Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом»

Виконав: студент 2-го курсу, групи 2ПІ-22м (д/н)
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Паляниця Д. Р.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д. І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«13» грудня 2023 р.
Опонент: к.т.н., доц. каф. ОТ Колесник І. С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
«13» грудня 2023 р.

Допущено до захисту
Завідувач кафедрою ПЗ
д.т.н., проф., Романюк О. Н.
«13» грудня 2023 р.

ВНТУ – 2023

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« 19 » вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Паляниці Дмитру Романовичу

1. Тема роботи – розробка методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом.

Керівник роботи: Катєльніков Денис Іванович, д.т.н., завідувач кафедри ПЗ, затверджені наказом вищого навчального закладу від « 18 » вересня 2023 р. № 247.

2. Строк подання студентом роботи

5 грудня 2023 р.

3. Вихідні дані до роботи: методи рендерингу сторінок – SSG, SSR, CSR, ISR; відкрита база даних ігор – igdb; фізичний сервер; база даних – MongoDB; мова розробки – Java Script; програмне оточення сервера – Node.js; фреймворк програмного оточення сервера – express.js.

4. Зміст розрахунково-пояснювальної записки: аналіз стану питання розробки серверної частини та задач дослідження; розробка методу і моделі роботи серверної частини; розробка програмного забезпечення серверної

частини; тестування програми; економічна частина; висновки; перелік використаних джерел; додатки.

5. Перелік графічного матеріалу: схема роботи веб-ресурсу; діаграма компонентів серверної частини; блок-схема алгоритму роботи серверної; блок-схема алгоритму роботи клієнтської частини; блок-схема принципу роботи адміністративної частини; тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д. І., д.т.н.,	19.09.2023	05.12.2023
5	Кавецький В. В., к.т.н, доцент кафедри ЕПВМ	08.11.23	22.11.23

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання розробки серверної частини та задач дослідження	20.09.2023 – 25.09.2023	Вик.
2	Розробка методу і моделі роботи серверної частини	25.09.2023 – 04.10.2023	Вик.
3	Вибір середовища та мови розробки	04.10.2023 – 07.10.2023	Вик.
4	Розробка програмного забезпечення серверної частини	07.10.2023 – 27.10.2023	Вик.
5	Тестування роботи системи	27.10.2023 – 08.11.2023	Вик.
6	Економічна частина	08.11.2023 – 22.11.2023	Вик.
7	Оформлення матеріалів до захисту МКР	22.11.2023 – 01.12.2023	Вик.

Студент _____ Паляниця Д. Р.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Катєльніков Д. І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК. 004.4:004.92

Паляниця Д. Р. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. с.

Атестаційна робота містить 151 сторінки, 53 рисунків, 15 таблиць, 33 джерела, 4 додатки.

У магістерській кваліфікаційній роботі розроблено методи та програмні засоби серверу на базі комбінованих технологій SSG та SSR для системи керування контентом для веб-платформи про відеоігри. Під час виконання магістерської кваліфікаційної роботи було проаналізовано предметну область, проаналізовано основні аналоги, виявлено їх головні недоліки та переваги, на основі яких було створено методи та програмні засоби саме на базі цих технологій. Також було створено гнучку архітектуру для подальшої інтеграції нового функціоналу.

За допомогою розроблених програмних засобів можна прискорити процес рендерингу сторінки, а також матимемо можливість оновляти сторінки в будь-який момент часу, адже користувач матиме екземпляр сторінки, що знаходиться у кеші. Розроблено структуру програмного інтерфейсу, створено макети дизайну та реалізовано графічну складову веб-платформи. Для розробки обрано мову програмування JavaScript та серверне середовище виконання JavaScript – Node.js.

Ключові слова: сервер, рендеринг, SSG, SSR, веб-платформа.

ABSTRACT

Master degree thesis has 151 pages, 53 images, 15 tables, 33 sources, 4 additions.

In the master's thesis, methods and software tools for a server based on combined SSG and SSR technologies for a content management system for a web platform about video games have been developed. During the master's thesis, the subject area was analyzed, the main analogues were analyzed, their main disadvantages and advantages were identified, on the basis of which methods and software were created on the basis of these technologies. A flexible architecture was also created for further integration of new functionality.

With the help of the developed software tools, it is possible to speed up the process of rendering the page, and we will also be able to update the pages at any time, because the user will have an instance of the page in the cache. The structure of the software interface has been developed, design layouts have been created and the graphic component of the web platform has been implemented. The JavaScript programming language and the server-side JavaScript runtime environment – Node.js – were chosen for development.

Keywords: server, rendering, SSG, SSR, web platform.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. На даний момент сфера веб-розробки розвивається та зростає великими темпами, разом із цим зростають вимоги користувачів до веб-ресурсів, та як наслідок збільшується конкурентність самих веб-ресурсів. У кожного веб-розробника є ціллю зробити швидкий, та зручний веб-ресурс, щоб задовольнити потреби користувачів та мати перевагу над конкурентами з боку SEO.

Наслідком конкуренції є розвиток, тому на даний момент існують кілька підходів до рендерингу та генерації веб-сторінок, а саме:

- Client-Side Rendering (CSR);
- Server-Side Rendering (SSR);
- Static-Site Generation (SSG);
- Incremental Static Regeneration (ISR).

Але попри все ці методи мають свої недоліки. Наприклад, метод CSR попри свою просту реалізацію, витрачає забагато часу на рендеринг веб-сторінки. Недоліком методу SSR є необхідність у великій роздільній здатності сервера та часі очікування сторінки від серверу. Для методу SSG це відсутність гнучкості, для зміни контенту потрібно повністю замінити стару сторінку на нову. А при використанні методу ISR недоліком буде занадто важка архітектура серверу.

А тому, попри всі переваги існуючих методів рендерингу, актуальною залишається задача розробки гібридного методу, що поєднуватиме переваги та мінімізуватиме недоліки існуючих методів рендерингу.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою роботи є підвищення продуктивності клієнт-серверної взаємодії при використанні веб-платформи

без втрати можливості індексації сторінок веб-ресурсу роботами пошукових систем.

Основними задачами дослідження є:

- провести аналіз існуючих методів і засобів рендерингу та генерації веб сторінок;
- запропонувати новий метод, що буде поєднувати переваги існуючих методів, та перекривати їх недоліки;
- розробити програмні компоненти та систему візуалізації на основі запропонованого методу;
- провести експериментальні дослідження розроблених засобів рендерингу та генерації веб-сторінок.

Об’єкт дослідження – процес розробки веб-платформ для відеоігор та інших видів мультимедійного контенту.

Предмет дослідження – методи та програмні засоби серверної частини веб-ресурсу для отримання користувачем статичних сторінок та можливістю вибіркового перерендерення сторінок адміністратором.

Методи дослідження. У процесі дослідження використовувались: теорія баз даних при організації зберігання контенту, теорія розподілених систем для побудови розподіленої клієнт-серверної веб-платформи, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень, методи об’єктно-орієнтованого програмування та шаблони програмування для розробки архітектури класів та сервісів програмного забезпечення.

Наукова новизна отриманих результатів.

Подальшого розвитку отримав метод рендерингу веб-сторінок, у якому, на відміну від існуючих методів рендерингу, використовується комбінація SSG- та SSR- генерації, що дозволяє, з одного боку, підвищити продуктивність клієнт-серверної взаємодії при використанні веб-платформи, і, з іншого боку,

забезпечити можливість кращої індексації сторінок веб-ресурсу роботами пошукових систем.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає у тому, що розроблено алгоритми та програмні засоби, що забезпечують функціонування веб-платформи про відеоігри на конкурентному рівні продуктивності, надійності і доступності для користувачів.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві [*], [**], автору належать такі результати: розробка серверу використовуючи технологію SSR; інтегрування в сервер технології SSG; створення API сервісу для отримання даних з відкритої бази даних ігор, розробка клієнтської частини, що буде відображатися користувачу як сайт; в клієнтській частині буде відображатися та інформація, що занесена в адмін-панель.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях: III Всеукраїнська науково – технічна конференція молодих вчених, аспірантів та студентів «Комп'ютерні ігри і мультимедіа як інноваційний підхід до комунікації - 2023»(Одеса, 2023); Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (20-21 листопада, 2023, м.Суми/Вінниця).

Публікації. Основні результати досліджень опубліковано у двох матеріалах конференції.

1 АНАЛІЗ СТАНУ ПИТАННЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ ТА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану питання розробки серверної частини веб-ресурсу

Ще приблизно 10 років тому не було React або Angular, і всі веб-сайти були «багатосторінковими» додатками. Відкриваючи адресу сайту в браузері, йшов запит на сервер, де умовний PHP генерував цілу сторінку (увесь HTML, CSS та JS) і повертав клієнту. При натисканні на посилання на іншу сторінку, знову надсилався такий же запит, сервер генерував нову сторінку і повертав її. Браузер видаляв стару сторінку та рендерив нову. Даний підхід називається Multi page application (MPA)[1].

