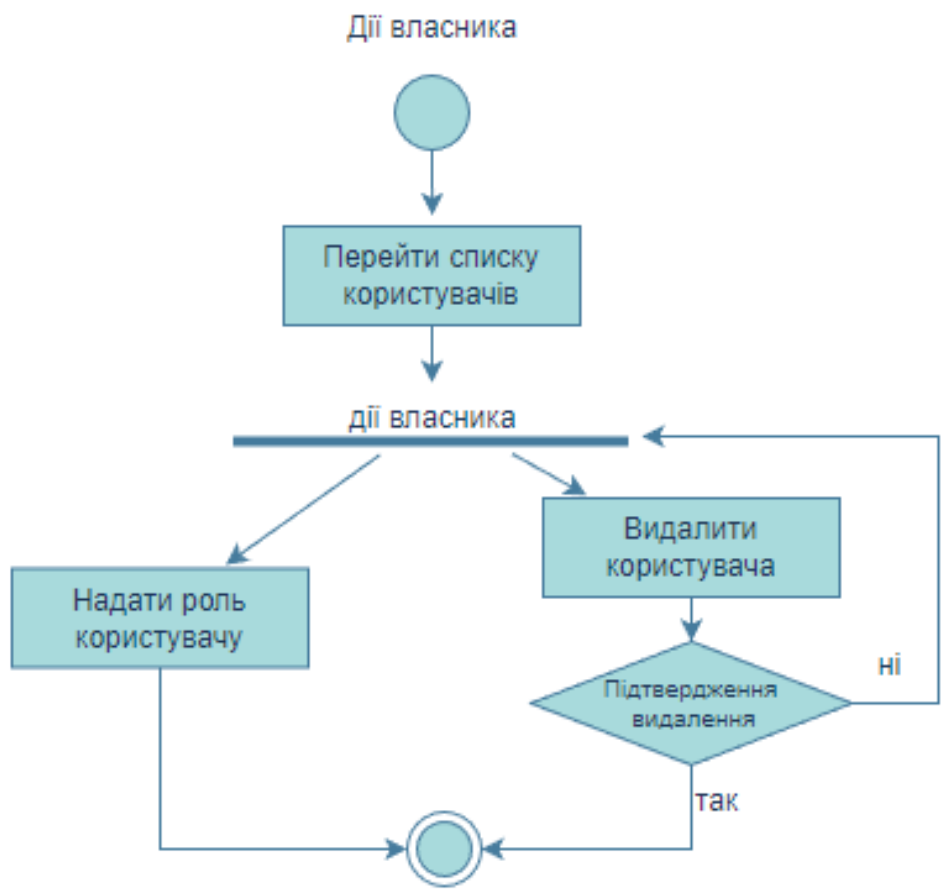


Рис.2.5. Діаграма діяльності, дії користувача з роллю власника

Останню діаграму діяльності яку слід розглянути це авторизація\реєстрація користувача. Початкова точка – відкриття вебсторінки після якої в рамках даної діаграми користувач переходить на сторінку авторизації. Далі він має такі дії як можливість входу у акаунт або реєстрація.

У випадку авторизації, користувач має ввести потрібні дані про свій акаунт, після яких йде перевірка їх валідності. У разі, якщо введені дані вірні, наступним йде рішення – перевірка чи такий акаунт існує. Якщо дані були введені валідно, акаунт існує – користувач авторизується, активність закінчиться, відбудеться перехід у кінцеву точку, якщо ж ні – активність продовжується.

У разі реєстрації, алгоритм схожий, у разі введення валідних даних, користувач може успішно зареєструватись та перейти у кінцевий вузол, який

зображений на даній діаграмі. В іншому випадку, якщо дані не валідні активність буде продовжуватись, користувач матиме змогу повторити цю дію.

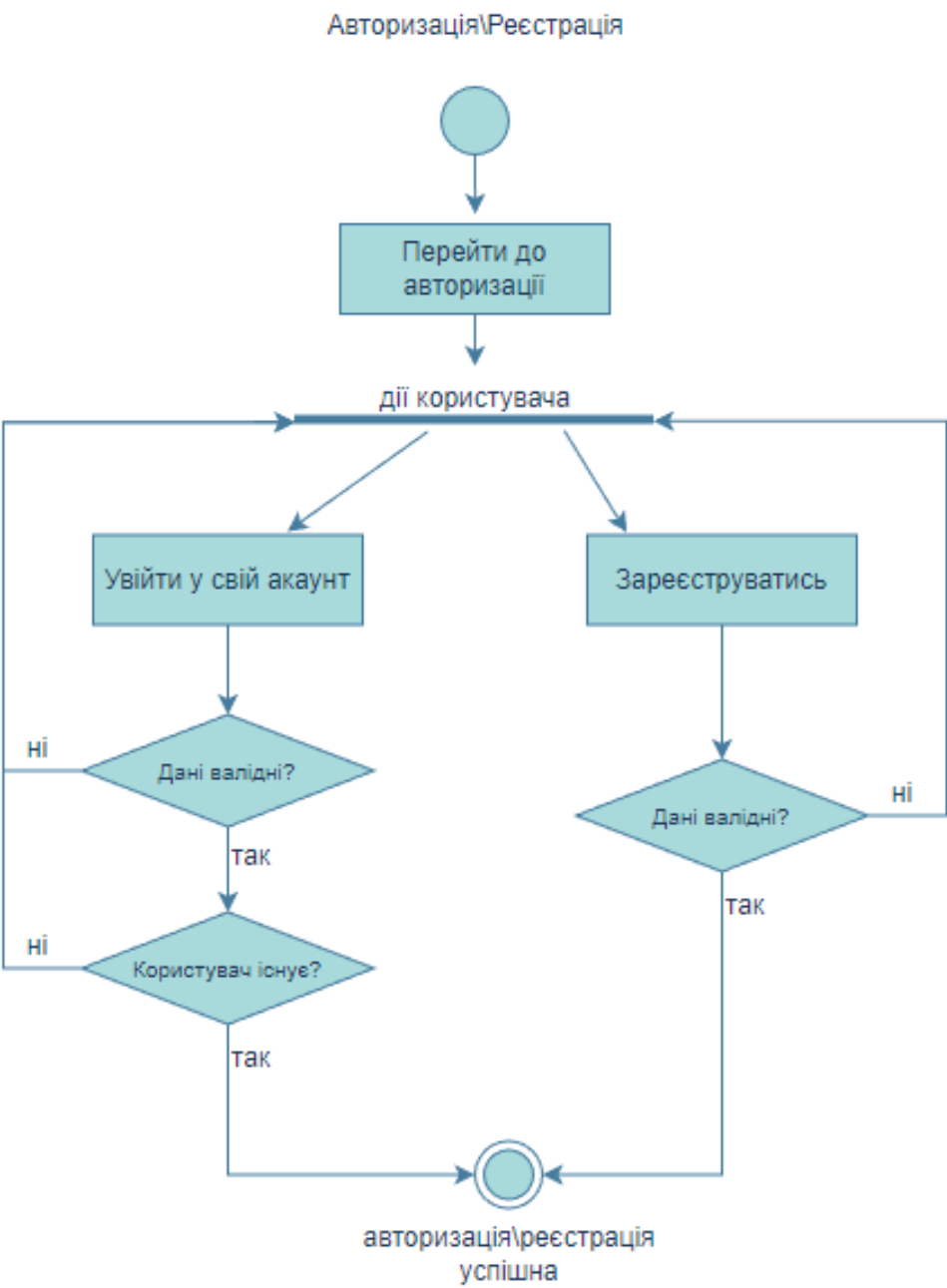


Рис.2.6. Діаграма діяльності, авторизація

2.2 Структура бази даних, взаємодія компонентів системи

Як було зазначено в першому розділі, буде використовуватись MongoDB. Було наведено основні характеристики які властиві даній БД. Тепер розглянемо діаграму бази даних для розроблювальної системи.

Діаграма бази даних – дозволяє візуально представити структуру БД, таблиці (колекції), зв’язки між ними, визначити поля та їх типи.

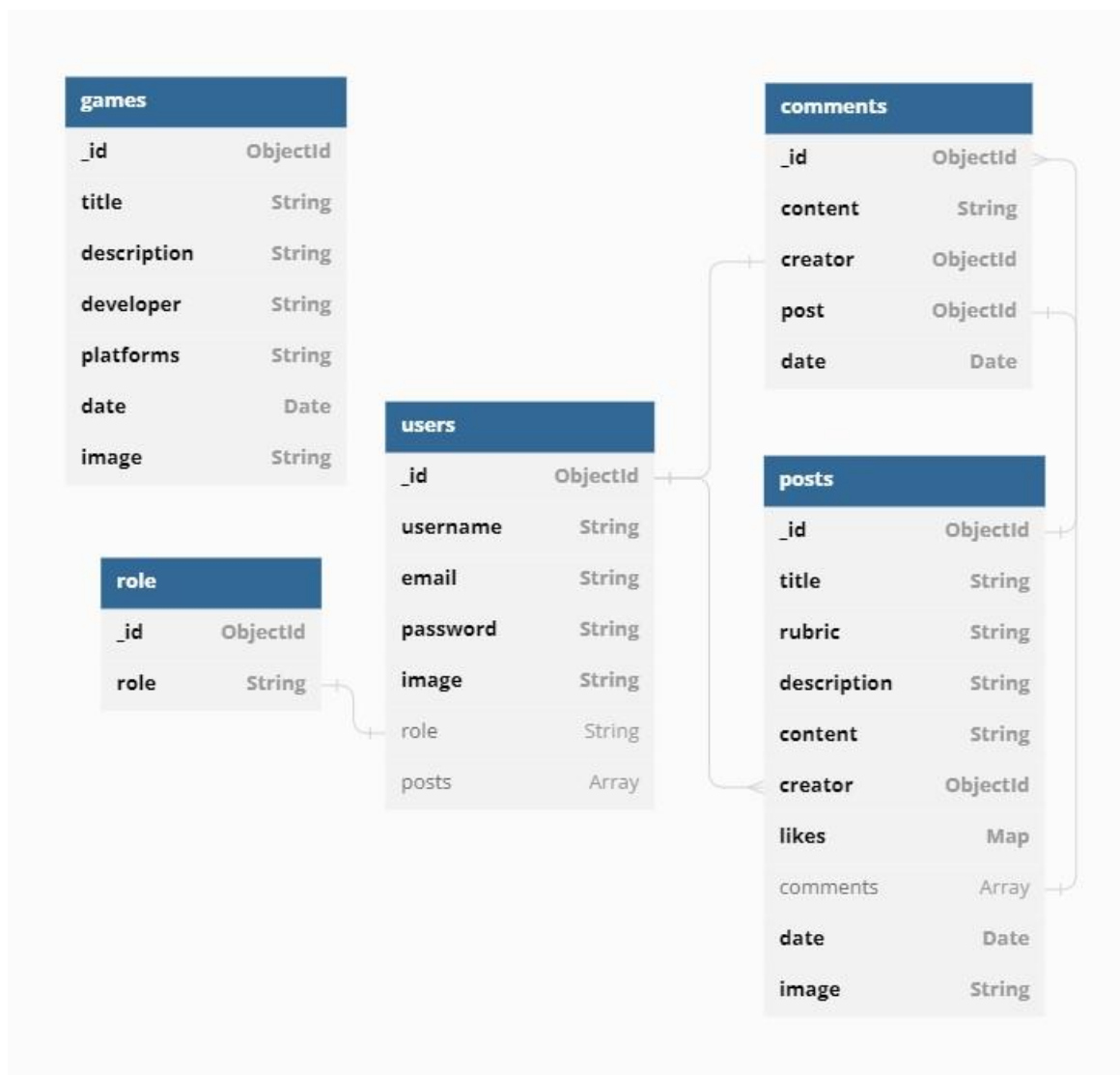


Рис.2.7. Діаграма бази даних

Для зазначених потреб, такої БД буде достатньо, у разі розширення функціоналу, ввести нову колекцію не буде проблемою. Розглянемо більш детальну наявну діаграму.

Колекція **users** – містить дані про користувачів, які потрібні для взаємодії авторизованих користувачів на вебсайті. Містить такі поля:

- **_id**: ідентифікатор користувача, поле має тип даних *ObjectId*, це спеціальний тип даних, який використовує MongoDB для унікальної ідентифікації документів;

- username: нікнейм, псевдонім користувача, тип *String*;
- email: електронна пошта, потрібна для авторизації, має бути унікальною, тип *String*;
- password: пароль до акаунту користувача, потрібен для авторизації, тип *String*;
- image: містить шлях до збереженої на сервері аватарки користувача, тип *String*;
- role: роль користувача, не виділений, адже не є обов'язковим, буде зазначено значення по замовчуванню "USER", тип *String*;
- posts: масив постів, створених користувачем, також не є обов'язковим полем, адже по замовчуванню буде пустий масив, щоб робити публікації треба мати роль "ADMIN".

Колекція **role** – визначення ролей в системі. Містить такі поля:

- _id: ідентифікатор ролі, тип *ObjectId*;
- role: роль користувача, тип *String*.

Колекція **posts** – зберігає дані про публікації. Містить такі поля:

- _id: ідентифікатор публікації, тип *ObjectId*;
- title: назва публікації, тип *String*;
- rubric: рубрика, тип публікації(новина, стаття, огляд), тип *String*;
- description: короткий опис публікації, тип *String*;
- content: головний контент, передбачається, що в строці буде міститись html код, який далі буде відображатись на сторінці публікації, тип *String*;
- creator: зберігає ідентифікатор користувача, який створив пост, тип *ObjectId*;

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

- likes: кількість лайків публікації, тип *Map*. Зберігається ідентифікатор користувача та буліан значення true, у разі якщо користувач поставив лайк до публікації;
- comments: масив ідентифікаторів коментарів які відносять до відповідної публікації. Не обов'язкове поле, адже по замовчуванню пустий масив, бо коментарів при створенні немає;
- date: дата створення поста, тип *Date*;
- image: містить шлях до збереженої на сервері обкладинки публікації, тип *String*.

Колекція **comments** – зберігає дані про коментарі. Містить такі поля:

- _id: ідентифікатор коментаря, тип *ObjectId*;
- content: сам коментар, текст, тип *String*;
- creator: ідентифікатор користувача, який написав коментар, тип *ObjectId*;
- post: ідентифікатор публікації до якої належить коментар, тип *ObjectId*;
- date: дата створення коментаря, тип *Date*.

Колекція **games** – зберігає дані про ігри. Містить такі поля:

- _id: ідентифікатор гри, тип *ObjectId*;
- title: назва гри, тип *String*;
- description: опис до гри, тип *String*;
- developer: розробники гри, тип *String*;
- platforms: платформи на яких підтримується гра, тип *String*;
- date: дата виходу гри, тип *Date*;
- image: містить шлях до збереженої на сервері обкладинки гри, тип *String*.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

Також в даному пункті розглянемо ще одну частину проектування – взаємодію компонентів системи. Це можна зобразити на діаграмі компонентів, яка відображає залежності між компонентами, самі компоненти можна представити у вигляді окермих модулів, які можуть існувати під час роботи програминого додатку, в нашому випадку, вебсайту.

На діаграмі присутні такі компоненти як: пости(публікації), вони містять в собі новини, статті та огляди, які в свою чергу мають прямий зв'язок з коментарями. Пости пов'язані з компонентом користувача через перегляд і пошук. Компонент – дані про ігри також таким чином пов'язаний з користувачем.

Користувач напряду залежить з компонентом авторизація, а вже з авторизації впливає авторизовний користувач, який також має зв'язки з постами та даними про ігри, пов'язаний через взаємодіяти, тобто крім перегляду і пошуку може мати ще дії (сторення, редагування, видалення, при умові наявності необхідної ролі для цього, що також є компонентом).

Авторизований користувач також може взаємодіяти з компонентом профілю користувача.

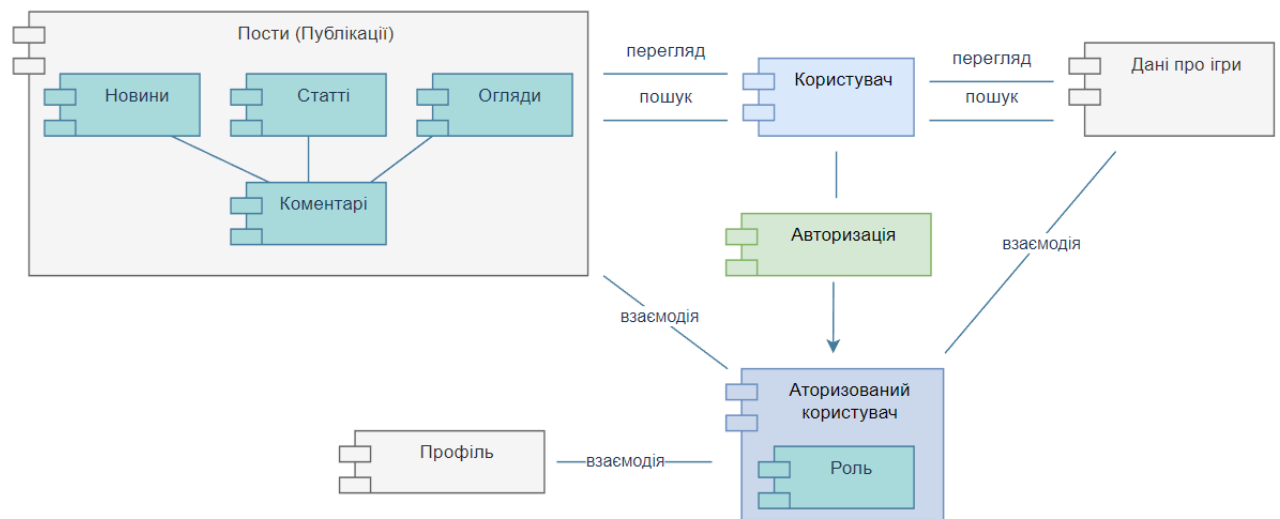


Рис.2.8. Діаграма компонентів

2.3 Реалізація функціоналу серверної частини вебсайту

Опишемо основні моменти розробки функціоналу для клієнт-серверного додатку. Оскільки клієнтська частина повинна брати дані з сервера, тому більшість роботи буде побудовано таким чином, що спочатку ми робимо певні дії на сервері, результат яких будемо повертати на клієнт і там вже з отриманими даними будувати систему реакт застосунку.