SEO (Search Engine Optimization) – це напрямок, що займається продвиганням сайтів та підвищенням їх важливості серед інших сатйів на подібну тематику, чи краще зроблена SEO частина тим більше шансів має сайт бути у топі запитів у браузері. Тобто SEO являє собою важливий аспект конкурентоспроможності сайту[2]. На відміну від SMM, що також займається продвиганням того чи іншого продукту, сайту чи сервісу, SEO – більш технічний напрямок, тобто напряму залежить від якості написаного коду на клієнті та на сервері. Якщо розглядати клієнтську частину, то на швидкість завантаження сайту, а тобто і на SEO впливає безліч аспектів, включаючи розмір сторінки, кількість DOM-елементів, вага зображень, наявність відкладеного завантаження зображень та скриптів, велике навантження складного коду тощо. З боку сервера все залежить від його архітектури та методології його роботи. Тобто найкращим варіантом буде віддавати клієнту вже готовий документ без різноманітного рендерингу. Щоб сайт почав відображатись у пошукових запитах, потрібно щоб сторінки цього сайту проходили індексування від пошукової системи. Кожна пошукова система має робота, який проводить індексацію сайтів. Чим вищий трафік на сайті – тим більша ймовірність, що робот буде щастіше проходитись по цьому сайту і індексувати сторінки з новою інформацією. Крім трафіку і оновлення даних

робот пошукових систем оцінює швидкість завантаження сторінки і доступність читання інформації (наприклад оцінює чи текст достатньо контрастний по відношеню до фону, щоб користувачі могли нормально прочитати інформацію). З точки зору SEO Multi Page Application – це ідеальний підхід. Тому що за кожну сторінку відповідає окремий файл, і немає потреби проводити маніпуляції з DOM, що також має навантаження. Користувач має повну сторінку з усією необхідною інформацією і будь-який пошуковий бот відразу її зчитує.

З точки зору користувачів сайтів – не все так ідеально. При відкритті кожної наступної сторінки користувач чекає поки завантажаться всі ресурси заново та відрендериться сторінка. Звісно, більшість ресурсів уже закешовані, але все одно очікуємо повного регенерації сторінки.

Революцією в цьому підході стала поява фреймворків, які дозволяли реалізовувати так звані «односторінкові» сайти – Single Page Application (SPA) (рисунок 1.1)[3]. Суть в тому, що клієнт один раз завантажує CSS, JS та практично порожню HTML-сторінку, і потім JS уже будує все в клієнтському браузері. При переході між сторінками JS перебудовує лише ту частину, яка змінилась, і при цьому, за потреби, робить AJAX запит на сервер, щоб отримати необхідні дані.

Рисунок 1.1 – Схеми роботи SPA та MPA

У такому випадку клієнт був задоволений – один раз завантажив ресурси й усі сторінки плавно відображаються в браузері. Але тоді страждав SEO. Пошукові боти завантажували сторінку, а на ній нічого нема. Виконувати JS вони не вміли, тому й отримати таку ж сторінку, як користувачі, не могли. До того ж URL-адреса не змінювалась.

З часом URL-адресу почали змінювати при переході між сторінками. Спочатку це була хеш-навігація (за допомогою символу #, JS дозволяв змінювати хеш в адресі), потім повноцінна зміна адреси за допомогою HistoryAPI. Деякі пошукові боти почали виконувати JS, але це не було вирішенням. Рендеринг повноцінної сторінки – довгий та витратний процес. Кожен бот має обмеження по часу, ресурсах та кількості сторінок, які він може просканувати. Розв’язанням проблеми SEO і став серверний рендеринг.

І ще трішки пізніше, а саме у 2016 році почали випускати фреймворки, що організовують рендеринг на стороні серверу, для фреймворку Vue розробили надфреймворк Nuxt.js, а для бібліотеки React.js розробили нфдфреймоврк Next.js. Ці «надфреймворки» стали вирішенням багатьох проблем і SEO і оптимізації сайтів у цілому, вдихнули нове життя та надали нові можливості для таких фреймворків.

1.2 Порівняльний аналіз аналогів

Кожен сервер працює згідно певної методології відображення сторінок, протоколів надсилання файлів, роутингу по файлах, редіректів тощо. Одним із найважливіших аспектів ефективності роботи серверу є метод по якому сервер віддає на клієнт файли-сторінки.

Кожен розробник ставить собі за мету зробити щось нове та щось краще за аналоги. Спочатку був розроблений метод за яким сервер просто віддає цілу сторінку, потім з’явилися нові методи рендерингу, що покращили гнучкість та зручність адміністрування сайту та оновлення інформації на ньому, але це викликало проблеми з оптимізацією, тому відносно недавно почалися пошуки варіантів вирішення проблеми оптимізації із неурізанням зручності

адміністрування. Більше проблем – більше варіантів вирішення – більша конкуренція. Наслідком конкуренції є розвиток, тому на даний момент існують кілька підходів до рендерингу та генерації веб-сторінок, а саме:

1. Client-Side Rendering (CSR);
2. Server-Side Rendering (SSR);
3. Static-Site Generation (SSG);
4. Incremental Static Regeneration (ISR).

SSG – це метод віддачі веб-сторінок, згідно якого попередньо завантажені готові та цілі HTML-сторінки на сервері просто відправляються на клієнт за запитом з нього[4]. Це означає, що сторінки кешовані і швидко завантажуються. З точки зору SEO – це ідеальний варіант. Для користувача сторінка відображається ще швидше ніж при SSR, ми не очікуємо дані на сервері, тому TTFB(Time To First Byte) менший. Недоліком даного підходу є те, що при оновленні даних та контенту, потрібно замінити старі HTML файли на нові, цей процес є доволі незручним та може зайняти доволі багато часу. Якщо сайт містить близько 100 000 сторінок, то цей процес може зайняти досить багато часу. (рисунок 1.2).

SSR – це метод послідовного рендерингу та віддачі сторінки клієнту [5]. Головна особливість цього підходу в тому, що вся сторінка рендериться на сервері при першому запиті, та рендерингу віддається на клієнт у вигляді повного та готового HTML-файлу. Дані завантажуються завжди актуальні, адже при кожному перезавантаженні сторінки відбувається запит на сервер, ререндеринг сторінки та знову віддача на клієнт вже нового HTML-файлу із новими даними в ньому. З точки зору SEO все також непогано – повертається повноцінна сторінка, проте можуть бути проблеми у випадку якщо сторінка надто складна, тобто на ній багато інформації або там відображається якась статистика, яку потрібно заново підраховувати на сервері, або якщо наприклад на сайт заходить одразу багато людей і ходить по сторінках – це може спричинити надто велике навантаження на сервер, отже потрібно також

розуміти, що SSR – це доволі зручний метод відображення актуальної інформації, але при великому розмірі сайту та залежно від його трафіку та призначення треба розраховувати на фізичний сервер із достатніми потужностями для цього. Користувачі також задоволені – працюють зі звичайним SPA, різниці вони не бачать.

Хоч і виникає потреба в підтримці потужного сервера, який буде обробляти усі запити, тут є і перевага. Можна налаштувати кешування для найпопулярніших запитів, що прискорить завантаження та відображення сторінки. (рисунок 1.3).

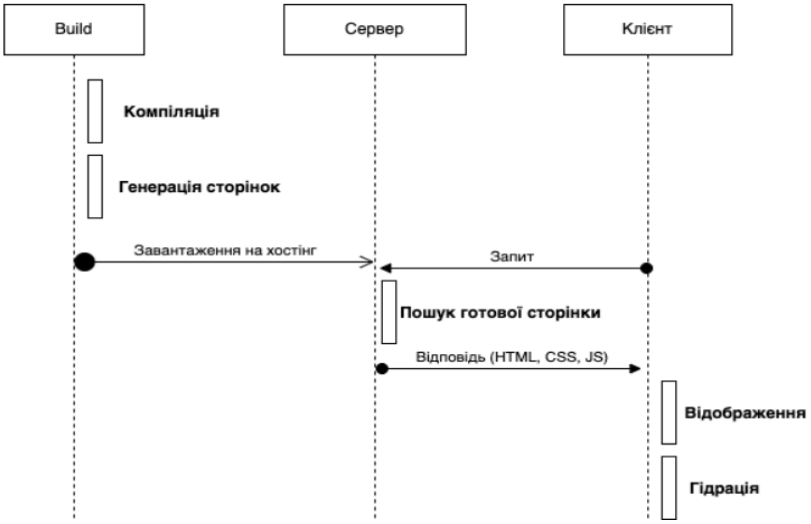


Рисунок 1.2 – Принцип роботи методу SSG

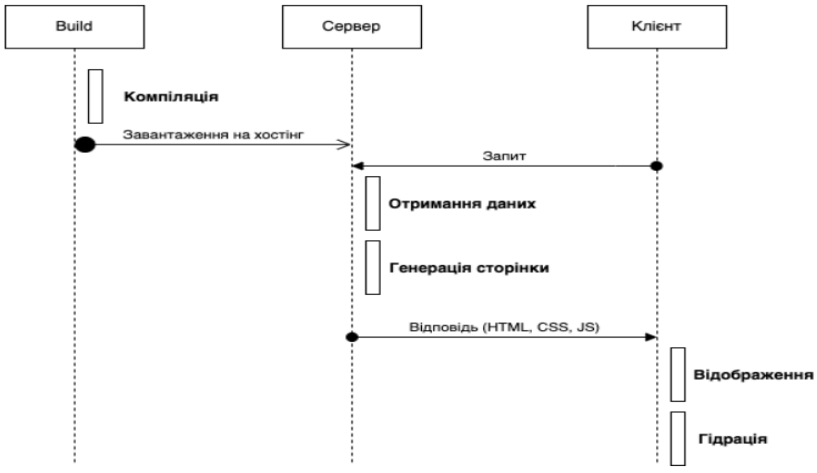


Рисунок 1.3 – Принцип роботи методу SSR

CSR – це метод генерації веб-сторінок, який використовує JavaScript для генерації HTML-сторінок на клієнтському боці. У файлі index.html з тегів є лише один div з id=”root”, де і будується вся сторінка. При переході між сторінками змінюється URL-адреса в браузері та підтягуються дані з сервера, за потреби. Це означає, що сторінки можуть бути дуже інтерактивними. Це забезпечує легкість маштабування, і данні можуть використовуватися різними клієнтами (мобільні додатки, сервери, веб-сторінки). Однак, цей метод має низьку SEO, більше навантажує пристрої клієнтів та має повільний час першого завантаження (рисунок 1.3) . Це доволі неоднозначний підхід. Якщо під час SSR методу, у всіх користувачів може бути різниця у швидкості завантаження залежно від доступу до інтернету, а час на рендеринг буде приблизно однаковий, так як для усіх користувачів для усіх запитів рендеринг сторінок здійснює один і той самий сервер, то під час CSR підходу різниця у швидкості завантаження сторінки буде не лише залежати від якості інтернет зв’язку, а ще і від потужності пристроя, через який відбувається запит на сторінку сайту, так користувачі потужних комп’ютерів будуть отримувати готову сторінку набагато швидше ніж навіть користувач на смартфоні, а варто відмітити, що більшу частину користувачів на даний момент на сайтах мають сами користувачі зі смартфонів.

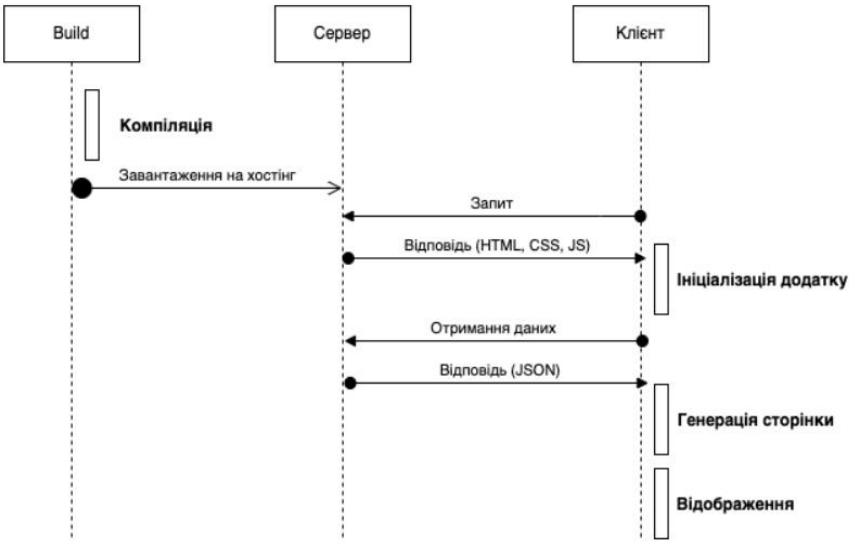


Рисунок 1.4 – Принцип роботи методу CSR

ISR – це метод генерації веб-сторінок, який реалізовується за допомогою розділення сторінок за частотою оновлення в ручну [6]. Ті, що потребують частого оновлення будуть рендеритися за допомогою SSR, а ті що оновлюються рідко – за допомогою SSG. Тому цей метод є складним у реалізації та потребує потужний сервер (рисунок 1.5). Якщо детальніше оглянути цей метод, то можна умовно навести такий приклад – стоірнки наприклад послуг або контактна сторінка буде відпрацьовуватись за методом SSG, тобто буде просто HTML-файл який просто надсилається на клієнт, а наприклад стоірнки блогу, статті, або сторінки із відеоіграми як у моєму випадку, будуть рендеритись за методом SSR при кожному запиті клієнта, тому що постійно треба оновлювати наявність нових елементів, кількість переглядів, коментарі тощо.

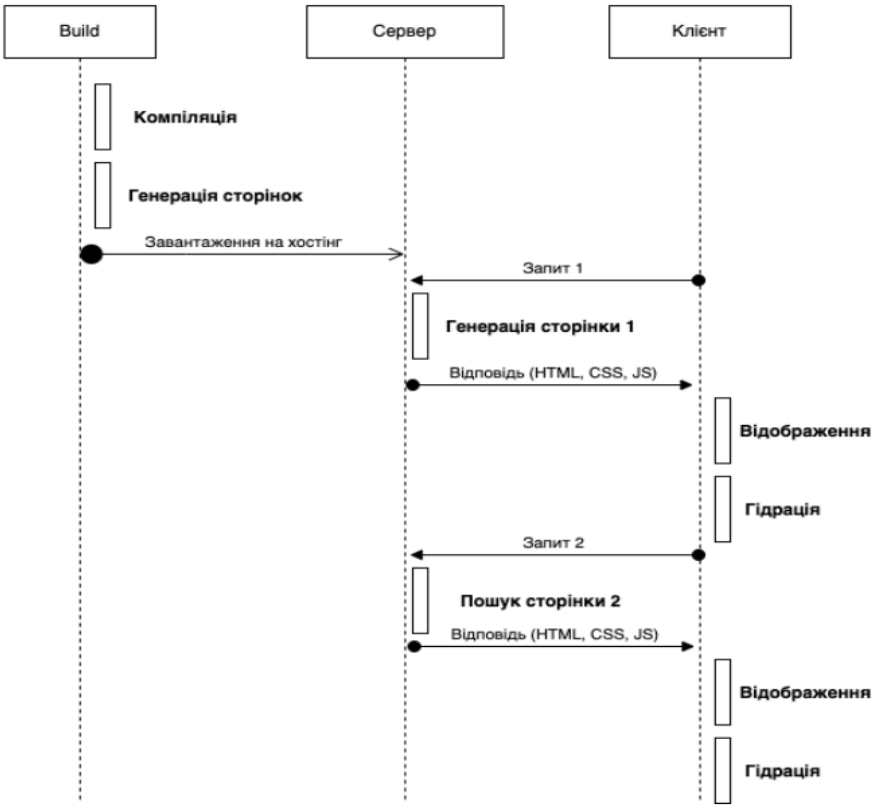


Рисунок 1.5 – Принцип роботи методу ISR

Проаналізувавши всі методи, визначено їх переваги і недоліки, які були враховані при розробці комбінованого методу (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристики типів рендерингу

Тип рендерингу	SEO	Швидкість	Гнучкість	Автоматизація оновлення
SSG	Високий	Висока	Обмежена	–
SSR	Середній	Середня	Середня	–
CSR	Низький	Низька	Висока	+
ISR	Середній	Середня	Середня	+
Комбінований метод	Високий	Висока	Середня	–

Таблиця порівняльних характеристик показала, що розробка власної системи є доцільною. В результаті нове рішення покриває недоліки існуючих методів та поєднує їх сильні сторони.

Суть комбінованого методу полягає у простоті архітектури сервера та покращенням оптимізації віддачі сторінок. Під час розробки клієнтської частини будуть зберігатись HTML-файли, сервер у свою чергу при отримані спеціального запиту буде проводити рендеринг на боці серверу для оновлення інформації, після чого буде зберігати готові файли в умовний кеш. Це дасть змогу не рендерити всі сторінки при отриманні запиту на них. Коли користувач буде заходити на сторінку йому буде одразу ж приходити готова HTML-сторінка без різноманітних ререндерингів.

1.3 Аналіз стану питання розробки інтерфейсу програмування додатків (API)

API (інтерфейс програмування додатків) – це набір правил та протоколів, які дозволяють різним програмам взаємодіяти між собою. Вони діють як посередник між різними програмами та системами, дозволяючи їм обмінюватися інформацією та взаємодіяти. API може бути внутрішнім (для внутрішнього використання в одній компанії) або зовнішнім (для взаємодії зі сторонніми службами) [7] (рисунок 1.6).

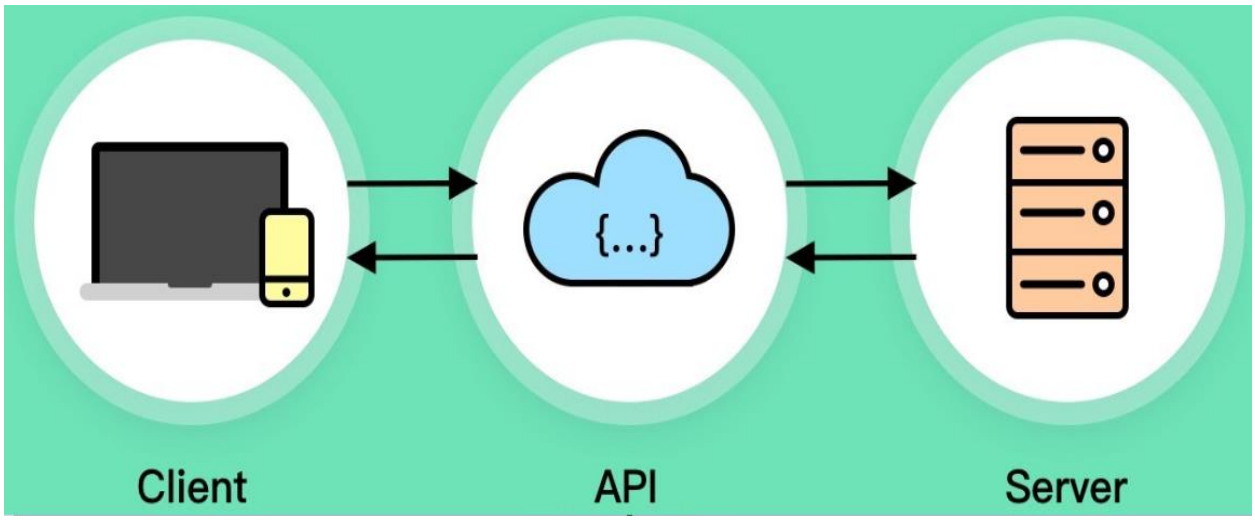


Рисунок 1.6 – Принцип роботи API

Існує два основні види API:

1. API браузера;
2. Стороннє API.

До API браузера відносяться:

1. API для роботи з документами, завантаженими в браузер. Наприклад – DOM (Document Object Model) API [8] (рисунок 1.7), який дозволяє працювати з HTML та CSS – створювати, видаляти та змінювати HTML, динамічно змінювати зовнішній вигляд сторінки тощо.