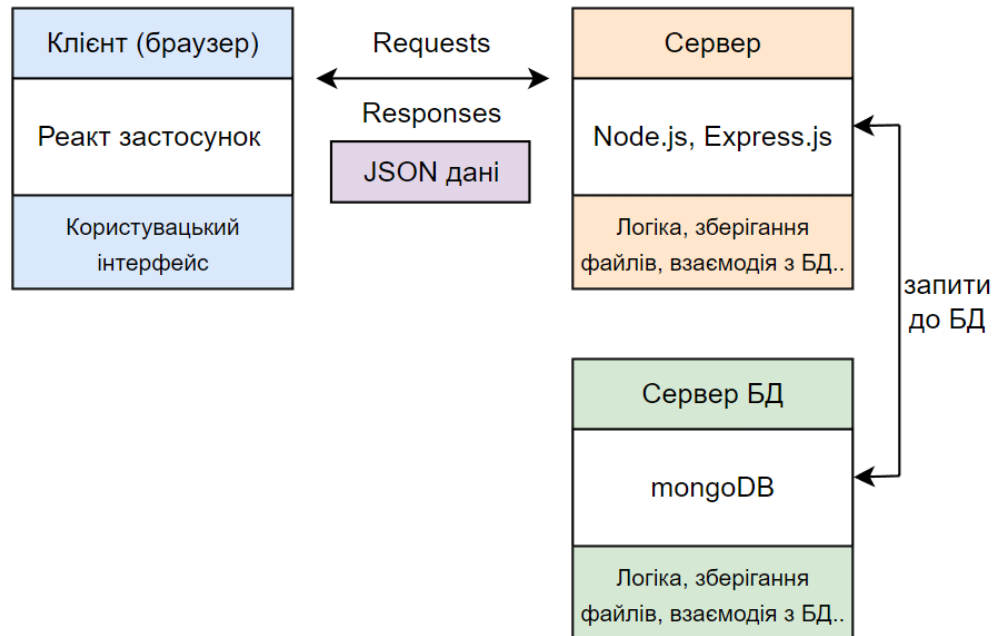


Рис.2.9. Діаграма принципу роботи стеку MERN

Почнемо з побудови серверної частини. Після ініціалізації проекту, встановлення початкових залежностей можемо почати з кореневого файлу `app.js`. Конфігурація зберігаються у файлі `package.json`, він виглядає наступним чином:

```

{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.0",
  
```

```

    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "express-validator": "^7.0.1",
    "jsonwebtoken": "^9.0.0",
    "mongoose": "^7.1.1",
    "multer": "^1.4.5-lts.1",
    "uuid": "^9.0.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}

```

Лістинг.1.1. Конфігурація сервера, package.json

Тут ми вказуємо назву проекту, версію, опис(опціонально), вказуємо головний, кореневий файл – це app.js, вказуємо скрипти, які можемо виконувати з терміналу (командного рядка), було вказано скрипт “start” для запуску сервера, nodemon – дозволяє запустити сервер і автоматично зберігати та застосовувати зміни, тим самим не потрібно кожного разу після змін власноруч запускати сервер. Також файл містить назву ліцензії і залежності, які необхідні для роботи додатку, їх буде розглянуто пізніше, яка і де використовується.

В головному файлі app.js ми створюємо сервер за допомогою Express.js, конфігуруємо його, підключаємо роути, базу даних, оброблюємо помилки.

Даний код створює екземпляр додатку Express, що дозволить далі налаштувати роути, middleware тощо. Також вказуємо, що додаємо middleware яка дозволяє розпарсити вхідні дані у форматі JSON та далі використовуємо middleware «cors», яка вирішує проблему політики міжсайтових запитів (CROSS-ORIGIN-RESOURCE-SHARING), таким чином ми дозволяємо використовувати додаток всім джерелам, не потрібно тепер це прописувати вручну:

```

const app = express();
app.use(bodyParser.json());
app.use(cors());

```

Лістинг.1.2. Створення сервера, використання middleware

Про npm пакети - “body-parser” та “cors” є одними з залежностей в конфігураційному файлі, це пакети, які додають потрібний функціонал, завантажуюємо через “npm install” в терміналі, так само робимо й з іншими які використовуються в проекті [21].

Також слід зазначити про middleware – це програмний код, який виконується в проміжному етапі обробки [17]. Використовується для перевірки даних, обробки помилок, аутентифікації, для перевірки ролей тощо.

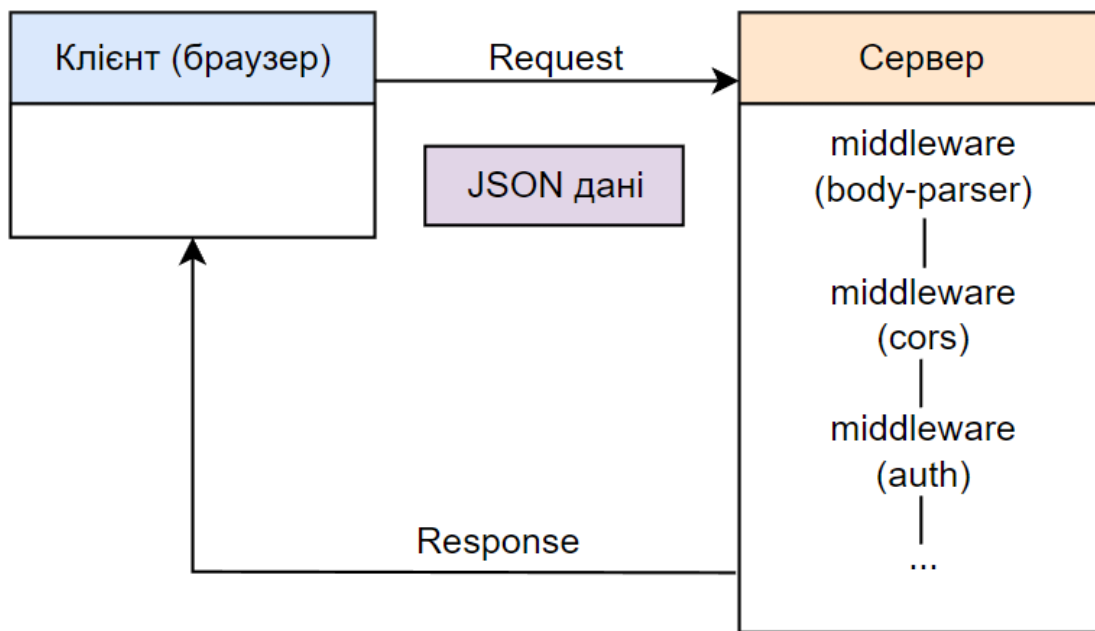


Рис.2.10. Принцип роботи middleware

Продовження файлу app.js. Тут йде встановлення шляху для статичних файлів, в нашому випадку зображень. Обробка помилок. Використання роутів, які ми до цього імпортуємо “import exampleRoute from “./routes/...””.

Та в кінці використовуємо бібліотеку mongoose для з’єднання та роботи з базою даних MongoDB. В методі connect якраз таки і під’єднуємось до бази, даний URL містить інформацію про хост, ім’я користувача, пароль, назву БД. Ми безпечно підключаємось, використовуючи змінні середовища, але про це в наступному розділі. Після успішного встановлення з’єднання з базою, запускаємо express сервер на вказаному порті. Також у разі помилки з’єднання, виводимо її в консоль:

```

app.use("/uploads/images", express.static(path.join("uploads", "images")));

app.use((error, req, res, next) => {
  if (req.file) {
    console.log("see" + req.file);
    fs.unlink(req.file.path, (err) => {
      console.log(err);
    });
  }

  if (res.headerSent) {
    return next(error);
  }
  res.status(error.code || 500);
  res.json({ message: error.message || "An unknown error occurred!" });
});

app.use("/api/users", usersRoutes);
app.use("/api/posts", postsRoutes);
app.use("/api/comments", commentsRoutes);
app.use("/api/games", gamesRoutes);

app.use((req, res, next) => {
  res.status(404).json({ message: "Could not find this route" });
});

mongoose
  .connect(
    `mongodb+srv://${process.env.DB_USER}:${process.env.DB_PASSWORD}@cluster0.juvc9
ln.mongodb.net/${process.env.DB_NAME}?retryWrites=true&w=majority`
  )
  .then(() => {
    app.listen(process.env.PORT);
  })
  .catch((err) => {
    console.log(err);
  });

```

Лістинг.1.3. Шлях статичних файлів, обробка помилок, використання роутів, підключення до БД

При розробці ПЗ часто використовують патерни проектування, вони являють собою багаторазові рішення багатьох проблем та просто такі шаблони забезпечують структуру й організацію коду, що дозволяє проекту бути більш модульним і масштабованим.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

У сучасній розробці бекенду шаблони проектування мають ключове значення для створення гнучкої системи, які можуть адаптуватися до різних вимог і потреб користувачів.

Один з найбільш поширених це MVC (Model-View-Controller) патерн. Головний принцип це відокремити презентаційний шар, від логіки та зберігання даних [16]. У класичному патерні MVC маємо:

- Модель (Model): дані, робота з даними;
- Представлення (View): відображення даних на UI;
- Контролер (Controller): обробка вхідних даних, взаємодія між моделлю і представленням.

Оскільки проект будується на стеку MERN, то ми не можемо розглядати класичний вигляд MVC патерну, але ми використовуємо деякі його принципи. Різниця полягає в тому, що React виступає у ролі представлення (View), оскільки ми його використовуємо для відображення даних і побудови користувацького інтерфейсу.

За допомогою бібліотеки Mongoose ми визначаємо частину даних, зберігаємо їх, створюємо модель. Для написання контролера використовуємо засоби Express та Node.js, пишемо логіку.

Розглянемо декілька моделей з проекту. Загалом моделі створюється по діаграмі БД. За допомогою mongoose створюємо схему де визначаємо потрібні поля, які будуть використовуватись при побудові документів в MongoDB. Визначаємо тип поля, його необхідність та ще деякі параметри по типу мінімальної довжини (password) або те, що поле має бути унікальний (email). Також визначаємо зв'язки з іншими моделями, а саме з роллю та постами:

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
  username: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true, minlength: 4 },
  image: { type: String, required: false },
  roles: [{ type: String, ref: "Role" }],
  posts: [
```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    {
      type: mongoose.Types.ObjectId,
      required: false,
      default: [],
      ref: "Post",
    },
  ],
});

```

```
export default mongoose.model("User", userSchema);
```

Лістинг.1.4. Модель користувача, User.js

Модель публікації (поста) виглядає наступним чином:

```
import mongoose from "mongoose";
```

```

const postSchema = new mongoose.Schema({
  title: { type: String, required: true },
  rubric: { type: String, required: true },
  content: { type: String, required: true },
  description: { type: String, required: true },
  creator: { type: mongoose.Types.ObjectId, required: true, ref: "User" },
  likes: { type: Map, of: Boolean },
  comments: [
    {
      type: mongoose.Types.ObjectId,
      required: true,
      default: [],
      ref: "Comment",
    },
  ],
  date: { type: Date, default: Date.now },
  image: { type: String, required: false },
});

```

```
export default mongoose.model("Post", postSchema);
```

Лістинг.1.5. Модель публікації, Post.js

Також одна з ключевих це коментар:

```
import mongoose from "mongoose";
```

```

const commentSchema = new mongoose.Schema({
  content: { type: String, required: true },
  creatorId: { type: mongoose.Types.ObjectId, required: true, ref: "User" },
  postId: { type: mongoose.Types.ObjectId, required: true, ref: "Post" },
  date: { type: Date, default: Date.now() },
});

```

```
export default mongoose.model("Comment", commentSchema);
```

Лістинг.1.6. Модель коментаря, Comment.js

Перед розглядом контролерів, спочатку слід пояснити принцип роутів. У кореневому файлі app.js ми вказали, що використовуємо роутери:

```
app.use("/api/users", usersRoutes);
app.use("/api/posts", postsRoutes);
app.use("/api/comments", commentsRoutes);
app.use("/api/games", gamesRoutes);
```

Лістинг.1.7. Використання роутів

Їх ми окремо виносимо, там також присутня певна логіка. Наприклад розглянемо users-routes.js.

В даному коді, ми імпортуємо потрібні модулі та middleware. Ключовий це Router – модуль з фреймворку express, дозволяє побудувати маршрутизатори для обробки http запитів. Використовуємо певні middleware, check – встановлений з npm пакета express-validator, використовується для валідації даних, інші middleware – створені власноруч для перевірки авторизації, ролі та для обробки завантаження зображення. Після імпорту прописуємо самі маршрути, які використовують функції з контролера та middleware при потребі:

```
import Router from "express";
import UserController from "../controllers/users-controller.js";
import { check } from "express-validator";
import { checkAuth } from "../middleware/check-auth.js";
import { checkRole } from "../middleware/check-role.js";
import { fileUpload } from "../middleware/file-upload.js";

const router = new Router();

router.get("/", checkRole(["ADMIN", "OWNER"]), UserController.getUsers);

router.get("/:uid", UserController.getUserById);
router.get("/profile/:uid", checkAuth, UserController.getUserProfileInfoById);

router.post(
  "/signup",
  fileUpload.single("image"),
  [
    check("username", "Username must contain at least 3 characters").isLength({
      min: 3,
```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        max: 18,
      }),
      check("email", "Email is not valid").normalizeEmail().isEmail(),
      check("password", "Password must contain at least 4 characters").isLength({
        min: 4,
        max: 18,
      }),
    ],
    UserController.signup
  );
router.post("/login", UserController.login);

export default router;

```

Лістинг.1.8. Роутер користувача, використання методів з контролера

Перейдемо до контролера `users-controller.js`. Ми будуємо контролер за допомогою класу, який має набір певних методів для роботи. Не обов’язково робити це через клас, можна використовувати звичайні методи.

Усі методи контролера є асинхронними, ми використовуємо ключове слово “`await`” перед викликом функції, це дозволяє зупиняти виконання коду, поки результат метода не буде повернуто, а інші операції в цей час можуть продовжуватись.

Контролер містить в собі такі методи як:

`signup()` – метод для реєстрації користувача, обробляє POST запит. Опишемо основні моменти коду: спочатку оброблюємо валідацію даних, яка перевірялась в роутері, використовуючи “`validationResult`” з пакету “`express-validator`”. Якщо при валідації даних не виникло помилок, тоді ми отримаємо дані з реєстрації – нікнейм, імейл, пароль. Далі перевіряємо наявність користувача з вхідним імейлом, він має бути унікальним, адже прикріплюється до користувача, за ним відбувається авторизація. У разі відсутності помилок, ми хешуємо пароль з допомогою пакету “`bcrypt`”, надаємо йому стандартну роль “`USER`”. Якщо користувач вирішив завантажити собі аватарку, то перевіряємо наявність файлу, вказуємо шлях до зображення і створюємо нового користувача, зберігаємо його дані в БД:

```

// SIGNUP
async signup(req, res) {

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const errors = validationResult(req);
if (!errors.isEmpty()) {
  return res
    .status(400)
    .json({ message: "Error with registration", errors });
}

const { username, email, password } = req.body;

let existingUser;
try {
  existingUser = await User.findOne({ email: email });
} catch (err) {
  return res
    .status(500)
    .json({ message: "Signing up failed, please try again later", err });
}

if (existingUser) {
  return res
    .status(422)
    .json({ message: "User exists already, email already exists" });
}

let hashedPassword;
let userRole;
try {
  hashedPassword = await bcrypt.hash(password, 10);
  userRole = await Role.findOne({ value: "USER" });
} catch (err) {
  return res
    .status(500)
    .json({ message: "Could not create user, please try again" });
}

let imagePath;
if (req.file) {
  imagePath = req.file.path;
} else {
  imagePath = null;
}

const newUser = new User({
  username,
  email,
  image: imagePath,
  password: hashedPassword,
  roles: [userRole.value],
});

try {

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        await newUser.save();
    } catch (err) {
        return res
            .status(500)
            .json({ message: "Signing up failed, please try again later ", err });
    }

    const token = generateAccessToken(newUser.id, newUser.email, newUser.roles);

    res.status(201).json({
        userId: newUser.id,
        username: newUser.username,
        email: newUser.email,
        password: newUser.password,
        token: token,
        role: newUser.roles[0],
    });
}

```

Лістинг.1.9. Метод для реєстрації нового користувача в контролері

Також в методі реєстрації ми генеруємо токен доступу за допомогою методу `generateAccessToken`, розглянемо це окремо.

Даний метод приймає інформацію про користувача - ідентифікатор, ім'я та роль і генерує JWT токен для автентифікації.

JWT (JSON Web Token) – стандарт для створення токенів, використовують для отримання доступу до захищених ресурсів на сервері [19].

На прикладі ігрового порталу, простими словами – після авторизації користувач отримує токен, який дає йому доступ до функцій та дій авторизованих користувачів.

Далі про метод, створюється об'єкт `payload`, містить дані про користувача, які будуть включені в токен, після чого ми використовуємо метод `Jwt.sign` для створення токена, підписуємо його секретним ключем, та вказуємо, що токен буде дійсним 1 годину:

```

const generateAccessToken = (userId, email, roles) => {
    try {
        const payload = { userId, email, roles };
        return Jwt.sign(payload, process.env.JWT_KEY, { expiresIn: "1h" });
    } catch (err) {
        throw new Error(err);
    }
}

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг.1.10. Метод для генерації токена

Перевіримо як працює реєстрація у Postman (додаток для тестування API, дозволяє легко і зручно виконувати різні типи запитів).