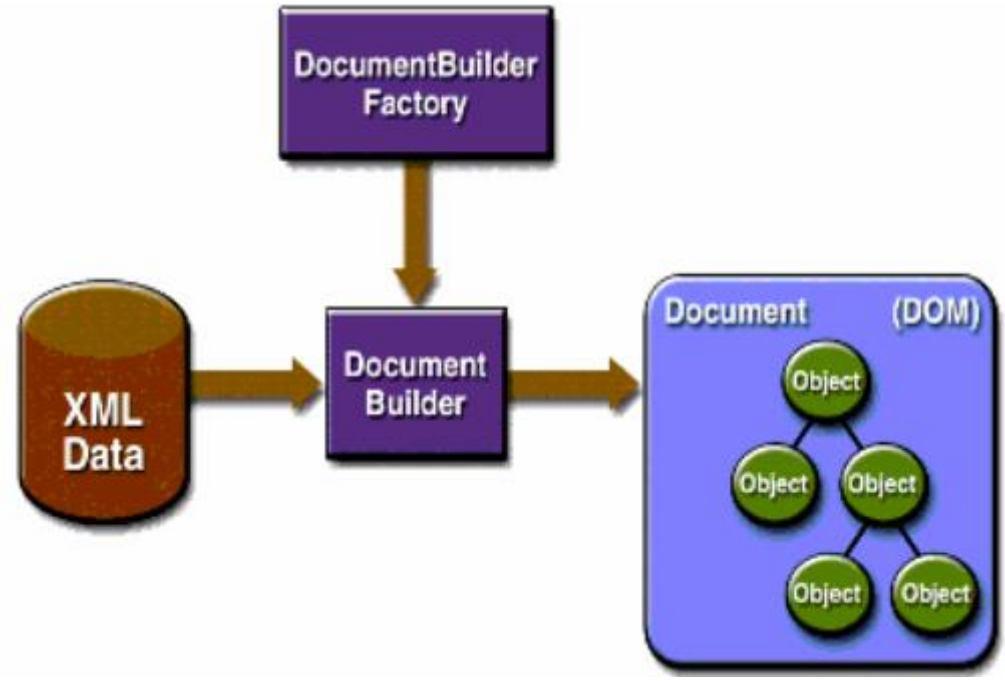


Рисунок 1.7 – Принцип роботи DOM API

1. API, які приймають дані з сервера – це API, які часто використовуються для оновлення невеликих частин веб-сторінки. Ця, здавалося б, незначна деталь має величезний вплив на продуктивність і поведінку сайтів, так як немає необхідності перезавантажувати всю сторінку, якщо потрібно просто оновити список товарів або нові доступні історії. Це також зробить додаток або веб-сайт більш чуйним і жвавим. Список API, які роблять це можливим, включає: XMLHttpRequest і Fetch API [9,10].
2. Графічні API – це API, що широко підтримуються браузерами, найпопулярнішими з яких є Canvas і WebGL, які дозволяють програмно змінювати піксельні дані, що містяться в елементі HTML <canvas> для створення 2D і 3D зображень. Наприклад, в них можна намалювати такі фігури, як прямокутники або кола, імпортувати зображення на полотно та застосувати до нього такі фільтри, як сепія або градації сірого, за допомогою Canvas API або створити складне 3D-зображення з освітленням і текстурами за допомогою WebGL. Такі API часто використовуються в поєднанні з API циклу анімації (наприклад, window.requestAnimationFrame()) та іншими для створення постійно мінливого зображення на екрані, як у мультфільмах чи іграх.
3. Аудіо та відео API – це такі API, як HTMLMediaElement, Web Audio API та WebRTC, вони дозволяють робити маніпуляції з медіа. Наприклад, створити власний інтерфейс користувача (UI) для відтворення аудіо/відео, виводити субтитри на екран, записувати відео з веб-камери для обробки в полотні, або для передачі на інший комп'ютер під час відеоконференції, застосовувати звукові ефекти до аудіофайлів (наприклад, посилення, спотворення, панорамування тощо).
4. API пристроїв – це API для обробки та зчитування даних із сучасних пристроїв веб-способом. Наприклад, API геолокації дозволяє

зчитувати дані про місцезнаходження пристрою. Інші приклади включають сповіщення користувача про оновлення веб-застосунку за допомогою системних сповіщень або вібрації.

5. API сховища на стороні користувача – це API, що дає можливість зберігати інформацію на стороні клієнта дуже корисна, коли вам потрібно створити додаток, який зберігатиме свій стан між перезавантаженнями сторінок або навіть працюватиме, коли пристрій знаходиться в автономному режимі. Таких API на даний момент досить багато. Наприклад, просте сховище даних у форматі ім'я/значення Web Storage API або сховище даних у форматі таблиці IndexedDB API.

До сторонніх API належать:

1. Twitter API – API для додавання таких функцій, як показ останніх твітів на сайті (рисунок 1.8).

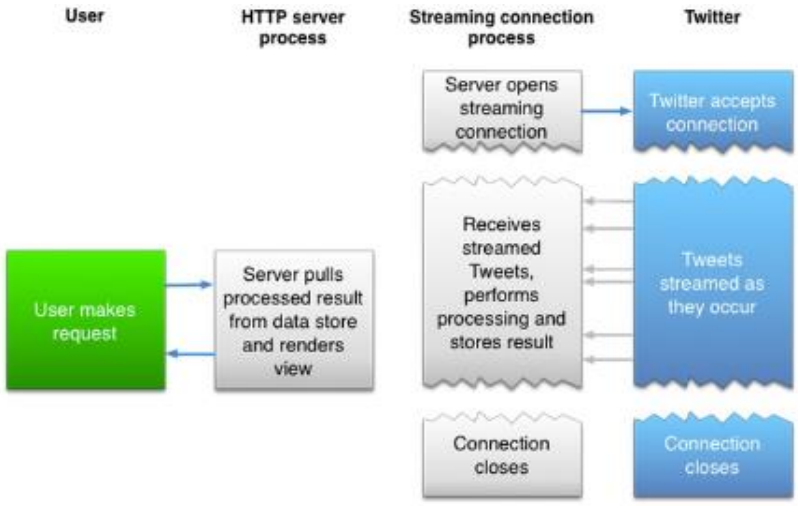


Рисунок 1.8 – Принцип роботи Twitter API

1 Google Maps API – API для роботи з картами на веб-сторінці. Тепер це цілий набір API, які можуть впоратися з широким спектром завдань, про що свідчить Google Maps API Picker.

2 Пакет Facebook API – API, який дозволяє використовувати різні частини платформи Facebook у вашому додатку, включає він, наприклад, можливість входити за допомогою логіна Facebook, оплачувати покупки в додатку, показувати цільову рекламу тощо.

3 YouTube API – API, який дозволяє вбудовувати відео YouTube на свій сайт, шукати, створювати плейлисти тощо.

4 Twilio API – це фреймворк для вбудовування функцій голосового та відеозв'язку у ваш додаток, надсилання SMS/MMS з програми тощо.

API запити – це способи взаємодії з API для отримання або відправки даних [11]. Основні типи запитів включають:

– GET-запит – цей запит використовується для отримання даних з сервера. Наприклад, запит на отримання інформації про користувача з сервера.
– POST-запит – цей запит використовується для відправки даних на сервер для створення нового ресурсу, наприклад, створення нового користувача.
– PUT-запит – цей запит використовується для оновлення існуючого ресурсу або створення нового, якщо він не існує. Наприклад, оновлення інформації про користувача.
– DELETE-запит – цей запит використовується для видалення ресурсу на сервері, наприклад, видалення запису користувача.

Серед бібліотек запитів слід виділити такі бібліотеки:

1 Axios – HTTP-клієнт на основі промісів для браузера та Node.js дозволяє перехоплювати запити та відповіді, перетворювати дані запиту та відповіді, скасовувати запити та автоматичне перетворення даних JSON.

2 Got – зручна і потужна бібліотека HTTP-запитів для Node.js, дуже легка, працює з Promise і Stream.

3 Superagent – невелика прогресивна бібліотека HTTP-запитів на стороні клієнта та модуль Node.js з тим самим API, що підтримує багато високорівневих функцій HTTP-клієнта.

4 Node Fetch – легкий модуль, який переносить Fetch API в Node.js, узгоджується з API, використовує власні функції промісів і асинхронних функцій.

Найбільш використовуваними та ефективни інструментами для тестування API є:

1 Postman – це інструмент для тестування API, який дозволяє створювати, надсилати та тестувати HTTP-запити та перевіряти відповіді на них.

2 Swagger – це інструмент для документування та тестування API, який дозволяє автоматично створювати документацію API з опису структури API у форматі YAML або JSON файлу.

3 SoapUI – це інструмент для тестування веб-сервісів, який дозволяє створювати та виконувати тести на SOAP-протоколі.

4 Fiddler – це інструмент для аналізу та налагодження HTTP-трафіку між веб-браузером та веб-сервером. З точки зору тестування API, Fiddler дозволяє перехоплювати, аналізувати та модифікувати HTTP-запити та відповіді, що передаються між клієнтом та сервером. Це дозволяє тестувальникам здійснювати валідацію запитів та відповідей, перевіряти правильність передачі параметрів, куків та інших елементів запитів, відстежувати проблеми з трафіком, а також виявляти і локалізовувати проблеми з API.

5 JMeter – це інструмент для тестування продуктивності та функціональності програмного забезпечення, який може бути використаний для тестування API. З точки зору тестування API, JMeter є інструментом, що дозволяє створювати запити до API, аналізувати відповіді та оцінювати продуктивність та функціональність API. JMeter може бути використаний для

створення навантаження на API, щоб виміряти продуктивність, час відповіді та інші метрики, які допоможуть виявити помилки в API та покращити його продуктивність та функціональність.

Якщо розглядати саме API, які використовуються на стороні серверу, то це здебільшого API для отримання певних даних.

Для цього існують відкриті та закриті бази даних. Тобто десь на фізичному сервері існує програмний сервер, який має своє API, що займається лише прийомом запитів, аналізом запитів та віддачею даних, що вимагаються у запиті.

Існують три види таких баз даних:

1. Відкриті бази даних;
2. Умовно відкриті бази даних;
3. Закриті бази даних.

Відкриті бази даних вимагають лише наявності коректного та валідного запиту, після обробки якого сервер відкритої бази даних надсилає запит у відпоідь із даними, які були запитані нашим сервером.

Умовно відкриті бази даних працюють за схожим принципом, але мають суттєву відмінність у доступі. Для отримання даних із такої бази даних потрібно мати спеціальний API-ключ, що працює як пароль для авторизації у системі. Для цього зазвичай потрібно зареєструватись у цій системі та зробити запис на генерацію API-ключа для вашого акаунту. Називаються вони умовно відкритими, тому що у таких баз даних, частина запитів знаходяться у відкритому доступі, і при наявності безкоштовного API-ключа можна отримати ці дані, а іншу частину даних можна отримати тільки якщо оформити підписку на цьому сервісі. Так залежно від рівня вашої підписки, буде змінюватись рівень доступу за вашим API-ключем.

Також є повністю закриті бази даних, працюють вони як і умовно відкриті бази даних, але з відмінністю у тому, що для отримання API-ключа у такій системі потрібно обов’язково оформлювати платну підписку за вашим

ключем. Проте такі закриті бази даних зазвичай мають якусь цінну та унікальну інформацію.

У даній дипломній роботі я буду використовувати саме умовно відкриту базу даних. Використовуватись буде саме база даних, системи IGDB. Ця база даних містить в собі ігрові новини, інформацію про різні відеоігри, інформацію про різні платформи для ігор та дані про різні ігрові події, які відбулися чи відбудуться [12].

1.4 Постановка задач розробки

Після аналізу актуальності стану питання методів рендерингу та порівняння існуючих аналогів за певним переліком критеріїв було визначено завдання, які необхідно виконати для розробки серверної частини для реалізації комбінованого методу рендерингу веб-сторінок:

- розробити загальну модель системи;
- розробити алгоритм роботи серверу;
- розробити власні API-запити для отримання даних із віддаленої бази даних;
- розробити власні API-запити для отримання даних із власної бази даних;
- розробити власні API-запити для оновлення, запису та видалення даних із власної бази даних;
- розробити принцип роботи адмін-панелі;
- розробити принцип роботи клієнтської частини;
- провести тестування серверної частини веб-ресурсу.

1.5 Висновки

У першому розділі було розглянуто актуальність питання створення комбінованого методу рендерингу веб-сторінок та проаналізовано питання вибору інтерфейсу програмування додатків (API), з метою покращення умов

використання клієнтом веб-ресурсів, та зручності взаємодії адміністратора з контентом веб-сторінки.

Було проаналізовано можливі підходи до вирішення питання створення комбінованого методу рендерингу, шляхом порівняння систем-аналогів, окреслення їх сильних та слабких сторін, таких як SEO, швидкість, динамічність та автоматизація оновлення.

Серед аналогів було виділено такі методи, як SSR, SSG, ISR, CSR. Основними недоліками цих методів є повільне перше завантаження, потреба у потужному сервері, низьке SEO та складність реалізації. Тому було вирішено розробити власний комбінований метод, який би мав критерії кращі ніж існуючі методи.

З огляду на недоліки аналогів було визначено такі основні задачі для розробки, як розробка методу з високим SEO, розробка методу з високою швидкістю рендерингу, розробка методу з гнучкою зміною контенту.

У результаті порівняння було відображено необхідність розробки магістерської кваліфікаційної роботи та сформульовано задачі, які необхідно розв’язати для вирішення питання.

2 РОЗРОБКА МЕТОДУ І МОДЕЛІ РОБОТИ СЕРВЕРНОЇ ЧАСТИНИ

2.1 Аналіз інформаційного забезпечення системи

Інформаційне забезпечення системи – це комплекс заходів і процесів, спрямованих на забезпечення потрібної інформацією функціонування конкретної системи, організації або проекту. Ця інформація може включати в себе дані, знання, процедури та ресурси, які необхідні для виконання завдань системи або проекту.

Основна мета інформаційного забезпечення – забезпечити доступність, цілісність, конфіденційність та якість інформації, необхідної для прийняття управлінських рішень, виконання операцій та досягнення поставлених цілей.

У контексті інформаційного забезпечення системи або даних, цілісність визначається як збереження всієї інформації в її первісному інтегральному стані, без будь-яких недозволених або несанкціонованих змін, пошкоджень або втрат.

Доступність інформаційної системи означає, що інформація та системи повинні бути доступними для користувачів, коли вони цього вимагають, і незалежно від можливих обмежень або непередбачуваних ситуацій.

З точки зору інформаційного забезпечення, конфіденційність означає, що інформація та дані повинні бути доступні лише тим особам або суб'єктам, які мають право на доступ до них, і не повинні бути доступні незаконно або іншим особам.

Якість інформації інформаційного забезпечення визначає, наскільки інформація відповідає вимогам користувачів та стейкхолдерів, і як точна, надійна та корисна вона є для прийняття рішень та виконання операцій.

Забезпечення стійкої роботи інформаційного забезпечення системи вимагає певного підходу підходу і розглядається як один із основних аспектів кібербезпеки та надійності. Для забезпечення надійності та стійкості інформаційної системи важливим рішенням є автоматичне відновлення у

випадку виявлення потенційних помилок. Тоді система має можливість виявити помилку, і тоді виконувати заходи для відновлення стабільної роботи.

У даній дипломній роботі стійка робота системи гарантується за рахунок простоти файлів та даних на виході. А якщо при генерації (рендерингу) файлу була виявлена помилка, то зі сторони клієнта сторінка не зникне. У цьому випадку клієнт матиме «стару» сторінку, тобто сторінку що збереглася до ререндерингу. Такий метод задовільнить потреби клієнта та забезпечить адміністратора часом для усунення помилки.

Контроль інформаційної системи – це процес визначення, впровадження та виконання методів і проведення заходів для забезпечення надійності, безпеки та ефективності функціонування інформаційної системи. Цей процес включає в себе різноманітні аспекти, включаючи технічні, організаційні та процедурні заходи забезпечення безпеки та правильності роботи системи. Основна мета контролю інформаційної системи - це запобігання можливим загрозам, ризикам та помилкам, які можуть виникнути в процесі роботи системи, і забезпечити її безперебійну та безпечну роботу.

Контроль інформаційної системи виконується через наступні етапи та методи:

1. Проводення аналізу ризиків, який допомагає визначити потенційні загрози та уразливості інформаційної системи. Цей етап допомагає ідентифікувати потенційні небезпеки і визначити, які контрольні заходи потрібно впровадити;
2. Розробка політик, процедур та стандартів безпеки, які визначають правила та вимоги для забезпечення безпеки інформаційної системи;
3. Використовлення технічних засобів, таких як антивіруси, захист від вторгнень, файрволи, шифрування та інші технології для захисту інфраструктури системи від зовнішніх атак та загроз;
4. Встановлення механізмів контролю доступу, які визначають, які користувачі мають право на доступ до конкретних ресурсів і функцій системи;

5. Ведення моніторингу та аудиту системи для виявлення та реагування на небажані події;
6. Навчання персоналу системи щодо політик безпеки та процедур забезпечення безпеки, а також навичок реагування на інциденти;
7. Регулярна перевірка та оцінка заходів безпеки та контролю, для того щоб впевнитися в їхній ефективності та актуальності;

За для зручності керування та контролю над інформаційною системою у дипломній роботі передбачене розділення по ролям. Це дозволяє призначити кожному користувачу, або групі користувачів певні функції, можливості та доступ до певної інформації та даних.

Для протидії несанкціонованому доступу передбачено реєстрацію, авторизацію та автентифікацію користувачів. Дані усіх користувачів будуть зберігатися та шифруватися на сервері окремо.

Веб-гнучкість (гнучкість сайту) – це здатність веб-сайту до адаптації, пристосування до різних умов, до пристроїв, до роздільної здатності екрану, потреб та вимог користувачів. Гнучкість сайту залежить від таких складових:

1. Адаптивний дизайн.
2. Підтримка різних браузерів.
3. Підтримка різних платформ.
4. Швидкість завантаження.
5. Зручність навігації.

Веб-гнучкість сайтів, що побудовані на даній технології залежить лише від вмінь розробника. Перевагою даного методу є те, що при його правильному налаштуванні та програмуванні, метод вдасться інтегрувати у будь-яку інформаційну систему.

Також важливим аспектом даної інформаційної системи є мінімізація введення та виведення інформації. Прикладами мінімізації є зжимання картинок на сайті, мінімізація css та js файлів, тобто зменшення об’єму файлів

шляхом прибирання пробілів та зведення усього файлу в одну стрічку для компактнішого зберігання.

2.2 Розробка комбінованого методу SSG та SSR

Однією із задач, поставлених у даній дипломній роботі є розробка методу рендерингу, представленого комбінацією двох уже існуючих методів, що буде поєднувати переваги цих методі і максимально перекривати їх недоліки.