Для реєстрації оберемо метод POST, пропишемо адресу на якій працює сервер та в тілі запиту в JSON форматі пропишемо дані:

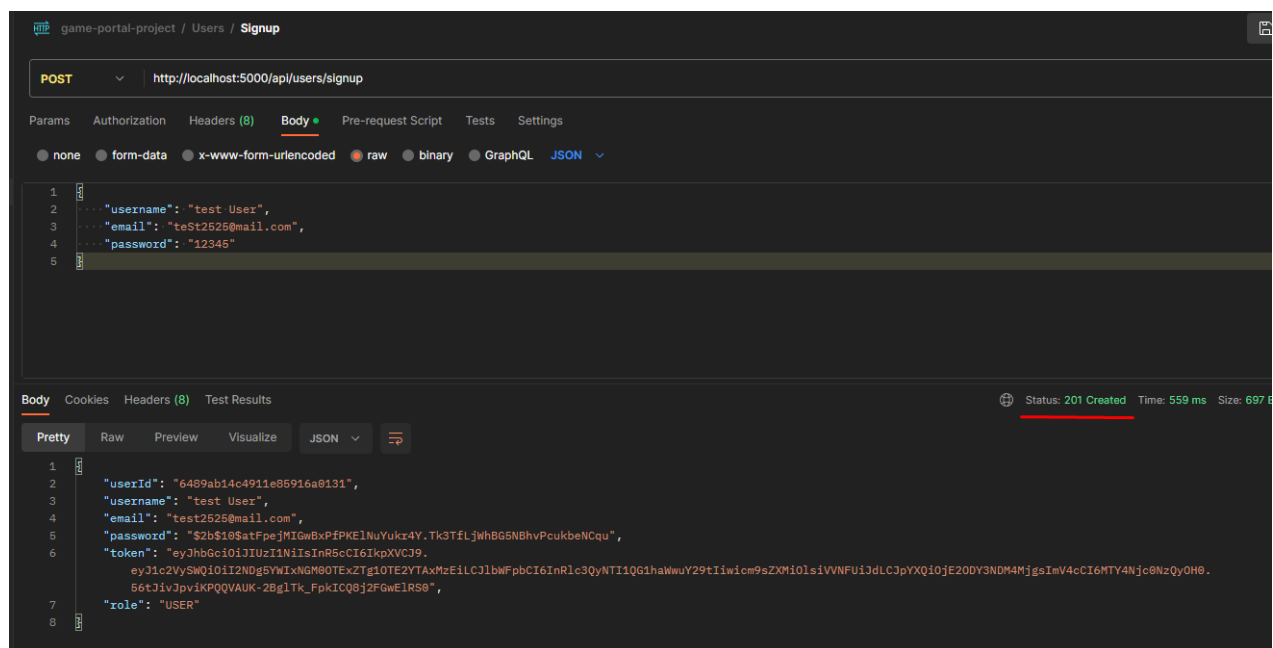


Рис.2.11. Перевірка реєстрації

Як бачимо, реєстрація користувача пройшла успішно, ми ввели валідні дані, отримало статус код 201 і сервер нам повернув дані про створеного користувача, захешований пароль та токен (при реєстрації надається токен, щоб на клієнті, користувач після реєстрації зайвий раз не авторизовувався).

Перевіримо як зреагує система бекенду, якщо ввести не валідні дані:

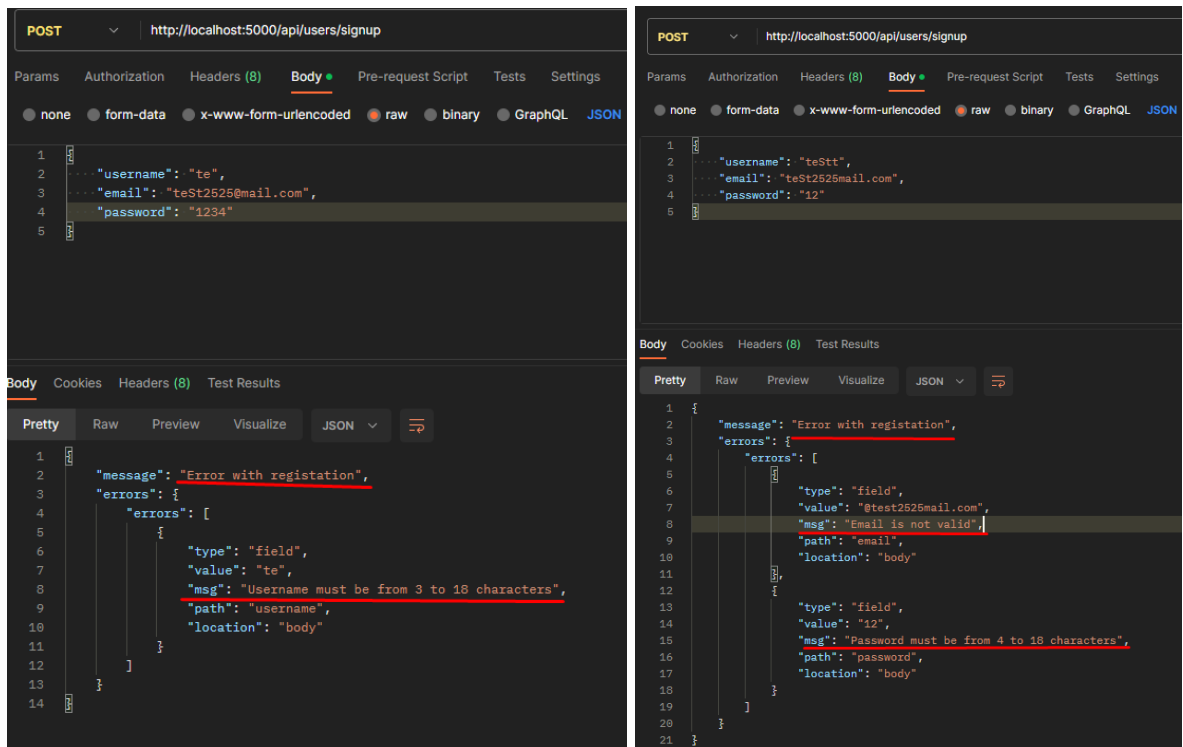


Рис.2.12. Перевірка введення помилкових даних

Отже валідація при реєстрації, зроблена за допомогою пакету “express-validator” працює коректно.

Ми створили нового користувача, тепер розглянемо метод авторизації в контролері UserController:

```
_id: ObjectId('6489ab14c4911e85916a0131')
username: "test User"
email: "test2525@mail.com"
password: "$2b$10$atFpejMIGwBxPfpKElNuYukr4Y.Tk3TfLjWhBG5NBhvPcukbeNCqu"
image: null
roles: Array
  0: "USER"
posts: Array
  __v: 0
```

Рис.2.13. Новий користувач в MongoDB

Опишемо моменти з коду. Спочатку приймаємо дані, потім перевіряємо по пошті чи такий користувач є в базі даних, якщо немає, то повертаємо помилку, якщо ж існує ідемо далі. Наступним кроком ми перевіряємо пароль, порівнюємо

його з інсуючим за допомогою пакету “bcrypt”. Якщо пароль вірний, то генеруємо токен по даним існуючого користувача та повертаємо потрібні дані:

```
// LOGIN
async login(req, res) {
  const { email, password } = req.body;

  let existingUser;
  try {
    existingUser = await User.findOne({ email: email });
  } catch (err) {
    return res
      .status(500)
      .json({ message: "Logging in failed, please try again later", err });
  }

  if (!existingUser) {
    return res.status(403).json({
      message: `Invalid credentials, user with ${email} not found, could not log
you in.`,
    });
  }

  let isValidPassword = false;
  try {
    isValidPassword = await bcrypt.compare(password, existingUser.password);
  } catch (err) {
    return res.status(500).json({
      message:
        "Could not log you in, please check your credentials and try again",
    });
  }

  if (!isValidPassword) {
    return res.status(403).json({
      message:
        "Invalid credentials, password not correct, could not log you in",
    });
  }

  const token = generateAccessToken(
    existingUser.id,
    existingUser.email,
    existingUser.roles
  );

  res.status(201).json({
    userId: existingUser.id,
    username: existingUser.username,
    email: existingUser.email,
```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    token: token,
    role: existingUser.roles[0],
  });
}

```

Лістинг.1.11. Метод для авторизації користувача

Перевіримо авторизацію в постмані:

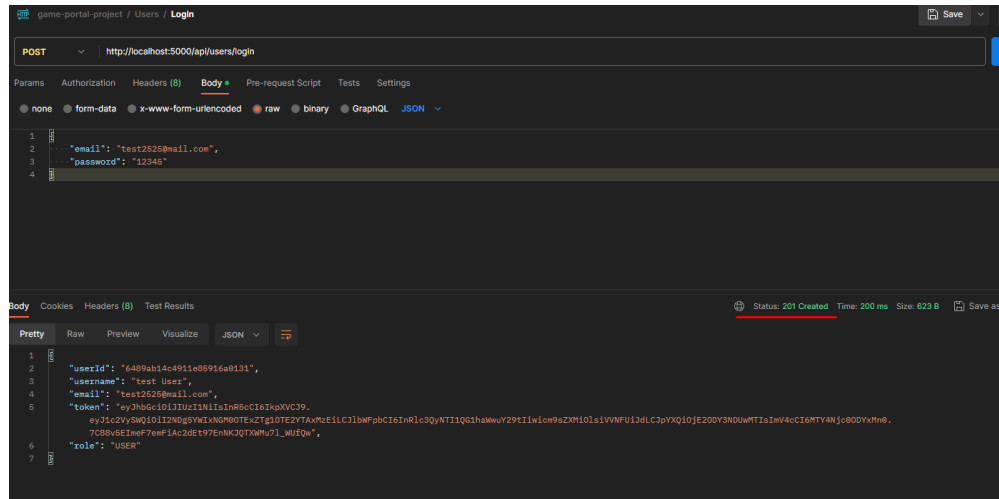


Рис.2.14. Успішна авторизація

Перевіримо спробу авторизацію з помилковим паролем або імейлом:

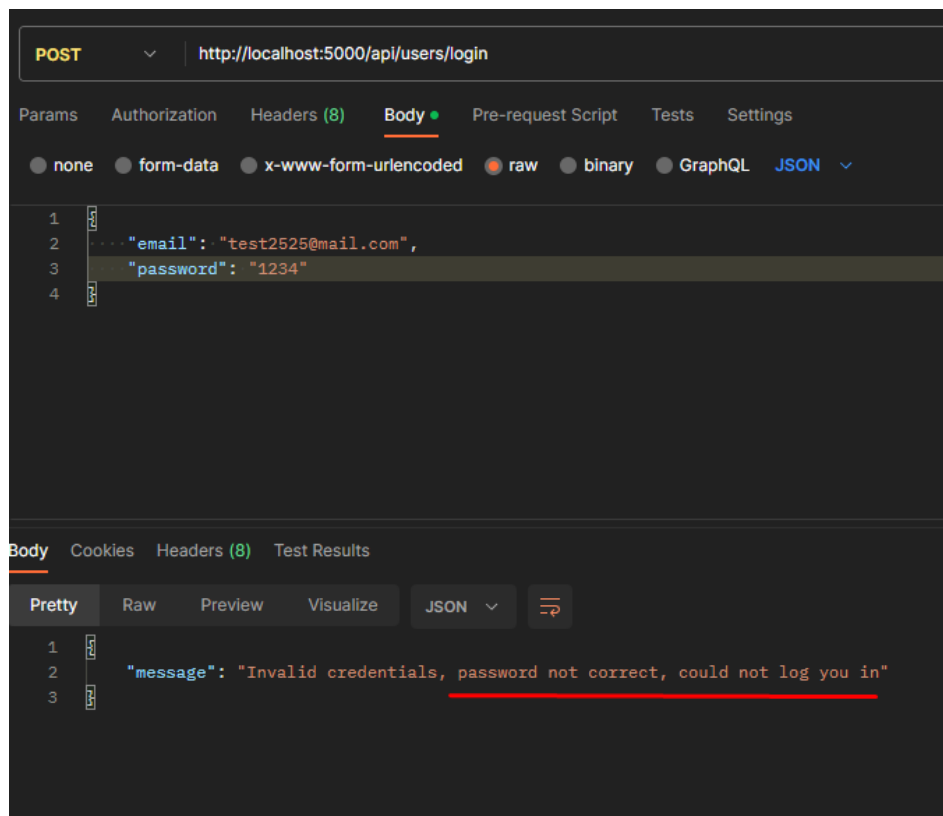


Рис.2.15. Помилка при авторизації, введення невірному паролю

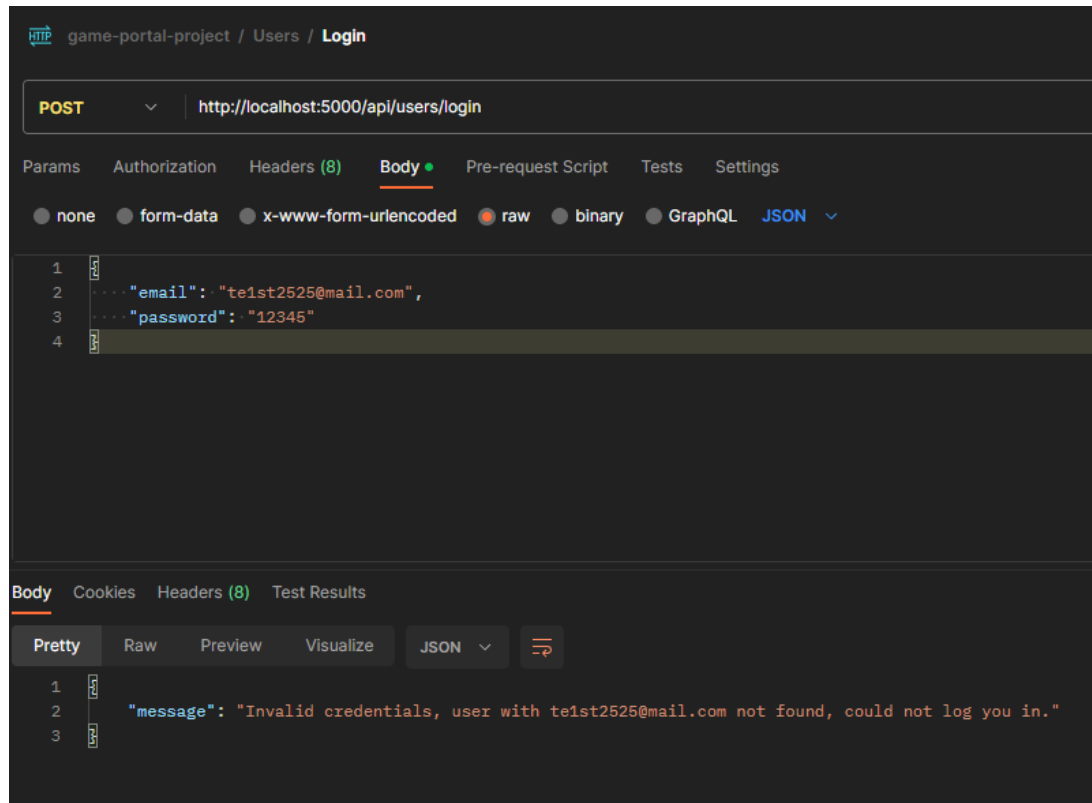


Рис.2.16. Помилка при авторизації, введення невірної пошти

Розглянемо також GET методи, які використовуються, а саме метод для отримання всіх користувачів та окремо одного за його ідентифікатором. Вони досить схожі, здійснюється пошук у БД, якщо вказуємо пустий об'єкт, то отримаємо всіх користувачів, якщо вкажемо метод, що шукаємо по ідентифікатору, то знайдемо відповідного користувача:

```
// GET ALL USERS
async getUsers(req, res) {
  try {
    const users = await User.find({}, "-password");
    res.status(200).json(users);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
}

// GET USER BY ID
async getUserById(req, res) {
  try {
    const userId = req.params.uid;
    const user = await User.findById(userId);

    res.status(200).json({
```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        userId: user.id,
        username: user.username,
        posts: user.posts,
        image: user.image || null,
    });
} catch (err) {
    res.status(404).json({ message: err.message });
}
}

```

Лістинг.1.12. Метод для отримання даних про користувачів та користувача за його ідентифікатором

Також в проєкті використовується middleware для обробки ролі користувача. Це зроблено для того, щоб забезпечити обмежений доступ до конкретних запитів. Наприклад в роутері, щоб отримати список всіх користувачів треба мати роль адміна або власника:

```
router.get("/", checkRole(["ADMIN", "OWNER"]), UserController.getUsers);
```