Почнемо з опису методу SSR або Server Side Rendering (рендеринг на стороні сервера). Цей тип генерації HTML є серверною обробкою всього коду, і увесь громіздкий код DOM (Document Object Model) доставляється в інтернет-браузер користувача. Іншими словами, код, який сьогодні найчастіше пишеться на JS, компілюється в html на одразу і доставляється в браузер як є, з усіма тегами і класами. Після того, як HTML-сторінка доставлена, DOM також завантажується з обробкою подій JS тощо.

Перевагами такого методу є:

1. Клієнту (браузеру) не потрібно чекати, поки JS відобразить HTML (DOM).
2. На клієнті (в браузері) не потрібно чекати, поки завантажиться весь JS, який в свою чергу почне рендеринг DOM (HTML).
3. Миттєве відображення сторінки (DOM/HTML) на стороні клієнта (браузера) [відразу після її відправки клієнту (браузеру) з сервера за умови, що всі файли, що блокують відображення (JS, CSS, шрифти, Media) також відправляються.

Метод SSG – метод рендерингу на стороні сервера. Тобто генерація сторінок на стороні сервера (а саме статичних, які потім просто зберігаються, і видаються сервером, якщо їх запитують). Цю техніку також прийнято називати JAMStack. Вона дозволяє генерувати статичні сайти, це також означає, що генерація файлів сайту (html-сторінок, JS-файлів та інших даних

[IMG...]) відбувається лише один раз, коли цього вимагає програміст або адміністратор проекту. Після цього на сервер завантажуються всі файли, необхідні для виконання сайту. Бекенд, або в цьому випадку краще написати CMS, може бути яким завгодно, певні скрипти потім будуть генерувати HTML-сторінки, які потім розміщуються на сервері і просто видаються користувачеві за запитом, без будь-якої бекенд (з точки зору генерації сторінок) логіки.

Перевагами даного методу є:

1. По суті, ви можете один раз створити собі сайт і завантажити його на сервер, де є тільки Nginx. Тоді вся магія буде в JS, в залежності від обраної вами технології.
2. Здебільшого 100% показників знаходяться в LightHouse (інструмент для тестування веб-додатків). Звичайно, із застереженням, що ваш сайт має досить оптимізований код JS, CSS, HTML і т.д.
3. Немає необхідності писати будь-яку логіку backEnd. За винятком випадків, коли користувач взаємодіє з вашим сайтом (отримання даних з форм, розсилка листів і т.д.), то вам потрібно буде прописати логіку бекенда.
4. Простота створення PWA (прогресивного веб-додатку). Багато інструментів (з наведених вище) дозволяють швидко і легко створити PWA.

Принцип комбінованого методу полягатиме у спрощенні архітектури сервера та поліпшенні оптимізації віддачі сторінок клієнту. Під час розробки клієнтської частини HTML-файли зберігатимуться, і при отриманні спеціального запиту сервер проводитиме рендеринг на своєму боці для оновлення інформації, а потім зберігатиме готові файли в умовний кеш [13]. Це дозволить уникнути повного рендерингу всіх сторінок при отриманні

запиту на них. Коли користувач звертається до сторінки, йому одразу подається готова HTML-сторінка без різноманітних перерендерингів.

2.3 Перспективи використання комбінованого методу з напрямку SEO

SEO, або пошукова оптимізація (Search Engine Optimization) – це комплекс заходів і стратегій, спрямованих на покращення видимості веб-сайту в результатах пошукових систем. Основна мета SEO – зробити веб-ресурс більш привабливим для пошукових систем, щоб забезпечити вищий рейтинг і вивести його на перші сторінки результатів пошуку [14].

Щоб розміщувати сайти вище або нижче, пошукові системи використовують не випадкові механізми, а чіткі алгоритми, які базуються на оцінці різного роду факторів. Всього таких факторів існує від 200 до 500, в залежності від конкретної пошукової системи (точна інформація залишається закритою).

Фактори впливуна SEO можна розбити на декілька основних груп:

1. Текстові фактори. Якість і унікальність контенту на сайті, на сторінках статтей, послуг і картках товарів. Їх оптимізація під ключові слова, інформативність і повнота розкриття теми.
2. Фактори внутрішньої якості. Швидкість завантаження сторінок, відсутність технічних помилок, якість адаптації під мобільні пристрої, відсутність дублів і т. д.
3. Зовнішні сигнали. Зворотні посилання, які ведуть з різних джерел та їх авторитетність. Вони бувають різного рівня ваги. Якщо про Ваш сайт позитивно відгукнеться мер міста? А якщо звичайний перехожий? Це різний рівень авторитету, але кожен з них важливий і потрібний.
4. Поведінкові фактори. Це набір метрик, які дозволяють пошуковій системі зрозуміти ступінь задоволеності сайтом з боку користувача. Скільки часу люди проводять на сайті, скільки сторінок

переглядають, чи не повертаються назад до пошукової видачі. Просування навіть ідеального в технічному плані сайту буде складним, якщо поведінкові фактори погані.

Комбінований метод SSG та SSR напряму впливає, а саме поліпшує фактори внутрішньої якості SEO. Це поліпшення відбувається тому, що використовуючи цей метод сторінки сайту кешуються на сервері уже готовими, і коли пошуковий робот знаходить сайт і посилає запит на отримання сторінки, то сервер не витрачає час і ресурси на рендеринг нової сторінки, і тому швидкість завантаження сторінки значно прискорюється.

2.4 Розробка загальної моделі системи

Будь-яка інформаційна система має 3 головних складових. Цими складовими є: серверна частина, адміністративна частина та клієнтська частина. У даному програмному рішенні також буде задіюватися відкрита стороння база даних.

Серверна частина інформаційної системи відіграє ключову роль у забезпеченні функціонування та обслуговуванні цієї системи. Вона відповідає за обробку запитів користувачів, зберігання даних, взаємодію з клієнтськими пристроями та забезпечення безпеки і доступності системи.

Інформаційна система зазвичай використовує сервери для обробки запитів. Сервери можуть бути фізичними апаратними пристроями або віртуальними машинами. Вони відповідають за виконання операцій, обробку даних та взаємодію з базами даних.

Для забезпечення взаємодії між серверами та клієнтами потрібно реалізувати мережеву інфраструктуру. Реалізується вона за допомогою комутаторів, маршрутизаторів, файерволів та іншого мережевого обладнання.

Бази даних відіграють роль простору для зберігання та управління даними. Але важливо правильно спроектувати їх структуру та попіклуватись про безпеку цих баз (рис. 2.1).

Безпека серверної частини надважлива для захисту даних та запобігання несанкціонованому доступу. До заходів запобігання можна віднести встановлення брандмауерів, використання автентифікації та авторизації користувачів, а також шифрування даних.

У проекті реалізована взаємодія веб-ресурсу з відкритою базою даних, для отримання з неї переліку ігор та докладної інформації про ці ігри (рис. 2.2).

Адміністративна частина інформаційної системи відіграє важливу роль у керуванні та управлінні системою. Ця частина дозволяє адміністраторам та управлінцям ефективно контролювати та підтримувати систему, забезпечуючи безпеку, надійність та ефективність її роботи.

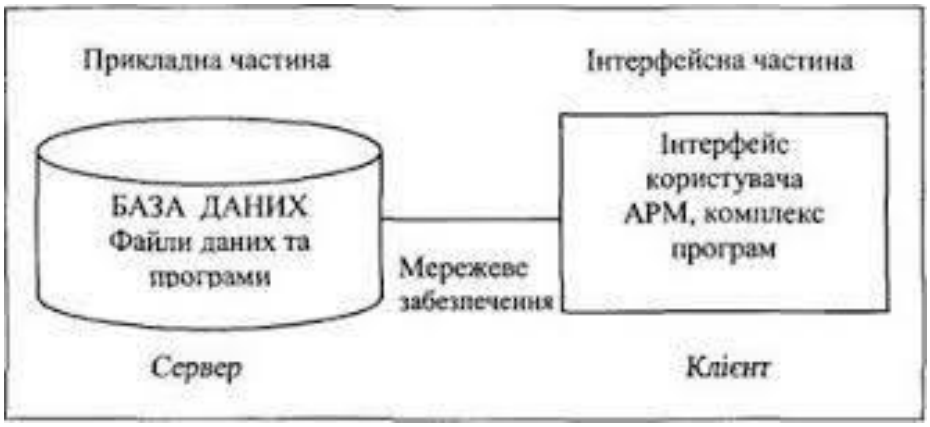


Рисунок 2.1 – Загальна модель системи

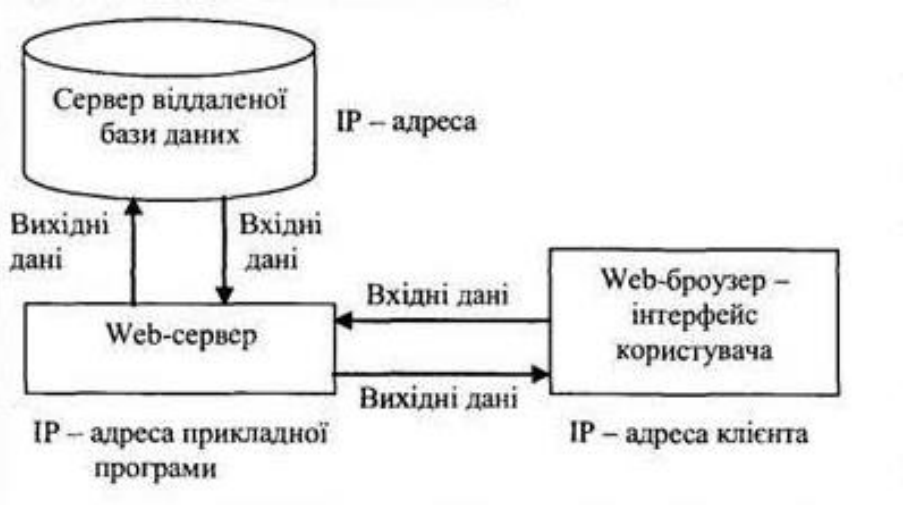


Рисунок 2.2 – Повна модель інформаційної системи

Система повинна мати механізми аутентифікації та авторизації для адміністраторів. Це дозволяє визначити, хто має право доступу до адміністративних функцій.

Адміністратори, в свою чергу, повинні мати можливість створювати, блокувати, редагувати та видаляти облікові записи користувачів, а також надавати їм ролі та права доступу до різних частин системи. Також вони повинні мати інструменти для моніторингу стану системи, збору даних про навантаження, продуктивність та безпеку. Аналіз цих даних допомагає вчасно виявляти проблеми та вдосконалювати роботу системи.

Адміністратори мають змогу керувати роботою серверів та програм, включаючи можливість запускати та зупиняти сервери, додатки та інші компоненти системи. Вони повинні встановити та керувати заходами забезпечення безпеки, включаючи брандмауери, антивірусне програмне забезпечення, моніторинг безпеки.

Важливо мати засоби для регулярного створення бекапів даних і можливість їх відновлення у випадку втрати або пошкодження даних.

Адміністративна частина інформаційної системи допомагає забезпечити ефективну та безпечну роботу системи, а також дозволяє забезпечити відповідність системи бізнес-вимогам і стандартам безпеки.

Клієнтська частина інформаційної системи представляє собою інтерфейс, через який користувачі взаємодіють з системою. Ця частина системи зазвичай включає в себе різні програми, додатки, веб-сайти або інші інтерфейси, які надають користувачам доступ до функцій і можливостей системи.

Для взаємодії користувача з системою клієнтська частина має графічний інтерфейс, включаючи кнопки, меню, форми та інші елементи керування. Також клієнтська частина може бути реалізована як веб додаток, який користувач відкриває у браузері, або у вигляді мобільного додатку, який завантажує користувач на свій телефон або планшет.

Для користувачів з різних регіонів та мовних спільнот може бути важливою можливість локалізації інтерфейсу системи, яка надається на різних мовах.

Клієнтська частина може включати в себе механізми автентифікації, які вимагають від користувачів введення ім'я користувача та пароля, а також систему авторизації для керування правами доступу користувачів.

Клієнтська частина інформаційної системи грає важливу роль у взаємодії користувачів з системою та впливає на їхню ефективність та задоволеність використанням системи. Тому важливо ретельно розробити цю частину системи, забезпечуючи зручність, безпеку та функціональність.

2.5 Розробка алгоритму роботи серверу

Для початку на сервері встановлюється програмне забезпечення, необхідне для роботи веб-ресурсу, також адміністративна та клієнтська частина веб-ресурсу інсталюють усі необхідні залежності для своєї роботи.

Наступним кроком є встановлення та конфігурація бази даних MongoDB, складається цей етап з встановлення бази даних та створення самої бази та налаштування доступу до неї.

MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів [15].

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій.

Для отримання контенту та детальної інформації про ігри необхідно отримати ключ доступу для підключення адмін-панелі до відкритої бази даних ігор IGDB.

IGDB перераховує подробиці про відеоігри та їхні компанії, знімальну групу та акторський склад. Подібно до Internet Movie Database від Amazon, контент IGDB орієнтований на користувача, дозволяючи зареєстрованим користувачам оцінювати, складати список та переглядати ігри. Користувачі також можуть редагувати та створювати сторінки, які публікуються після перевірки співробітниками IGDB[16].

Після цього адміністратор має можливість корегувати контент з IGBD або додавати новий контент на сторінку, використовуючи відведені поля.

Коли поля заповнені, та адміністратор натиснув кнопка «Зберегти», генерується запит на збереження на сервері уже у базу даних MongoDB.

Сервер перевіряє, чи дані в полях відповідають формату та логіці веб-ресурсу. Якщо дані відповідають вимогам, то вони успішно зберігаються в базі даних MongoDB. Інакше, сервер повертає повідомлення про помилку до адмін-панелі.

Підчас рендерингу сторінки, сервер надсилає запит на отримання даних з бази даних MongoDB і поміщає контент у шаблони, що створені в клієнтській частині. При відсутності помилок, сервер зберігає заповнені шаблони, які з клієнтської сторони стають готовими сторінками з контентом. Якщо виникла помилка, то сервер не зберігає відрендерину сторінку, а залишає сторінку, що була відрендерина попередньо до змін.

2.6 Розробка принципів роботи адмін-панелі

Більшість сайтів, веб-систем або веб-додатків мають адмін панель, доступ до якої здійснюється за допомогою авторизації користувача. Людина, відповідальна за редагування та наповнення сайту інформацією вводить за відповідним веб-посиланням ім’я користувача і пароль, і отримує доступ до CMS Більшість адмін панелей мають типову функціональність, і для

прискорення розробки цифрових рішень зручно використовувати гнучкі багатофункціональні адмін панелі, такі як:

- Admin LTE
- Devias Kit
- WordPress
- ArchitectUI

Сучасний підхід до створення сайтів і web-додатків дозволив подолати згадані обмеження. Тепер прийнято використовувати готові шаблони адмін панелей на базі фреймворків. Вони дають можливість реалізувати весь необхідний для управління контентом функціонал з урахуванням необхідної якості UX / UI-дизайну.

Сучасна адмін панель сайту повинна мати привабливий і інтуїтивно зрозумілий зовнішній вигляд, адаптивно працювати під управлінням різних браузерів на різних операційних системах і відкриватися з різних пристроїв.

До того ж важливо, щоб на проєктування і створення адмін панелі сайту не витрачалися зайві ресурси, адже типові завдання повинні мати типове рішення.

Це чудове рішення являє собою набір шаблонів і елементів інтерфейсу, які підходять для адміністрування будь-яких цифрових проєктів, включаючи веб-системи для управління бізнесом. За допомогою адмін панелі на базі Admin LTE можна адмініструвати сайти, портали, CRM- і ERP-системи, сервери мобільних додатків, багатофункціональні веб-системи та інші цифрові рішення.

За допомогою адмін-панелі адміністратор може:

1. налаштувати структуру сайту;
2. змінити зовнішнє оформлення сайту;
3. додати та редагувати контент;
4. створити додаткових користувачів і налаштувати права доступу.

Великим проривом у технології створення та розробки сучасних вебпроектів є створення системи керування вмістом (CMS, Content Management System) – це інформаційна система або комп'ютерна програма, що використовується для забезпечення та організації спільного процесу створення, редагування та управління контентом [17].

Якщо раніше більшість сайтів були статичними і вимагали внесення змін в їх вміст вручну, зараз динаміка розвитку проектів вимагає готовності швидко реагувати на зміни і впроваджувати їх з максимальною оперативністю. При цьому не усі користувачі хочуть або можуть собі дозволити звертатися до розробників, особливо якщо сайт вимагає постійної роботи над ним.

Сучасні системи керування вмістом широко використовуються на просторах Інтернету при створенні проектів будь-якої складності, при цьому не потребуючи знань в HTML, CSS та інших областях веб-програмування, що є їх головною перевагою. Також дані системи дають можливість швидкого, простого та інтуїтивного додавання, видалення, редагування та форматування контенту, що значно спрощує та полегшує завдання адміністрування сайту.

З використанням CMS можливе не лише додавання текстового контенту, а й різноманітного мультимедійного матеріалу, що дозволяє значно урізноманітнити сайт. Також системи керування вмістом автоматично генерують панель адміністратора, яка зачіпає всі сфери роботи сайту, що дуже зручно та практично,

Всі системи керування вмістом можна умовно розділити на чотири види:

- З відкритим кодом: WordPress, OpenCart, Joomla!, Drupal, Magento, PrestaShop [18];
- Коробкові: Tilda, Wix, SitePro, Shopify, Squarespace;
- Самописні - розробляються на замовлення під конкретний проект;
- Фреймворки: Laravel, Ruby on Rails, Django.

API, або інтерфейс прикладного програмування, — це набір визначених правил, які дозволяють різним програмам взаємодіяти один з одним. Він діє як проміжний рівень, який обробляє передачу даних між системами, дозволяючи

компаніям відкривати свої дані додатків і функціональність зовнішнім стороннім розробникам, діловим партнерам і внутрішнім відділам своїх компаній.

У даній дипломній роботі розроблена самописна система керування вмістом. Використовуючи розроблену адмін панель, адміністратор надсилає запити на збереження елементів, їх видалення, редагування, або отримання усіх елементів бази через API на сервер, після чого у відповідь на post-запити буде виведений статус «Success», або «Failure», а у відповідь на get-запит – інформація про елемент, або елементи бази даних.

Система передбачатиме реєстрацію користувача, його автентифікацію та авторизацію.

Реєстрація – це процес документального фіксування інформації про особу, об'єкт або подію в спеціальному реєстрі, базі даних, журналі або іншому системному засобі для подальшого контролю, ідентифікації або ведення обліку [19].

Автентифікація - це процес перевірки ідентичності особи, користувача, пристрою або програми для визначення, чи має вони доступ до певних ресурсів, послуг чи інформації. Цей процес спирається на представлення об'єктом (особою, користувачем, пристроєм) чогось, що підтверджує їхню ідентичність, такого як пароль, біометричні дані (наприклад, відбитки пальців або розпізнавання обличчя), фізичний об'єкт (наприклад, ключ), або інші ідентифікуючі фактори [20].