Лістинг.1.13. Використання middleware для перевірки ролі

Тепер розглянемо саму middleware check-role.js. В проєкті також присутній check-auth.js, по своїй структурі вони досить схожі, але є певні відмінності. Для перевірки ролі, нам спочатку треба її отримати, після чого використовуємо принцип замикання в js, таким чином повертаємо функцію в тілі іншої функції, яка вже приймає запит [18]. Спочатку перевіряємо метод запиту, якщо це “OPTIONS”, то пропускаємо і йдемо далі. Наступним кроком ми дістаємо з заголовку сам токен, декодуємо його з використанням секретного ключа та витягуємо ролі. Якщо користувач має одну з вказаних ролей, то викликаємо next(), таким чином передаємо доступ далі. Перевірка токена та ролей супроводжуються обробкою помилок:

```

import Jwt from "jsonwebtoken";

export const checkRole = (roles) => {
    return function (req, res, next) {
        if (req.method === "OPTIONS") {
            next();
        }

        try {

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const token = req.headers.authorization.split(" ")[1];
if (!token) {
  return res.status(403).json({ message: "User is not authorized" });
}

const { roles: userRoles } = Jwt.verify(token, process.env.JWT_KEY);
let hasRole = false;
userRoles.forEach((role) => {
  if (roles.includes(role)) {
    hasRole = true;
  }
});

if (!hasRole) {
  return res.status(403).json({ message: "You don't have access" });
}
next();
} catch (err) {
  return res.status(403).json({ message: "User is not authorized" });
}
};
};
};

```

Лістинг.1.14. Middleware для перевірки ролі

Тож, тепер перевіримо як працюють методи `getUsers` та `getUserById`. Пам'ятаємо, що для отримання всіх користувачів, треба мати роль адміністратора або власника:

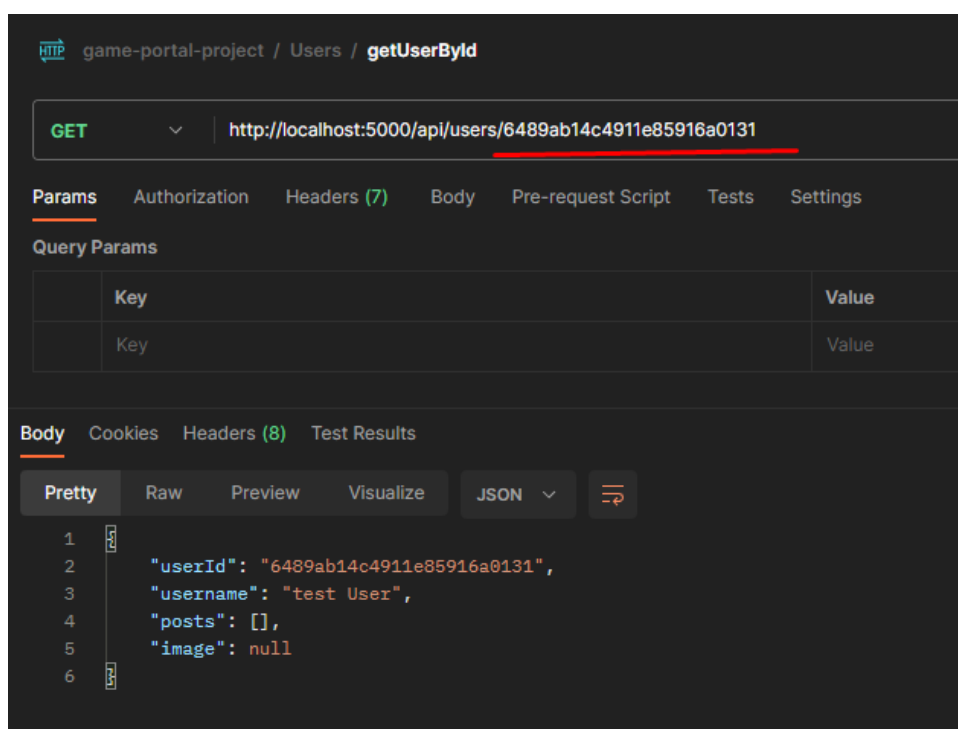


Рис.2.17. Перевірка отримання даних про користувача за ідентифікатором

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

Для отримання списку всіх користувачів ми повинні передати значення токена в хедері “Authorization”:

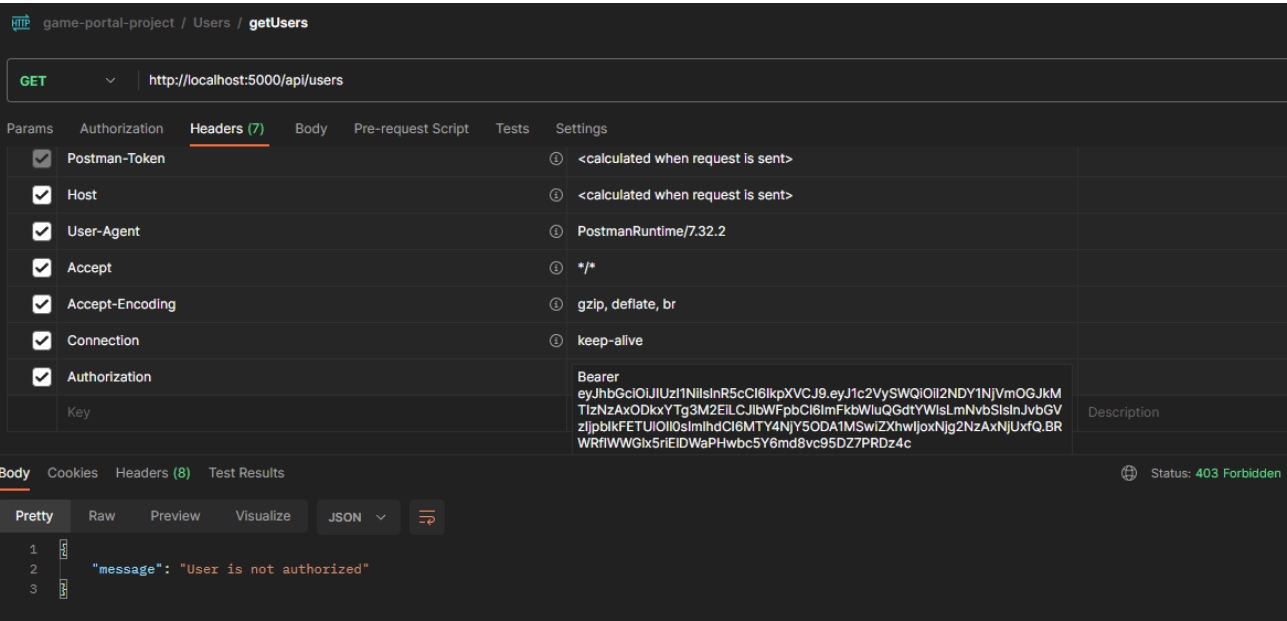


Рис.2.18. Перевірка отримання списку користувачів, введено недійсний токен

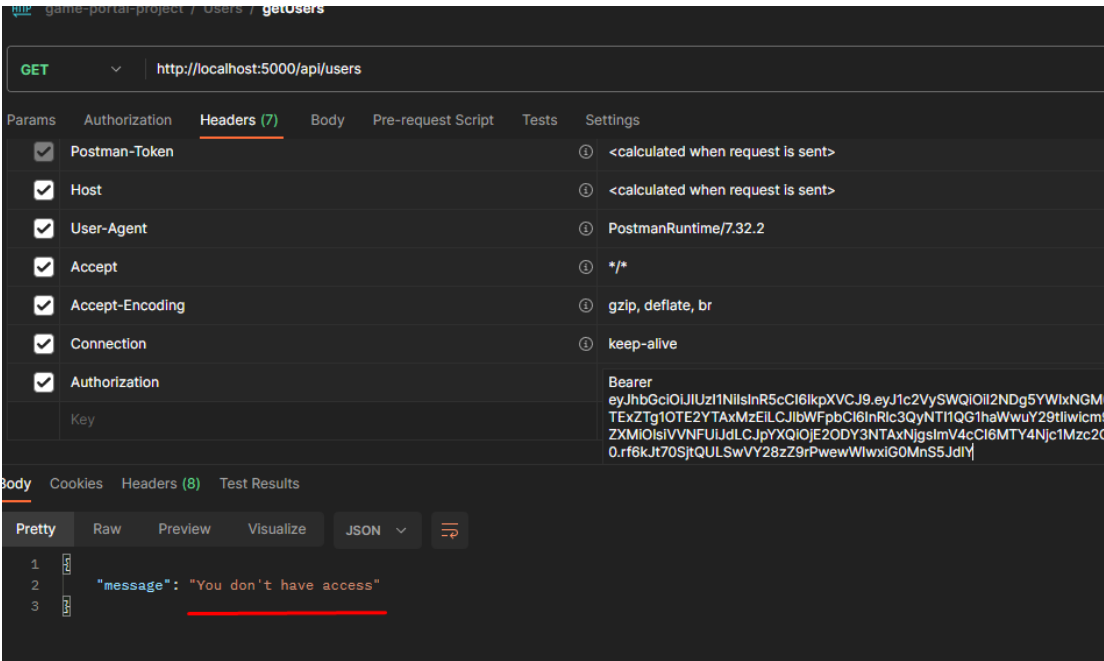


Рис.2.19. Перевірка отримання списку користувачів, введено токен звичайного користувача


```

const pageSize = limit ? parseInt(limit) : 0;
const pageNumber = page ? parseInt(page) : 0;

let user;
let comments;
try {
  user = await User.findById(userId);

  comments = await Comment.find({ creatorId: userId })
    .sort({ date: -1 })
    .limit(pageSize)
    .skip(pageSize * pageNumber);

  const totalCount = await Comment.countDocuments({ creatorId: userId });
  res.header("Access-Control-Expose-Headers", "X-Total-Count");
  res.header("X-Total-Count", totalCount);
} catch (err) {
  return res.status(500).json({ message: err.message });
}

res.status(200).json({
  userId: user.id,
  email: user.email,
  username: user.username,
  image: user.image || null,
  comments: comments,
});
}

```

Лістинг.1.15. Метод для отримання даних для профілю користувача,
використання параметрів запити

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

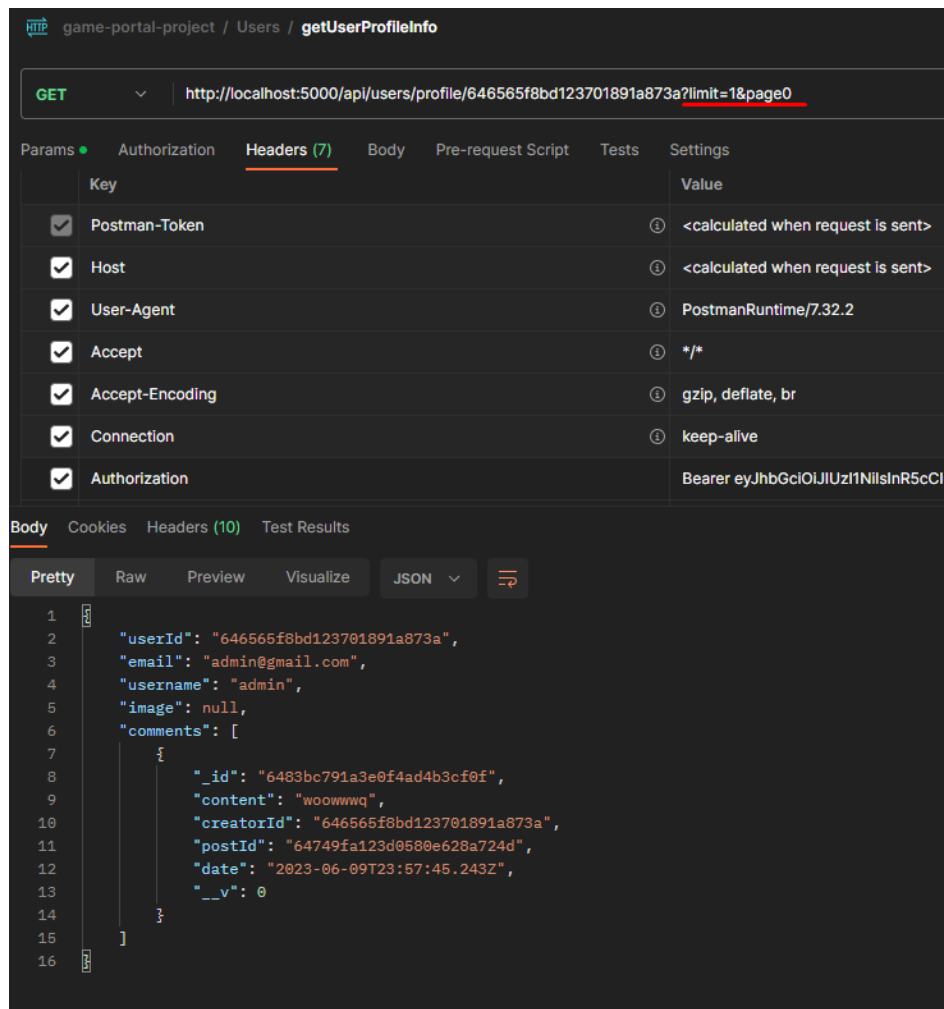


Рис.2.21. Вивід даних по сторінках

Key		Value
X-Powered-By	i	Express
Access-Control-Allow-Origin	i	*
Access-Control-Expose-Headers	i	X-Total-Count
X-Total-Count	i	4
Content-Type	i	application/json; charset=utf-8
Content-Length	i	280

Рис.2.22. Загальна кількість даних

Тепер розглянемо як будуються методи для оновлення даних. Зробимо це на прикладі оновлення даних про публікацію, це метод `updatePost`, з контролера `posts-controller.js`. Оснвоный принцип – отримуємо дані, якщо вони валідні,

якщо намагається оновити користувач – автор поста, то ми оновлюємо дані в базі за допомогою метода PATCH. Також присутня обробка помилок та визначення, чи користувач завантажив нове зображення, якщо так, то встановлюємо нове, попереднє видаляємо, якщо ні – залишаємо те, яке є:

```
// UPDATE
async updatePost(req, res) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(422).json({
      message: "Invalid inputs passed, please check your data",
      errors,
    });
  }

  const { title, rubric, description, content } = req.body;
  const postId = req.params.pid;

  let post;
  try {
    post = await Post.findById(postId);
  } catch (err) {
    return res
      .status(500)
      .json({ message: "Something went wrong, could not find a post", err });
  }

  if (post.creator.toString() !== req.userData.userId) {
    return res
      .status(403)
      .json({ message: "You are not allowed to edit this post" });
  }

  if (req.file) {
    post.image &&
      fs.unlink(post.image, (err) => {
        if (err) {
          console.error("Failed to delete image:", err);
        }
      });
  }

  let imagePath;
  if (req.file) {
    imagePath = req.file.path;
  } else {
    imagePath = post.image;
  }
}
```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

Спробуємо оновити дані, залогінившись під автором поста:

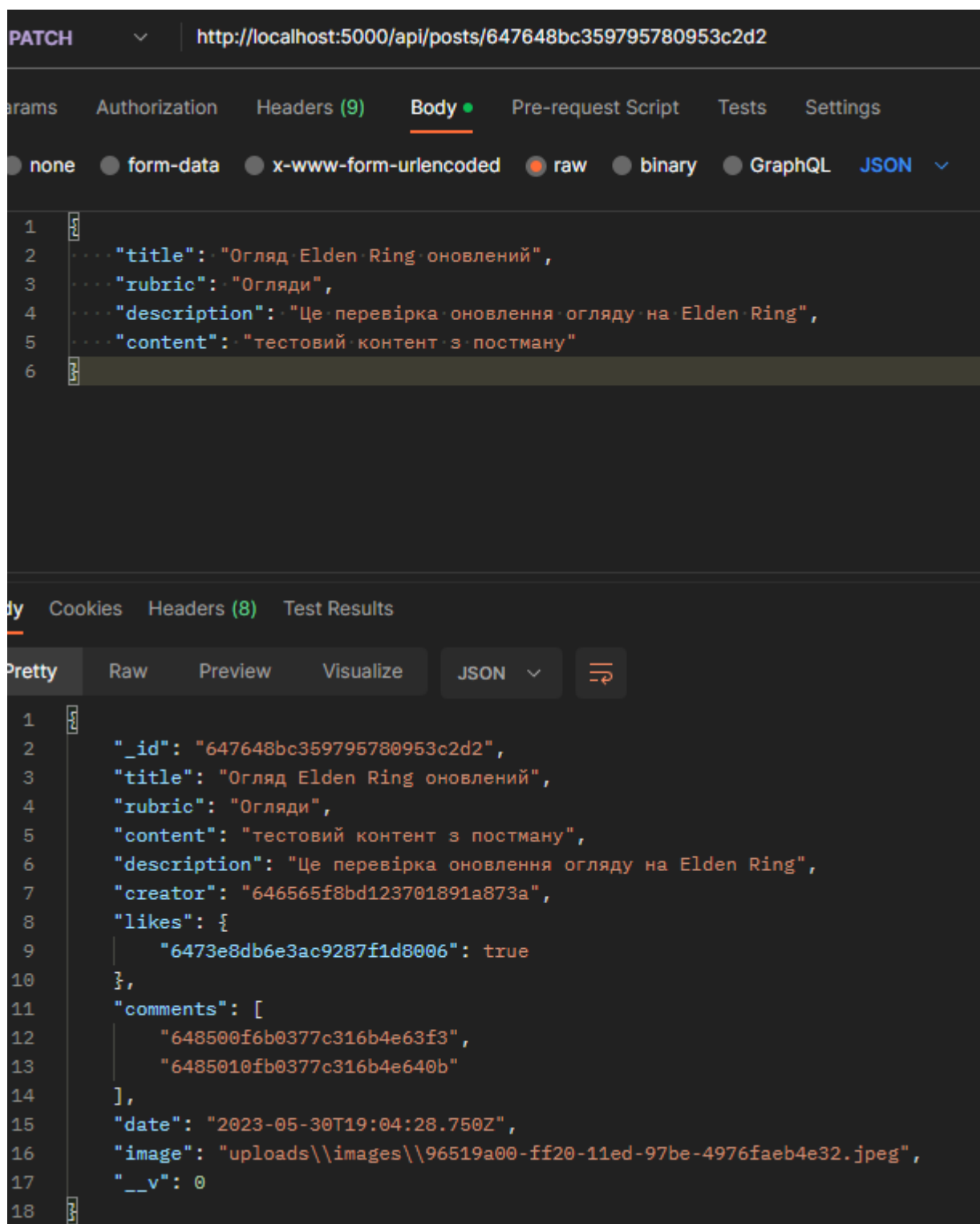


Рис.2.24. Оновлення даних публікації

Як бачимо, дані оновились, зображення залишилось попереднє, оскільки ми його і не оновлювали. Також всі інші дані про лайки та коментарі зберіглись.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

Видалення даних, розглянемо як це зроблено у контролері comments-controller.js., за це відповідає метод deleteComment. Шукаємо в базі коментар за ідентифікатором, якщо такий існує, якщо авторизований користувач це автор коментаря, то видаляємо його. Також треба оновити список коментарів у відповідної публікації, це ми також робимо за ідентифікатором, видаляємо коментар у масиві коментарів у публікації, за допомогою оператора \$pull:

```
async deleteComment(req, res) {
  const commentId = req.params.cid;

  let comment;
  try {
    comment = await Comment.findById(commentId);

    if (!comment) {
      return res
        .status(404)
        .json({ message: "Could not find comment for this id" });
    }

    if (comment.creatorId.toString() !== req.userData.userId) {
      return res
        .status(403)
        .json({ message: "You are not allowed to delete this comment" });
    }

    comment = await Comment.findByIdAndDelete(commentId);

    await Post.findByIdAndUpdate(comment.postId, {
      $pull: { comments: commentId },
    });
  } catch (err) {
    return res
      .status(500)
      .json({ message: "Something went wrong, could not find a comment" });
  }

  res.status(200).json({ message: "Comment deleted successfully" });
}
```

Лістинг.1.17. Метод для видалення коментаря

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

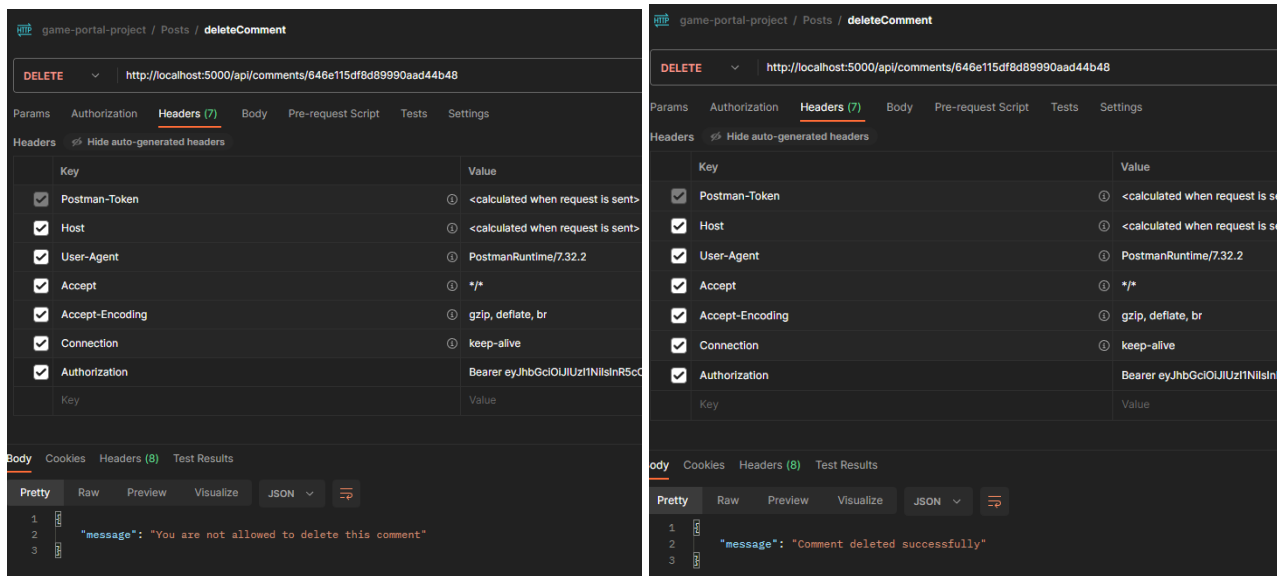


Рис.2.25. Видалення коментаря, спроба видалення авторизувавшись не як автор коментаря; успішна спроба видалення.

В проєкті також використовуються декілька важливих middleware, які варто розглянути.

Перевірка авторизації – постійно використовується, за структурою схожа на middleware перевірки ролі користувача. Отримуємо токен, декодуємо, якщо проходимо верифікацію, то надаємо доступ:

```
import Jwt from "jsonwebtoken";

export const checkAuth = (req, res, next) => {
  if (req.method === "OPTIONS") {
    next();
  }

  try {
    let token;
    if (req.headers.authorization) {
      token = req.headers.authorization.split(" ")[1];

      if (!token) {
        return res.status(403).json({ message: "User is not authorized" });
      }

      const decodedToken = Jwt.verify(token, process.env.JWT_KEY);
      req.userData = { userId: decodedToken.userId };
      next();
    } catch (err) {
```

```

    return res.status(403).json({ message: "User is not authorized" });
  }
};

```

Лістинг.1.18. Middleware для перевірки авторизації користувача для надання доступу

І ще одна, це middleware file-upload.js. Тут використовуємо пакет “multer”, він надає можливість обробляти і завантажувати файли, за налаштуваннями, дозволяє зберігати файли на диску і фільтрувати за типом MIME [20].

Об’єкт MIME_TYPE_MAP використовується для збереження зображень з унікальними іменами, також визначаємо, що файли – це зображення з відповідними розширеннями.

В самій функції fileUpload, визначаємо розмір файлів та конфігуруємо сховище зображень: шлях, назву та фільтруємо відповідно до об’єкта MIME_TYPE_MAP:

```

import multer from "multer";
import { v1 as uuidv1 } from "uuid";

const MIME_TYPE_MAP = {
  "image/png": "png",
  "image/jpeg": "jpeg",
  "image/jpg": "jpg",
};

export const fileUpload = multer({
  limits: 500000,
  storage: multer.diskStorage({
    destination: (req, file, cb) => {
      cb(null, "uploads/images");
    },
    filename: (req, file, cb) => {
      const ext = MIME_TYPE_MAP[file.mimetype];
      cb(null, uuidv1() + "." + ext);
    },
  }),
  fileFilter: (req, file, cb) => {
    const isValid = !!MIME_TYPE_MAP[file.mimetype];
    let error = isValid ? null : new Error("Invalid mime type!");
    cb(error, isValid);
  },
});

```

Лістинг.1.19. Middleware для завантаження зображень

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

Отже, ми розглянули всі ключові методи для роботи з базою даних, для отримання даних з параметрами запиту та без, для їх створення, редагування та видалення. Були наведені приклади роботи деяких таких методів у контролерах, інші будуються схожим чином, звичайно різниця буде вже залежати від моделі та конкретних вимог.

Висновки до другого розділу

Під час виконання другого розділу було пройдено чимало моментів розробки вебсайту, пророблено етапи проектування та створення серверної частини веб-додатку.

По початковому баченні сайту, оснований на вимогах та критеріях з першого розділу кваліфікаційної роботи, були побудовані UML діаграми, а саме діаграма варіантів використання, яка покажує можливі дії користувачів, діаграма діяльності, яка зображує алгоритми роботи активностей в системі.

Була побудована діаграми бази даних, визначені залежності між колекціями, типи полів. Були визначені компоненти системи та зроблена відповідна діаграма.

Далі була проведене масштабна робота по побудові повноцінного сервера вебсайту на основі проектування, за допомогою засобів розробки Node.js в поєднання з Express.js. У висновку отримали пророблену бекенд частину, яка готова до взаємодії з фронтендом.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ПОБУДОВА ІНТЕРФЕЙСУ, ПОРЯДОК РОБОТИ З ВЕБСАЙТОМ, ТЕСТУВАННЯ

3.1 Реалізація користувацького інтерфейсу, побудова React-застосунку

Маючи готову серверну частину можна приступити до розробки фронтенду. В межах кваліфікаційної роботи розробки ігрового порталу створення користувацького інтерфейсу буде займати найбільше часу, проте використовуючи бібліотеку React це буде досить зручний і надійний процес розробки.

Основний підхід в розробці реакт застосунку – побудова системи на основі функціональних компонентів, які повторно можуть використовуватись та складати разом складні та швидкі інтерфейси. Один з ключових принципів це розділення відповідальності, таким чином проект буде зручно масштабуватись і в цілому структура проекту буде зрозумілою та чіткою.

Реакт додаток ігрового порталу буде включати в себе велику кількість компонентів, це можуть бути як і частини зі своєю логікою так і UI компоненти, сторінки, хуки, утиліти, стилі тощо.

Зупинимось на ключових моментах, які грають фундаментальну роль у побудові якісного фронтенду.

Перше це маршрутизація на клієнті. Для виконання даної задачі будемо використовувати бібліотеку React Router DOM – дозволяє створювати навігацію та маршрутизацію, зберігаючи принцип SPA. Основна ціль – забезпечити перехід між різними сторінками без їх перезавантаження.

Щоб це зробити, імпортуємо з бібліотеки react-router-dom функцію `createBrowserRouter` – для створення роутера та компонент `RouterProvider` щоб передати створений роутер у кореневий компонент, щоб надати доступ до маршрутизації усім іншим компонентам. Компонент `Navigate` – для редіректу, в нашому випадку на головну сторінку, якщо сторінка має відображатись тільки для конкретної ролі користувача.

Створюємо роутер та вказуємо батьківський елемент, щоб забезпечити компонент `Layout`, який містить в собі навігаційну панель й футер, та сторінки

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

помилки. Це зроблено таким чином, щоб на всіх сторінках мати навігацію та у разі помилки `ErrorPage`. Далі, як дочірні елементи вказуємо всі сторінки. Об'єкт сторінки містить шлях та відповідний компонент у елементі:

```
const router = createBrowserRouter([
  {
    element: <Layout />,
    errorElement: <ErrorPage />,
    children: [
      {
        path: "/",
        element: <HomePage />,
      },
      {
        path: "/new-post",
        element: role === "ADMIN" ? <NewPostPage /> : <Navigate to="/" />,
      },
      {
        path: "/add-game",
        element: role === "ADMIN" ? <NewGamePage /> : <Navigate to="/" />,
      },
      // ----- Пропуск, тут ще сторінки

      {
        path: "/:userId/posts",
        element: <UserPostsPage />,
      },
      {
        path: "/update-posts/:postId",
        element: <UpdatePostPage />,
      },
      {
        path: "/update-game/:gameId",
        element: <UpdateGamePage />,
      },
      {
        path: "/profile/:uid",
        element: token ? <UserProfilePage /> : <Navigate to="/" />,
      },
    ],
  },
]);
```

Лістинг.2.1. Налаштування маршрутизації

Далі передаємо роутер у провайдер, який повертає кореневий компонент `App.jsx`:

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

```
<RouterProvider router={router}></RouterProvider>
```

Лістинг.2.2. Передача роутера у провайдер

В роутері ми перевіряли роль користувача за допомогою стану `role`. Ми її беремо з хука `useAuth`. Розглянемо даний хук, він використовується для управління автентифікацією та зберігання даних користувача під час його сесії. Тут ми створюємо стани токена, стан про збереження дати закінчення дії токена ролі та ідентифікатора користувача, які використовуємо в функціях `login`(авторизації користувача) та `logout`(виходу користувача). В них ми заносимо дані у стейти, а також додаємо дані в локальне сховище або у разі виходу очищаємо локальне сховище. Маємо також два використання хука `useEffect`, в першому встановлюємо таймер виходу, зважаючи на термін дії токена, коли токен або термін дії змінюється, тоді таймер виходу буде встановлено або завершено. В другому хуці `useEffect`, при монтуванні, перевіряємо чи є збережені дані в локальному сховищі, якщо є, то викликаємо функцію `login` для відновлення автентифікації користувача зі збереженими даними. Простими словами, якщо користувач залогінився і не вийшов, то при перезавантаженні сторінки він всеодно буде авторизований, поки діє токен:

```
import { useState, useCallback, useEffect } from "react";

let logoutTimer;

export const useAuth = () => {
  const [token, setToken] = useState(false);
  const [role, setRole] = useState(false);
  const [tokenExpirationDate, setTokenExpirationDate] = useState();
  const [userId, setUserId] = useState(false);

  const login = useCallback((uid, token, role, expirationDate) => {
    setToken(token);
    setUserId(uid);
    setRole(role);
    const tokenExpirationDate =
      expirationDate || new Date(new Date().getTime() + 1000 * 60 * 60);
    setTokenExpirationDate(tokenExpirationDate);
    localStorage.setItem(
      "userData",
      JSON.stringify({
        userId: uid,
```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        token: token,
        role: role,
        expiration: tokenExpirationDate.toISOString(),
    })
    );
}, []);

const logout = useCallback(() => {
    setToken(null);
    setTokenExpirationDate(null);
    setRole(null);
    setUserId(null);
    localStorage.removeItem("userData");
}, []);

useEffect(() => {
    if (token && tokenExpirationDate) {
        const remainingTime =
            tokenExpirationDate.getTime() - new Date().getTime();
        logoutTimer = setTimeout(logout, remainingTime);
    } else {
        clearTimeout(logoutTimer);
    }
}, [token, logout, tokenExpirationDate]);

useEffect(() => {
    const storedData = JSON.parse(localStorage.getItem("userData"));
    if (
        storedData &&
        storedData.token &&
        new Date(storedData.expiration) > new Date()
    ) {
        login(
            storedData.userId,
            storedData.token,
            storedData.role,
            new Date(storedData.expiration)
        );
    }
}, [login]);

return { token, login, logout, userId, role };
};

```

Лістинг.2.3. Хук для управління автентифікацією та зберігання даних сесії користувача

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

Також для уникнення пропс дрілінгу, будемо використовувати контекст, що дозволить передавати дані безпосередню на пряму до компонентів, без необхідності передавати їх через пропси всіх проміжних компонентів.

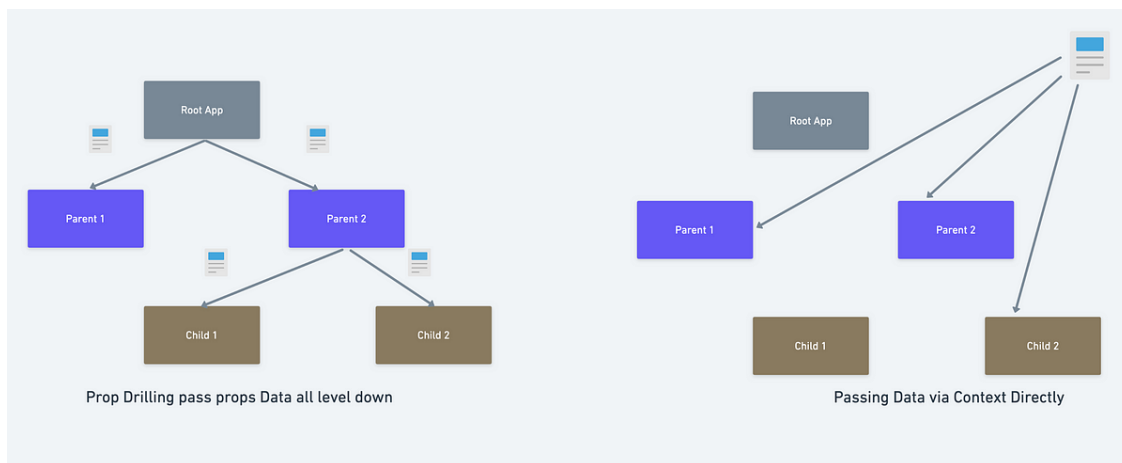


Рис.3.1. Наглядний приклад принципу роботи контексту в реакті

Імортуюмо функцію `createContext` та використовуємо її для створення контексту `AuthContext`:

```
import { createContext } from "react";

export const AuthContext = createContext({
  isLoggedIn: false,
  userId: null,
  token: null,
  role: null,
  login: () => {},
  logout: () => {},
});
```

Лістинг.2.4. Створення контексту

В кореневому компоненті `App.jsx`, обертаємо все в `AuthContext.Provider` для надання значень контексту усім компонентам, які знаходяться всередині нього:

```
return (
  <AuthContext.Provider
    value={{
      isLoggedIn: !!token,
      token: token,
      role: role,
      userId: userId,
      login: login,
```

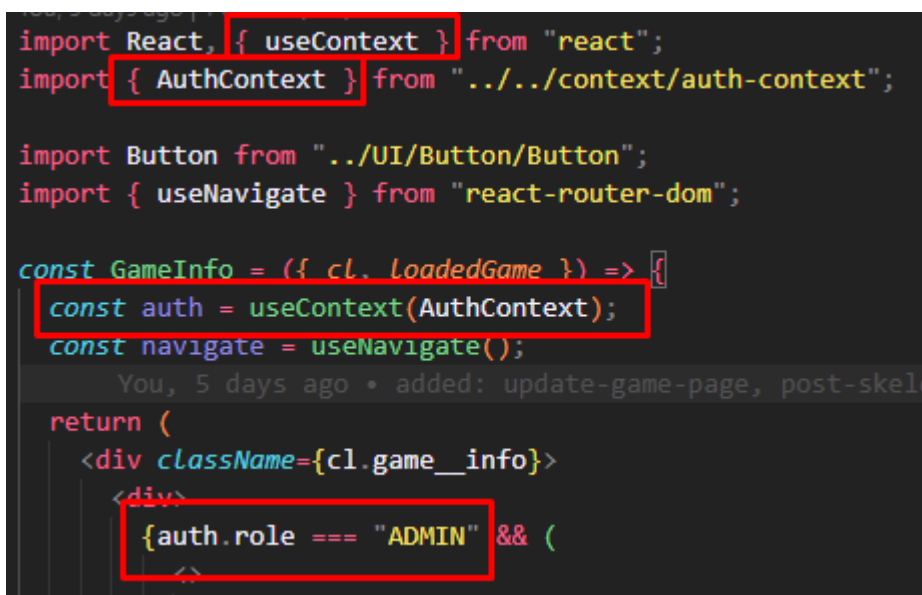
```

      logout: logout,
    }}
  >
  <RouterProvider router={router}></RouterProvider>
</AuthContext.Provider>
);

```

Лістинг.2.5. Використання контекст провайдера

Тож тепер ми можемо імпортувати контекст в будь якому компоненті та використовувати потрібні значення, при цьому уникаємо пропс дрілінгу:



```

import React, { useContext } from "react";
import { AuthContext } from "../../context/auth-context";

import Button from "../UI/Button/Button";
import { useNavigate } from "react-router-dom";

const GameInfo = ({ cl, LoadedGame }) => {
  const auth = useContext(AuthContext);
  const navigate = useNavigate();

  You, 5 days ago • added: update-game-page, post-skele

  return (
    <div className={cl.game__info}>
      <div>
        {auth.role === "ADMIN" && (

```

Рис.3.2. Приклад використання контексту в компоненті

Важливим моментом є отримання даних з сервера. Розглянемо типовий приклад, як це можна зробити у реакт компоненті, на прикладі функції fetchPosts, яка виконує запит на сервер для отримання даних про новини. Ключовий момент це використання асинхронного запиту за допомогою fetch до відповідного URL, отриману відповідь конвертуємо в JSON формат:

```

const fetchPosts = async (limit = 0, page = 0) => {
  try {
    setIsLoading(true);
    const response = await fetch(
      `${process.env.REACT_APP_API_URL}/posts/news?limit=${limit}&page=${page}`
    );

    const responseData = await response.json();

    if (!response.ok) {

```

```

        console.log(responseData.message);
    }

    const totalCount = response.headers.get("X-Total-Count");
    setTotalPages(getPageCount(totalCount, limit));

    setLoadedPosts(responseData);
    setIsLoading(false);
  } catch (err) {
    console.log(err);
    setIsLoading(false);
  }
};

```

Лістинг.2.6. Метод для отримання даних з сервера

Якщо ж нам треба скористатись методом POST, PATCH, DELETE, то в методі fetch додаємо об'єкт, який містить назву метода запиту, тіло, хедери та сигнал, якщо це потрібно.

Останній важливий момент який слід розглянути – це валідація на клієнті, може використовуватись на сторінці авторизації\реєстрації, на сторінках створення або редагування даних. Для її обробки було створено хук useForm, де ми керуємо станом форми. Використовуємо formReducer – визначає зміни стану, коли “INPUT_CHANGE” або “SET_DATA” повертають новий стан. Кастомний хук useForm, використовує useReducer для управління станом форми за допомогою диспетчера - dispatch (викликається для виконання дії - action). В результаті, хук повертає formState - стан зі значенням та значенням валідності, функцію inputHandler – яка буде відслідковувати події введення на полі форми та функцію setFormData - для встановлення даних на всій формі одночасно:

```

import { useCallback, useReducer } from "react";

const INPUT_CHANGE = "INPUT_CHANGE";
const SET_DATA = "SET_DATA";

const formReducer = (state, action) => {
  switch (action.type) {
    case INPUT_CHANGE:
      let formIsValid = true;
      for (const inputId in state.inputs) {
        if (!state.inputs[inputId]) {
          continue;
        }
      }

```

```

    }
    if (inputId === action.inputId) {
      formIsValid = formIsValid && action.isValid;
    } else {
      formIsValid = formIsValid && state.inputs[inputId].isValid;
    }
  }
  return {
    ...state,
    inputs: {
      ...state.inputs,
      [action.inputId]: { value: action.value, isValid: action.isValid },
    },
    isValid: formIsValid,
  };
case SET_DATA:
  return {
    inputs: action.inputs,
    isValid: action.formIsValid,
  };
default:
  return state;
}
};

export const useForm = (initialInputs, initialFormValidity) => {
  const [formState, dispatch] = useReducer(formReducer, {
    inputs: initialInputs,
    isValid: initialFormValidity,
  });

  const inputHandler = useCallback((id, value, isValid) => {
    dispatch({
      type: INPUT_CHANGE,
      value: value,
      isValid: isValid,
      inputId: id,
    });
  }, []);

  const setFormData = useCallback((inputData, formValidity) => {
    dispatch({
      type: SET_DATA,
      inputs: inputData,
      formIsValid: formValidity,
    });
  }, []);

  return [formState, inputHandler, setFormData];
};

```

Лістинг.2.7. Хук для валідації

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

Приклад використання хука на сторінці авторизації\реєстрації:

```
const [formState, inputHandler, setFormData] = useForm(  
  {  
    email: {  
      value: "",  
      isValid: false,  
    },  
    password: {  
      value: "",  
      isValid: false,  
    },  
  },  
  false  
);
```

Лістинг.2.8. Використання хука для валідації

Таким чином ми розглянули основні важливі моменти, які грають основну роль при побудові реакт застосунку.

3.2 Огляд роботи, інтерфейс вебсторінки

Опишемо роботу з додатком, коли маємо вже готовий фронтенд та бекенд.

Після завантаження вебсайту бачимо домашню\головну сторінку. В хедері бачимо навігаційно панель, вона знаходиться на всіх сторінках, адже є в компонентів Layout. На головній сторінці є карусель популярний постів за певний період (на рисунку зображені публікації з найбільшою кількістю лайків за останній місяць):

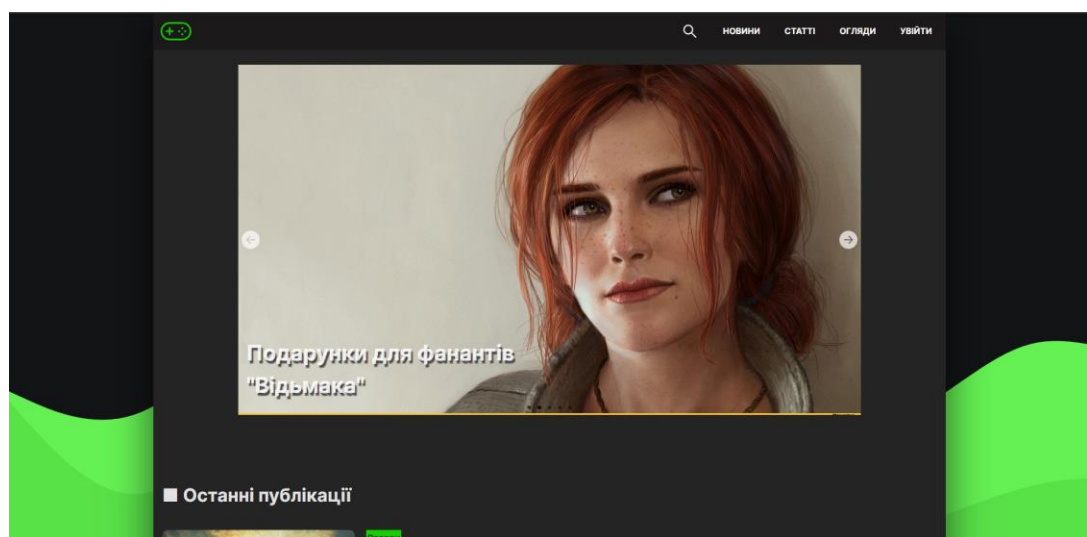


Рис.3.3. Домашня сторінка вебсайту, карусель зображень

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

Також маємо на головній сторінці останні публікації. Тут задля прикладу виставлений ліміт в 3 елемента на сторінку, тобто маємо тут пагінацію:

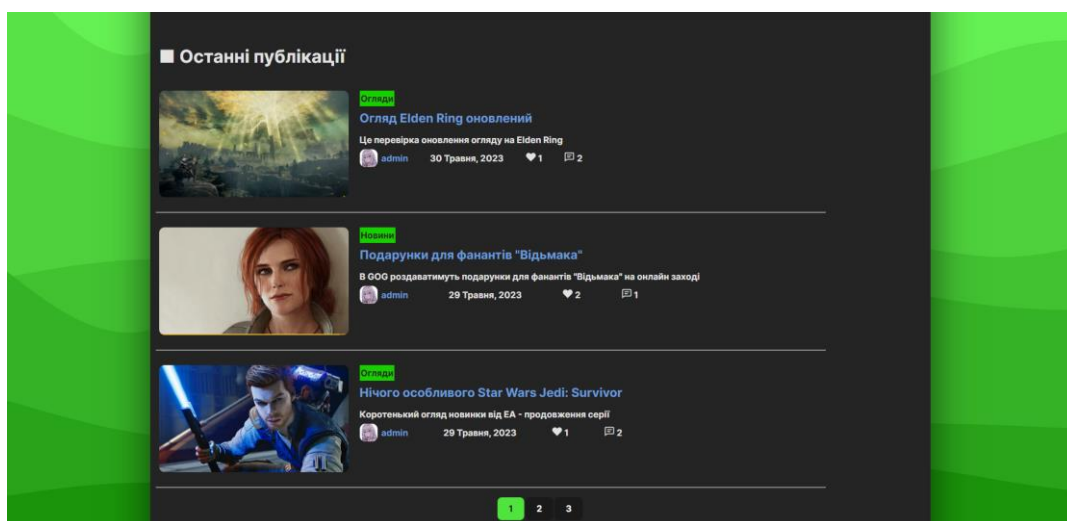


Рис.3.4. Домашня сторінка вебсайту, останні публікації

Скористаємось навігаційною панеллю, перейдемо до сторінки оглядів, де присутні всі публікації за категорією огляд:

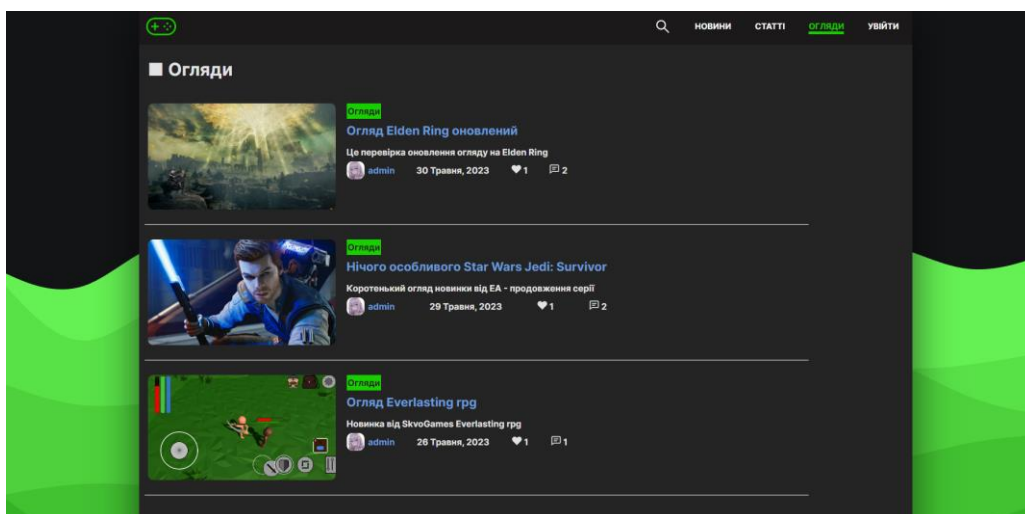


Рис.3.5. Сторінка оглядів

Зайдемо на сторінку певної публікації, перейдемо до блоку коментарів. Зробимо це з «телефона», щоб перевірити адаптивність:

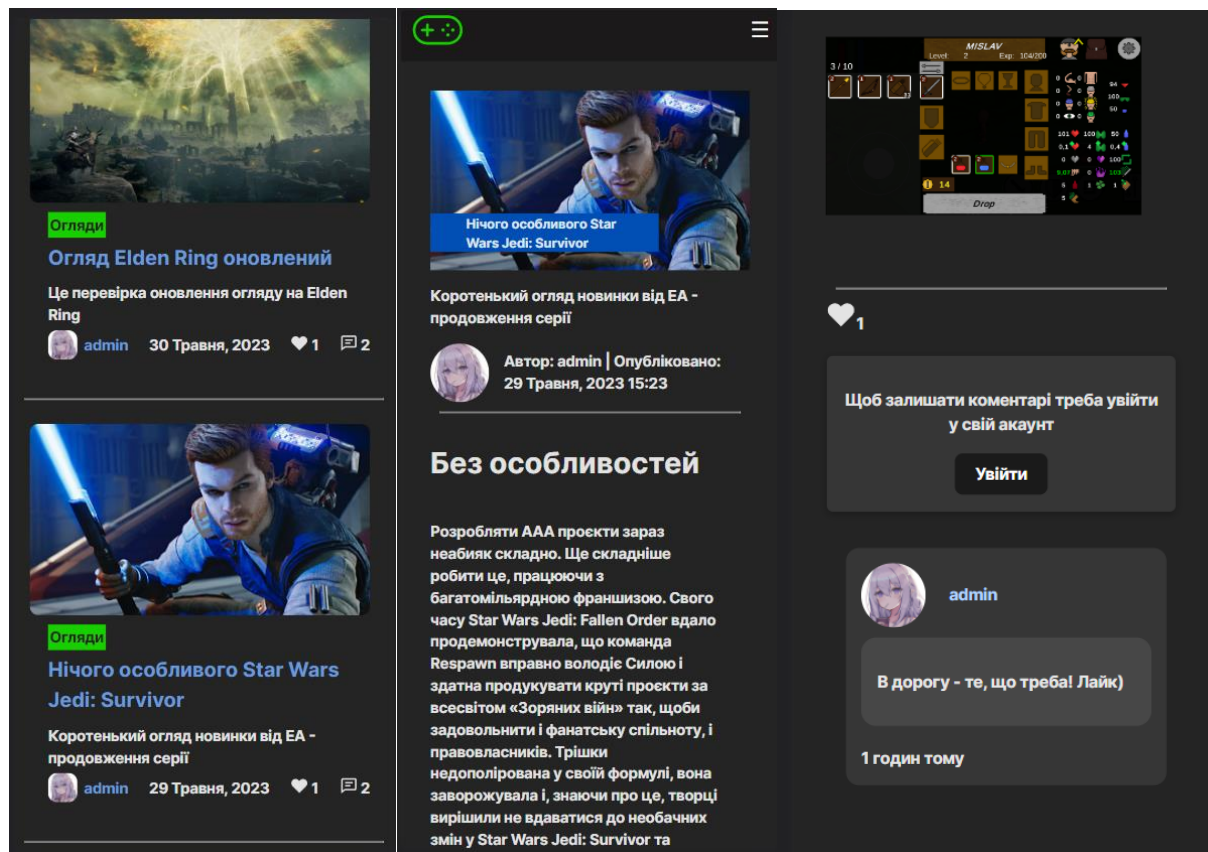


Рис.3.6. Сторінки вебсайту, вид з маленького екрану

Сторінка публікації може мати певну текстову стилізацію, контент поста може містити зображення:

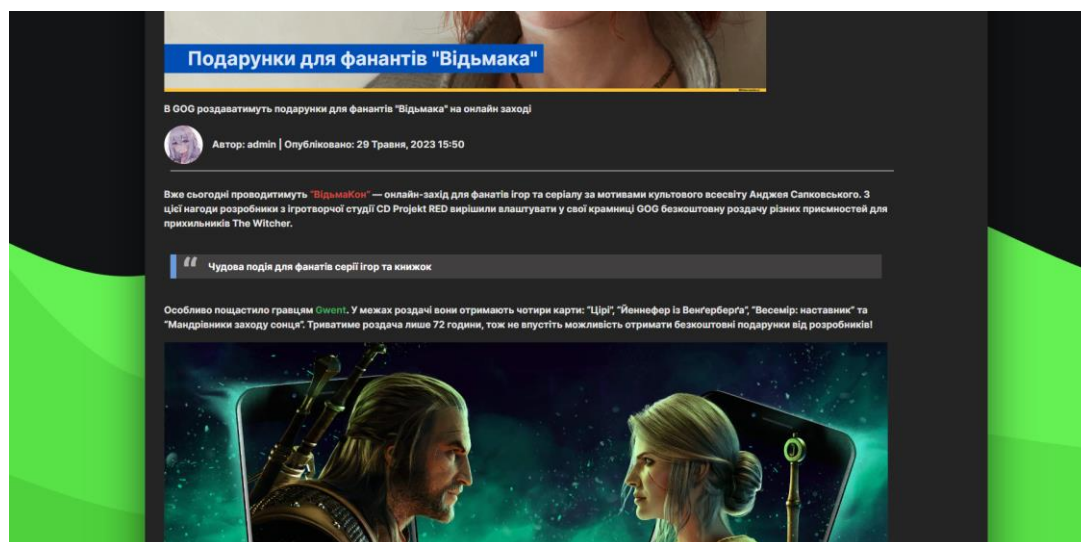


Рис.3.7. Сторінка публікації

Скористаємось пошуком, бачимо розділення на пости та ігри, знайдемо гру, подивимось інформаційну сторінку про неї:

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		69

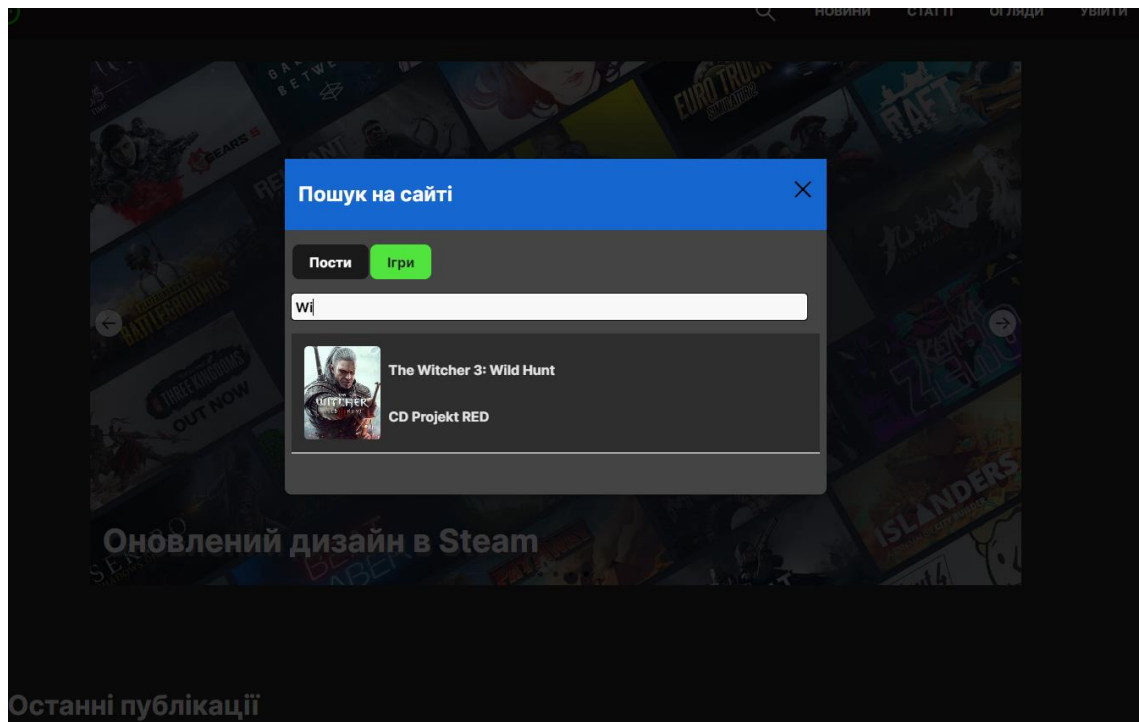


Рис.3.8. Пошук

Сторінка з грою має певну інформацію про гру. Наприклад короткий опис, платформи, на яких гра підтримується тощо:

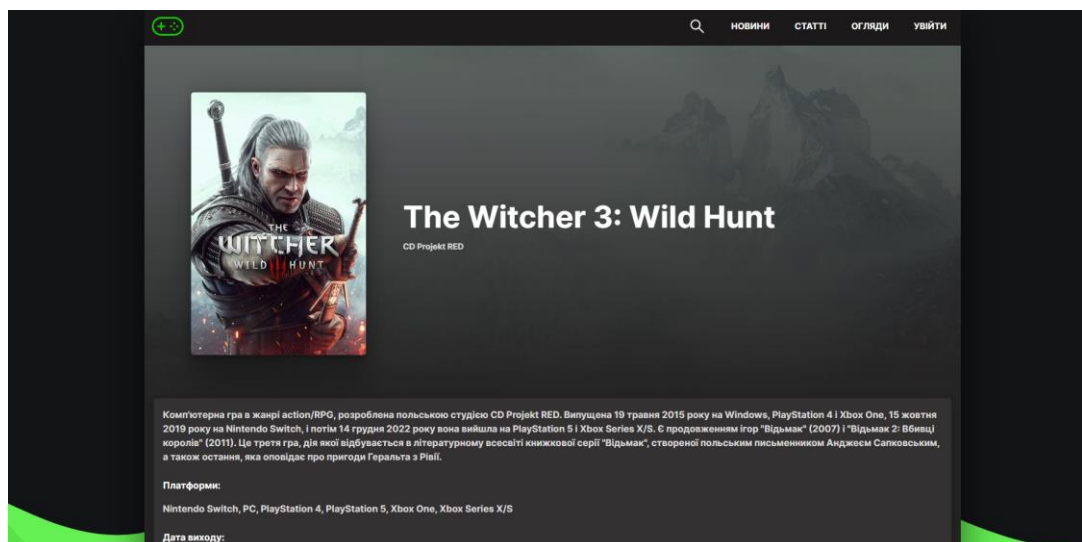


Рис.3.9. Сторінка з інформацією про гру

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

Сторінка з грою містить також карусель зображень та блок з досягненнями які можна отримати в цій грі:

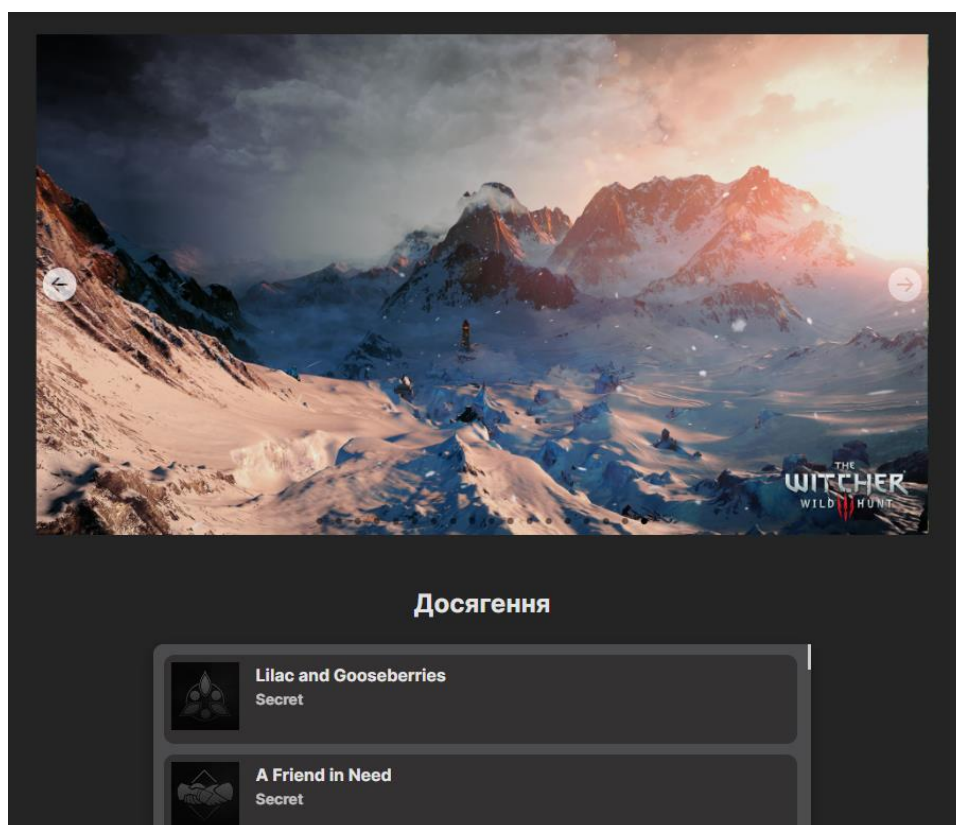


Рис.3.10. Блок каруселі зображень, досягнення

В кінці сторінки можемо побачити пости, які пов'язані з назвою гри:

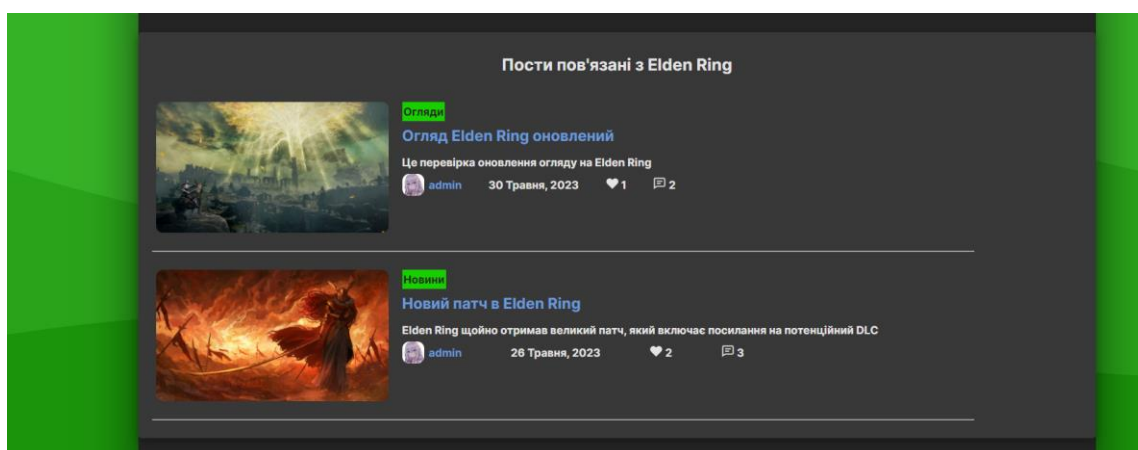


Рис.3.11. Пов'язані з грою публікації на вебсайті

Перейдемо на сторінку профілю користувача, можемо побачити дані про нього, можливість редагувати їх. Також є блок з активністю користувача на сайті:

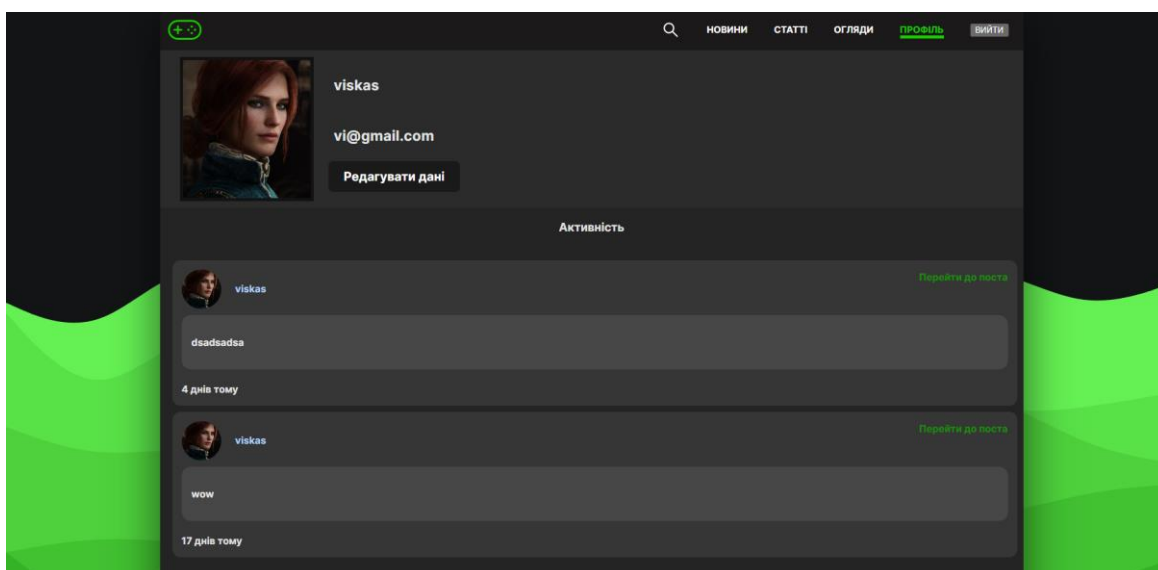


Рис.3.12. Сторінка профілю користувача

Зайдемо під адміністратором та перейдемо на сторінку його постів, які даний адмін створював. Є можливість редагувати та видаляти дані. Також можемо побачити, що на навігаційній панелі додалась можливість створити пост, додати гру на сайт:

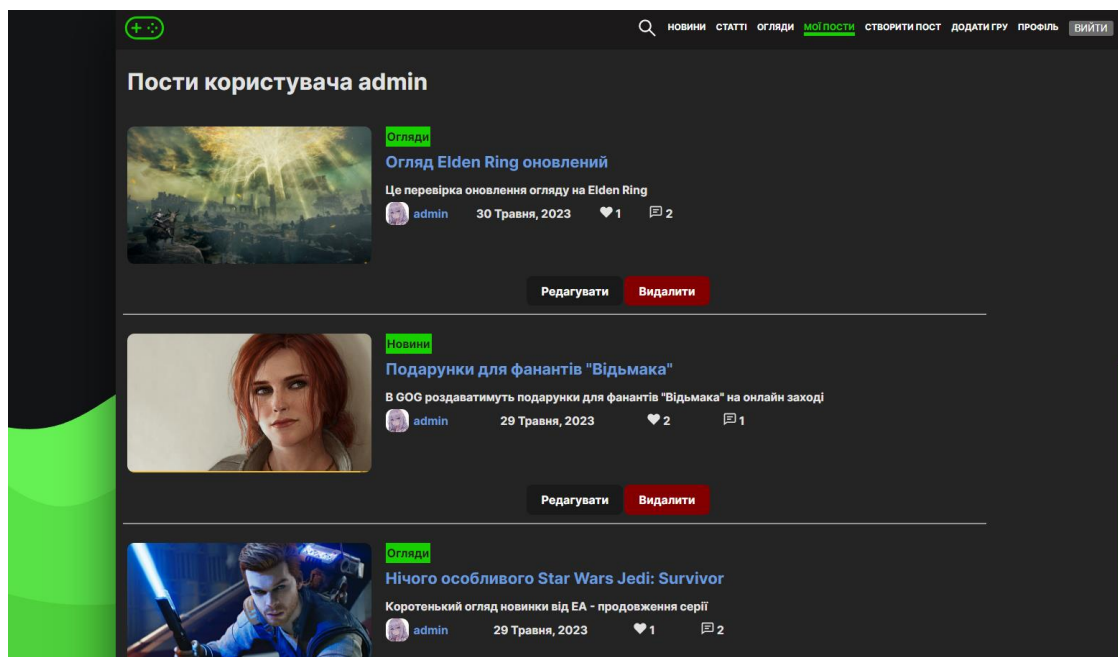


Рис.3.13. Публікації користувача

Можемо оновити дані про публікацію, на сторінці є форма, компонент для завантаження зображення та компонент – редактор контенту:

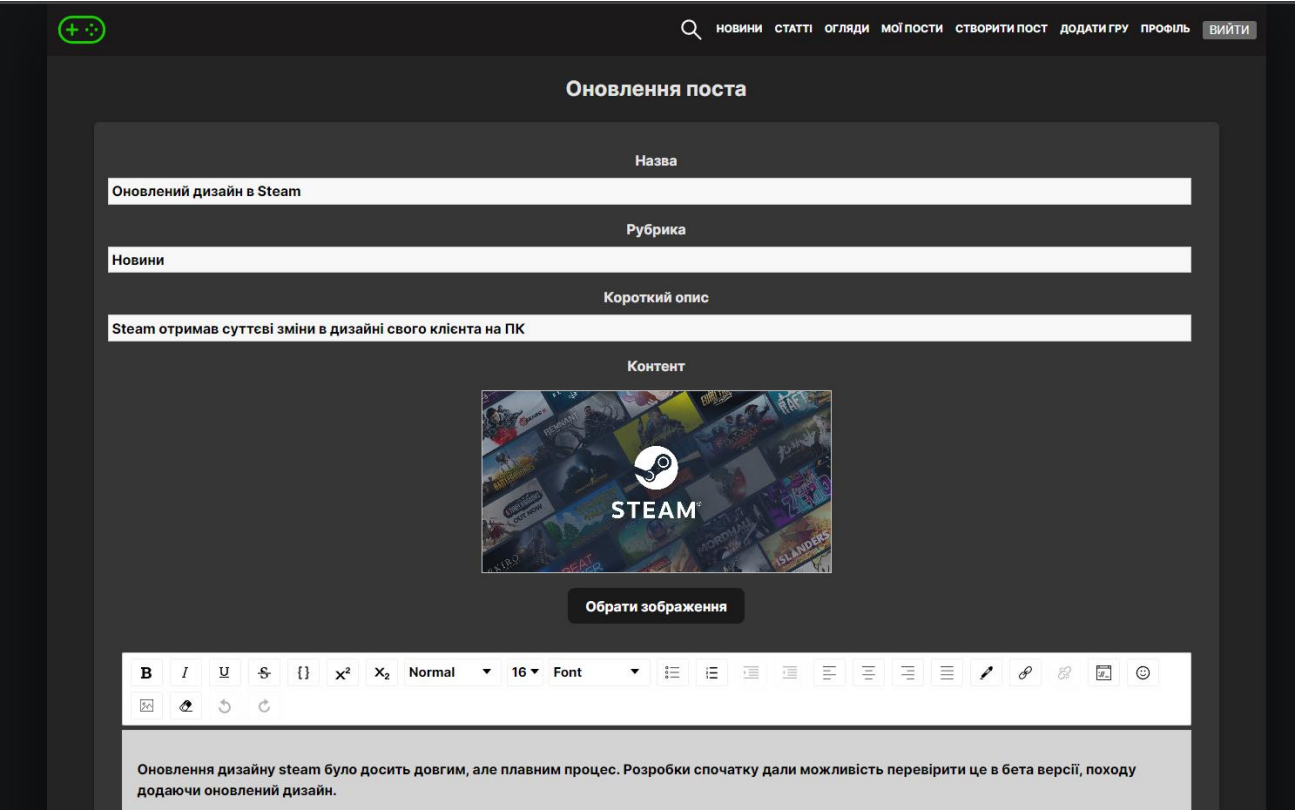


Рис.3.14. Оновлення даних публікації (1)

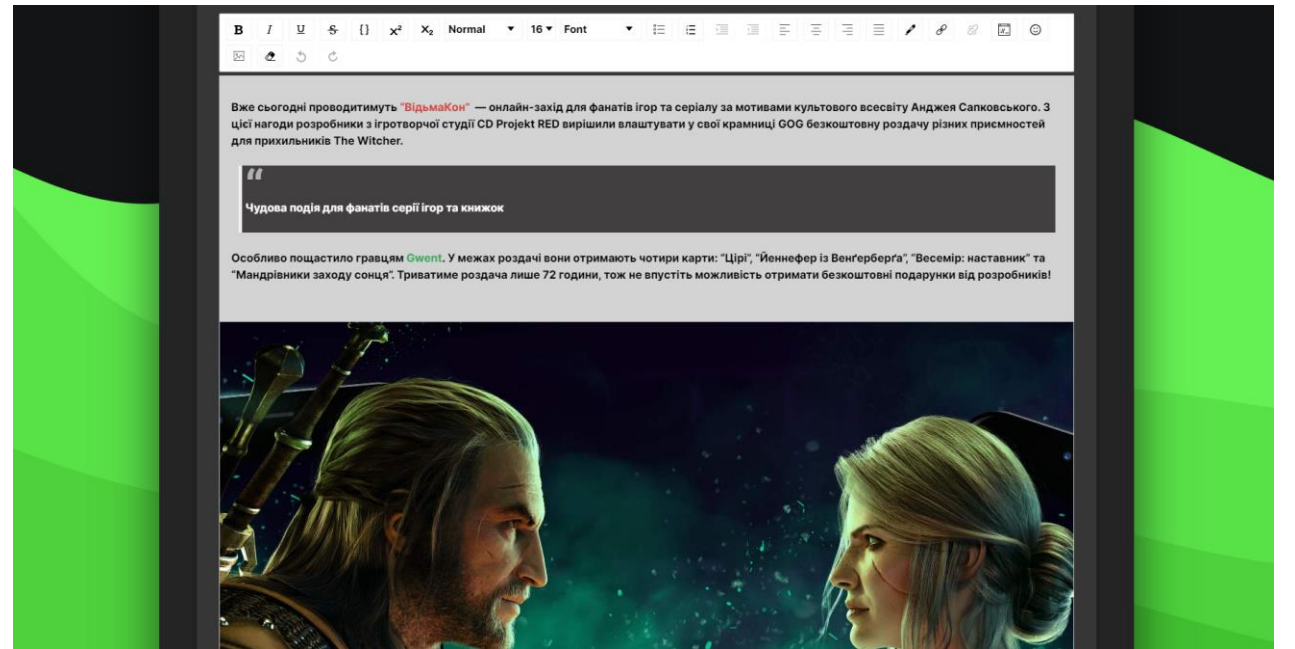


Рис.3.15. Оновлення даних публікації (2)

На таких формах оновлення, створення присутня валідація, якщо не впишемо обов'язкове поле, то буде помилка, можливість оновлення\створення даних заблокується, поки форма не стане валідною:

Рис.3.16. Додавання даних про гру, помилка про не введене поле

Заповнивши всі поля, додамо дані про гру на вебсайт:

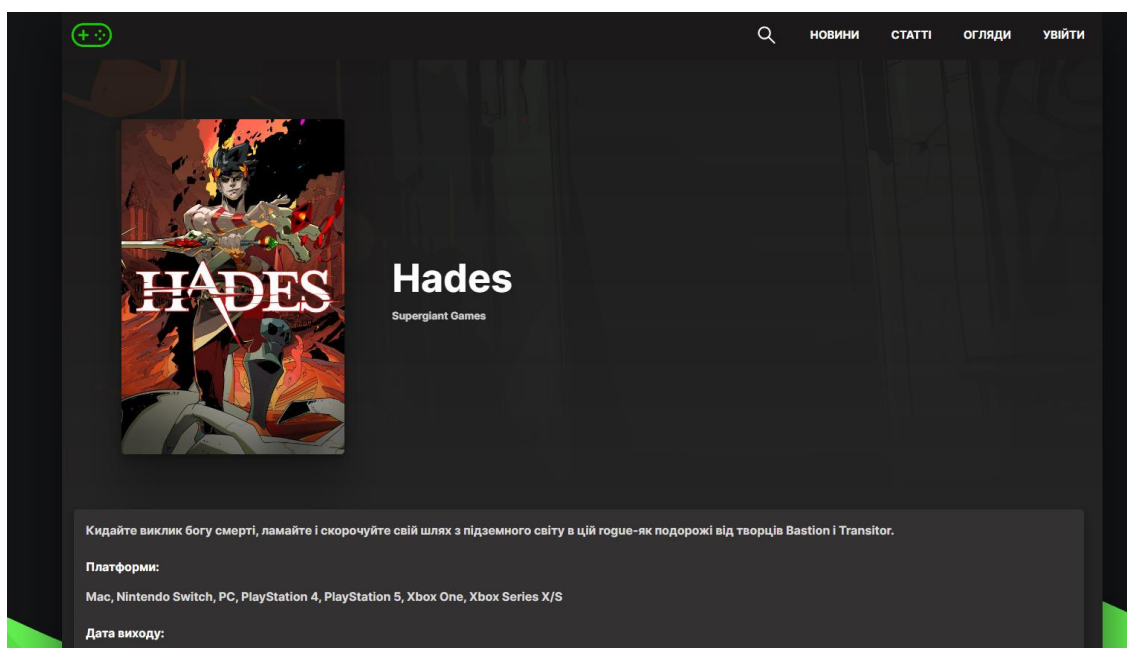


Рис.3.17. Додана гра на сайт

3.3 Тестування роботи системи

Щоб переконатись в правильності роботи вебсайту проведемо тестування його функціоналу. Опишемо моменти перевірки у вигляді чеклістів та тест кейсів. Розіб'ємо тестування вебсайту на декілька частин та протестуємо.

Наприклад тестування авторизації\реєстрації виглядає наступним чином:

Procedure	Expected Result	Status	Comment/Actual
Натиснути на кнопку "Увійти"	+	Пройшов	Відбувся перехід на сторінку авторизації
Натиснути на кнопку "Перейти до реєстрації"	+	Пройшов	Форма змінилась, додалися потрібні інпути для реєстрації
Натиснути на кнопку "Зареєструватися" не ввівши реєстраційних даних	+	Пройшов	Нічого не вводити. Кнопка реєстрації заблокована
Спробувати зареєструватись, ввівши пошту без знаку @ в імені домені	+	Пройшов	Нікнейм: test_user, Імейл - test@gmail.com, Пароль - 1111Password Кнопка реєстрації заблокована, з'явилась помилка про неправильний формат пошти
Спробувати зареєструватись, ввівши пароль 11	+	Пройшов	Нікнейм: test_user, Імейл - test@gmail.com, Пароль - 11 Кнопка реєстрації заблокована, з'явилась помилка про замалу кількість символів в паролі
Спробувати зареєструватись, не ввівши нікнейм	+	Пройшов	Нікнейм: , Імейл - test@gmail.com, Пароль - 1111Password Кнопка реєстрації заблокована, з'явилась помилка, що нікнейм не введено
Спробувати увійти в акаунт, ввівши тільки пароль	+	Пройшов	Кнопка "Увійти" заблокована
Натиснути на кнопку "Зареєструватися" ввівши всі дані вірно	+	Пройшов	Нікнейм: test_user, Імейл - test@gmail.com, Пароль - 1111Password Реєстрація успішна, перехід на головну сторінку
З форми реєстрації - натиснути на кнопку "Перейти до авторизації"	+	Пройшов	Форма змінилась, залишилися лише потрібні інпути для авторизації
Спробувати авторизуватись, ввівши пароль 11	+	Пройшов	Імейл - test@gmail.com, Пароль - 11 Кнопка "Увійти" заблокована, з'явилась помилка про замалу кількість символів в паролі
Спробувати авторизуватись, не ввівши імейл	+	Пройшов	Імейл - , Пароль - 1111 (вірний) Кнопка "Увійти" заблокована, з'явилась помилка про обов'язковість введення пошти
Спробувати авторизуватись, ввівши дані валідно, але не під існуючим користувачем	+	Пройшов	З'явилась помилка, що дані не вірні, користувача під введеною поштою не існує
Спробувати авторизуватись, ввівши дані валідно, існуючий імейл, але не правильний пароль	+	Пройшов	З'явилась помилка, що користувач ввів не правильний пароль
Спробувати авторизуватись, ввівши всі дані існуючого користувача вірно	+	Пройшов	Авторизація успішно

Рис. 3.18. Тестування авторизації\реєстрації у вигляді чеклістів

Перевірка пошуку						
ID	Description	Steps	Test Data	Expected Results	Actual Results	Status
1	Перевірка пошуку коректним запитом	1. Натиснути на іконку пошуку на навігаційній панелі 2. Обрати категорію "гри" 3. В поле пошуку ввести "Elden"	Текст пошуку: "Elden"	Виведуться гри, які включають в назві текст пошуку	Відповідає очікуваному	Пройшов
2	Перевірка пошуку, ввівши не коректний текст	1. Натиснути на іконку пошуку на навігаційній панелі 2. Обрати категорію "гри" 3. В поле пошуку ввести "dsadsa"	Текст пошуку: "dsadsa"	Нічого не буде знайдено, виведеться відповідне повідомлення	Відповідає очікуваному	Пройшов
3	Перевірка переходу на сторінку з грою після пошуку	1. Натиснути на іконку пошуку на навігаційній панелі 2. Обрати категорію "гри" 3. В поле пошуку ввести "Elden" 4. Обрати гру	Текст пошуку: "Elden"	Завантажиться сторінка яка відповідає пошуку	Відповідає очікуваному	Пройшов
Сторінка гри (звичайний користувач)						
ID	Description	Steps	Test Data	Expected Results	Actual Results	Status
1	Перевірка відображення сторінки	1. Зайти на сторінку з грою 2. Перелюнути сторінку	-	Весь контент сторінки відображається коректно, всі дані з сервера отримані	Відповідає очікуваному	Пройшов
2	Перевірка каруселі зображень	1. Зайти на сторінку з грою 2. Перелюнути сторінку 3. Натиснути декілька разів на стрілочку на каруселі зображень	-	Картинка змінюється на наступну, відповідно до вказаного напрямку на іконці	Відповідає очікуваному	Пройшов
3	Перевірка переходу на сторінку публікації	1. Зайти на сторінку з грою 2. Перелюнути сторінку 3. У блоці публікацій натиснути на одну з публікацій	-	Завантажиться сторінка відповідна сторінка публікації	Відповідає очікуваному	Пройшов
Сторінка гри (авторизований користувач - адміністратор)						
ID	Description	Steps	Test Data	Expected Results	Actual Results	Status
1	Перевірка відображення сторінки	1. Зайти на сторінку з грою 2. Перелюнути сторінку	-	Весь контент сторінки відображається коректно, всі дані з сервера отримані. Також має бути можливість редагування та видалення	Відповідає очікуваному	Пройшов
2	Перевірка редагування даних про гру	1. Зайти на сторінку з грою 2. Натиснути на кнопку "Редагувати" 3. Змінити назву гри, інші поля залишити як є 4. Натиснути "Оновити дані про гру"	Сторінка оновлення: Назва: "Elden Ring1"	Редирект на сторінку з даним про гру. Назва гри зміниться на введену	Відповідає очікуваному	Пройшов
3	Перевірка редагування даних про гру, ввести не валідні дані	1. Зайти на сторінку з грою 2. Натиснути на кнопку "Редагувати" 3. Ввести нову назву та розробника, інші поля залишити як є 4. Натиснути "Оновити дані про гру"	Сторінка оновлення: Назва: "" Розробник: ""	Кнопка оновлення даних заблокується. Виведуться помилка про не валідні дані	Відповідає очікуваному	Пройшов
4	Перевірка редагування даних про гру	1. Зайти на сторінку з грою 2. Натиснути на кнопку "Редагувати" 3. Обрати зображення для обкладинки гри	Завантажене зображення, формат .png	Редирект на сторінку з даним про гру. Буде виведена нова обкладинка гри	Відповідає очікуваному	Пройшов

Рис. 3.19. Тестування пошуку гри, сторінки гри з різними ролями користувача у вигляді тест кейсів

Під час тестування проблем не виникло, валідація працює як на клієнті так і на сервері, всі помилки обробляються, всі сторінки відображуються, розділення можливостей різних ролей працює коректно.

Інші функціональні частини вебсайту були протестовані подібним чином, все працює згідно очікуваного результату.

Висновки до третього розділу

Під час виконання третього розділу було зроблено фронтенд частину вебсайту, згідно всіх вимог та критеріїв. Це було зроблено за допомогою бібліотеки React. Користувацький інтерфейс вийшов швидким і зручним, були дотримані усі принципи SPA.

Також було продемонстровано порядок роботи з вебсайтом, були оглянуті його основні можливості. В кінці також провели тестування системи у ході якої визначали, що побудована вебсторінка вийшла надійною та правильною.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та розроблено вебсайт ігрового порталу, платформи для надання спільноті геймерів актуальної інформації про відеоігри, включаючи новини, статті та огляди.

У ході роботи було пройдено основні етапи розробки, а саме проектування, створення серверної частини, клієнтської, тестування і налагодження системи.

Перед етапом проектування, було визначено вимоги та критерії для подальшої розробки вебсайту, були оглянуті та проаналізовані аналогічні веб-додатки, був обраний стек технологій на якій буде розроблятися система.

На етапі розробки бекенду, на основі проектування, враховуючи всі можливі потреби для платформи, було створено серверну частину з архітектурним підходом REST API, використовуючі засоби розробки Node.js в поєднанні з фреймворком Express.js.

Маючи готовий сервер, було зроблено фронтенд частину. У ході виконання цього етапу, для побудови і налагодження користувацького інтерфейсу використовувалась бібліотека React. У хід йшли всі переваги реакту, принцип SPA, побудова на основі функціональних компонентів, використання різних пакетів та бібліотек, яку забезпечує чудова екосистема даної технології.

Провівши тестування, було отримано коректно-робочу вебсторінку ігрового порталу, яка може використовуватись як чудова платформа для спільноти геймерів.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GameRant [Електронний ресурс] – Режим доступу до ресурсу: <https://gamerant.com/>
2. IGN [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ign.com/>
3. PlayUA [Електронний ресурс] – Режим доступу до ресурсу: <https://playua.net/>
4. Node.js Design Patterns / Mariano Casciaro, 2014. - 454с.
5. Web Application Architecture: Principles, Protocols, and Practices / Leon Shklar and Rich Rosen, 2003. – 608с.
6. REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development / Fernando Doglio, 2018. – 402с. – (Second Edition.)
7. Документація мови для API – GraphQL [Електронний ресурс] – Режим доступу до ресурсу: <https://graphql.org/>
8. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / Martin Kleppmann, 2017. – 611с.
9. Документація до використання MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/docs/>
10. Документація до СУБД MongoDB Atlas [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/docs/atlas/>
11. Документація Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/uk>
12. Документація Express.js [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>
13. React: up and Running : Building Web Applications by Stoyan Stefanov / Stoyan Stefanov, 2021. – 222с.
14. Документація бібліотеки React [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/>

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

15. Документація бібліотеки React Router DOM [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/en/main>
16. FreeCodeCamp, стаття про патерни проектування для сучасних веб-додатків [Електронний ресурс] – Режим доступу до ресурсу: <https://www.freecodecamp.org/news/design-pattern-for-modern-backend-development-and-use-cases/#example-of-using-the-mvc-pattern-in-a-web-application>
17. Medium – платформа публікацій, часто про ІТ тематику [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/>
18. LogRocket – сервіс для розробників [Електронний ресурс] – Режим доступу до ресурсу: <https://logrocket.com/>
19. Документація JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/>
20. Документація пакету multer [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/expressjs/multer>
21. Сервіс для завантажень npm пакетів [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/>

ДОДАТОК

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

Частина лістингу `post-controller.js`, контролера публікацій на сервері, містить в собі методи для отримання даних про популярні пости за останній місяць, отримання даних про публікацію за ідентифікатором та пошук поста за назвою:

```
import * as fs from "fs";
import { validationResult } from "express-validator";

import Post from "../models/Post.js";
import User from "../models/User.js";

class PostController {
  async getPopularPosts(req, res) {
    try {
      const popularPosts = await Post.aggregate([
        {
          $match: {
            date: { $gte: new Date(Date.now() - 30 * 24 * 60 * 60 * 1000) },
          },
        },
        {
          $addFields: {
            likesCount: { $size: { $objectToArray: "$likes" } },
          },
        },
        {
          $sort: { likesCount: -1 },
        },
        {
          $limit: 5,
        },
      ]);

      const formattedPosts = popularPosts.map(
        ({ _id, title, likes, image }) => {
          return { _id, title, likes, image };
        }
      );

      res.status(200).json(formattedPosts);
    } catch (err) {
      res.status(404).json({ message: err.message });
    }
  }

  async getPostById(req, res, next) {
    const postId = req.params.pid;
```

```

    let post;
    try {
      post = await Post.findById(postId);
    } catch (err) {
      return res
        .status(500)
        .json({ message: "Something went wrong, could not find a post" });
    }

    if (!post) {
      return res
        .status(404)
        .json({ message: "Could not find post for provided id" });
    }

    res.json(post);
  }

  async searchPosts(req, res) {
    const searchQuery = req.query.q;

    try {
      let posts;
      if (searchQuery) {
        posts = await Post.aggregate([
          {
            $search: {
              index: "default",
              autocomplete: {
                query: searchQuery,
                path: "title",
              },
            },
          },
          {
            $limit: 5,
          },
          {
            $project: {
              _id: 1,
              title: 1,
              rubric: 1,
              description: 1,
              content: 1,
              likes: 1,
              comments: 1,
              creator: 1,
              date: 1,
              image: 1,
            },
          },
        ],
      );
    }
  }

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

]);

if (
  posts.length === 0 &&
  searchQuery !== undefined &&
  searchQuery.length > 2
) {
  posts = "Нічого не знайдено";
}
} else {
  posts = "";
}
res.json(posts);
} catch (error) {
  res.status(500).json({ error: "Error with searching posts" });
}
}

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		