Авторизація - це процес надання права доступу користувачеві, пристрою або програмі до певних ресурсів, функцій або послуг. Після того, як користувач (або інший суб'єкт) успішно автентифікується, тобто підтверджує свою ідентичність, авторизація визначає, які конкретні дії чи ресурси цей суб'єкт має право використовувати. Основна ідея авторизації полягає в тому, що після визначення ідентичності користувача через автентифікацію, система визначає, які права, привілеї чи ролі у цій системі має цей користувач. Це може означати, що користувач має доступ до певних функцій програми, файлів,

документів або ресурсів, а також може визначати рівні доступу до цих ресурсів [21].

Адмін-панель матиме сепарацію по ролям. Сепарація по ролям – це модель керування доступом, яка базується на призначенні доступу до ресурсів користувачам на основі їхніх ролей в організації чи системі. У цій моделі користувачі наділяються ролями, а ці ролі визначають, які дії, ресурси і функції вони мають право використовувати в системі.

Ролі визначають набір повноважень і доступів, які пов'язані з конкретною функцією або позицією в організації. Наприклад, в системі можуть бути ролі, такі як "адміністратор", "менеджер", "користувач", "гість". Адміністратори системи призначають користувачам певні ролі відповідно до їхніх обов'язків або потреб в системі. Для кожної ролі встановлюються правила і політики доступу, які визначають, які дії та ресурси доступні користувачам з цією роллю. Наприклад, адміністратор може мати повний доступ до всіх ресурсів, тоді як звичайний користувач може мати обмежений доступ.

Система повинна підтримувати можливість зміни ролей, призначення нових ролей та заборони ролей для користувачів відповідно до змін в організаційній структурі або потребах.

Система у даній дипломній роботі передбачає такі ролі, як (один) admin, editor (кілька – 2-3), reader та client.

Адміністративна частина матиме таку структуру:

1. Базові сторінки мають поля для вводу, що можна корегувати.
2. Вкладка пости, що мають поле заголовку, поле для картинки, поле опису, часу поста, також матиме можливість зберігати чорновик, редагувати пости, та публікувати їх.
3. Вкладка ігри, що мають поле заголовку, поле для картинок, поля для опису гри, час поста, зберігання чорновика та публікування, поле з кнопкою для пошуку гри по базі даних.

4. Кнопка для перерендерингу усього вмісту усіх сторінок.
5. Кнопка перерендериння на кожній окремій сторінці.

2.7 Розробка принципів роботи клієнтської частини

Клієнтська частина веб-ресурсу – це фронтенд, тобто частина веб-додатка або сайту, яка відповідає за відображення і взаємодію з користувачем у браузері або на мобільному пристрої [22].

Структура клієнтської частини включає в себе такі основні компоненти:

1. HTML.
2. CSS.
3. JavaScript.
4. DOM.
5. Фреймворки та бібліотеки.
6. Зображення.
7. Структура сторінок.
8. Кешування і оптимізація завантаження сторінки.

HTML визначає структуру веб-сторінки та вміст, який користувач бачить в браузері. Це включає в себе тексти, зображення, посилання, форми та інше.

CSS відповідає за зовнішній вигляд веб-сторінки, включаючи кольори, шрифти, розміщення елементів, анімацію та інші стилістичні аспекти. Він дозволяє задавати оформлення і макет сторінки.

JavaScript - це мова програмування, яка використовується для створення інтерактивних функцій та функціоналу на стороні клієнта. Він дозволяє реагувати на дії користувача, виконувати запити на сервер, маніпулювати DOM (Document Object Model) та інші функції.

DOM представляє структуру HTML-документа як об'єкт, з яким JavaScript може взаємодіяти. Він дозволяє змінювати вміст та структуру сторінки після завантаження в браузері.

Бібліотеки і фреймворки, такі як React, Angular, Vue.js використовуються розробниками для спрощення розробки фронтенда. Ці інструменти надають готові рішення для створення інтерфейсу взаємодії з сервером.

Веб-сторінки зазвичай включають в себе зображення, відео та інші мультимедійні ресурси, які відображаються в браузері.

Структура сторінок – це організація різних сторінок на веб-сайті, таких як головна сторінка, сторінки ігор, контактна інформація та інші.

Кешування і оптимізацію ресурсів використовують для покращення швидкості завантаження сторінки.

2.8 Висновки

У другому розділі докладно оглянуто ключові вимоги до системи, які будуть виконані на подальших етапах розробки проекту. Крім цього, надано детальний опис алгоритму роботи серверної частини веб-ресурсу, розкрито принципи функціонування адміністративної панелі та її приблизний вміст. У розділі також розглянуто структуру та принципи роботи клієнтської частини проекту, що визначають інтерфейс та взаємодію з користувачем.

Крім цього, в розділі подана загальна модель системи, яка охоплює важливі аспекти її функціонування та взаємодії між компонентами. Описана також модель системи для роботи з відкритою базою даних, що вказує на важливі аспекти зберігання та обробки інформації в системі.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СЕРВЕРНОЇ ЧАСТИНИ

3.1 Варіантний аналіз і обгрунтування вибору засобів реалізації програмного засобу

Перед початком розробки серверної частини веб-ресурсу про відеоігри необхідно обрати мову програмування, яку краще для цього застосувати. В зв’язку з необхідністю створити програму, що використовую функції Rest API для безпечного запуску користувацьких рішень, в якості мови програмування необхідно обрати ту, що підтримує такі виклики. Також мова програмування повинна мати зручний інтерфейс для роботи з протоколами мережі Інтернет для забезпечення комунікації з клієнтською частиною.

C# – це об'єктно-орієнтована мова програмування, яка розроблена корпорацією Microsoft. Вона входить до складу платформи розробки Microsoft .NET і є однією з найпопулярніших мов програмування для розробки додатків для Windows і веб-програм [23]. C# спроектована з використанням об'єктно-орієнтованого підходу, що дозволяє розробникам створювати програми, орієнтовані на об'єкти, що полегшує створення складних і структурованих додатків. Вона спроектована з урахуванням продуктивності і може виконувати оптимізації на рівні мови, що дозволяє створювати швидкодіючі додатки. C# підтримує асинхронне програмування і паралельні обчислення, що дозволяє створювати додатки, які працюють ефективно в умовах багатозадачності. Синтаксис C# є досить легким для читання і розуміння, що сприяє розробці і обслуговуванню коду. C# надає можливість створювати додатки, які наділені повною підтримкою Windows, включаючи доступ до API операційної системи. C# дозволяє легко розгортати додатки на різних платформах, включаючи Windows, Linux і macOS. Ця мова використовується для розробки різноманітних типів програм, включаючи десктопні додатки, веб-додатки, мобільні додатки, ігри, серверні додатки і багато інших. Вона є важливою мовою для розробників, які працюють у середовищі Microsoft, і має активну

спільноту розробників, яка підтримує розвиток мови і створення корисних інструментів та бібліотек.

C++ – це потужна мова програмування, яка комбінує в собі можливості високорівневої та низькорівневої мов програмування. C++ стала однією з найпопулярніших мов програмування в світі та використовується для розробки різноманітних програм і систем, включаючи десктопні програми, ігри, веб-додатки та вбудоване програмне забезпечення [24]. C++ підтримує об'єктно-орієнтований підхід, що дозволяє розробникам створювати класи та об'єкти для організації коду. Це сприяє створенню більш структурованих і підтримуваних програм. Вона дозволяє використовувати покажчики для безпосередньої адресації пам'яті, що робить її потужною мовою для розробки операційних систем, драйверів та інших системних програм. Мова C++ включає підтримку шаблонів, що дозволяє розробникам створювати загальні алгоритми та структури даних, що можуть бути параметризовані різними типами даних. C++ надає засоби для створення багатопотокових програм, що дозволяє використовувати багато потоків в одній програмі для поліпшення продуктивності. Вона підтримує розробку багатозадачних програм, які можуть одночасно виконувати кілька завдань або процесів. Мова є розширенням мови C і зберігає сумісність з нею, що дозволяє використовувати бібліотеки і код, написаний на C, у програмах на C++, а також має багатий набір бібліотек, які дозволяють розробникам використовувати готові рішення для різних завдань, включаючи роботу з текстом, мережами, графікою, базами даних і багато іншого. Завдяки своїм можливостям C++ широко використовується у різних галузях програмування, включаючи розробку ігор, наукових досліджень, вбудованих систем, фінансових програм та багато інших областей.

JavaScript – це мова програмування, яка використовується для розробки веб-додатків та динамічного веб-змісту. Вона є однією з найпоширеніших мов програмування в сфері веб-розробки і дозволяє створювати взаємодію на веб-

сторінках, анімацію, валідацію форм, обробку подій та багато інших речей, які забезпечують високу функціональність і взаємодію користувачів з веб-сайтами. JavaScript підтримується всіма сучасними веб-браузерами, і він виконується безпосередньо в них, що робить його ідеальним інструментом для веб-розробки. Мова дозволяє змінювати вміст та вигляд сторінки після завантаження без необхідності перезавантаження сторінки [25]. Це дозволяє створювати динамічні та інтерактивні веб-сайти. Вона взаємодіє з DOM, який представляє структуру сторінки, і дозволяє змінювати вміст, стилі та властивості елементів сторінки, також JavaScript дозволяє визначати обробники подій, такі як клік мишею, введення тексту, натискання клавіш і багато інших. Це дозволяє створювати відгуки на дії користувачів. JavaScript дозволяє виконувати асинхронні запити до сервера без перезавантаження сторінки, що дозволяє отримувати або відправляти дані на сервер без затримок. Вона дозволяє перевіряти введені користувачем дані на валідність перед відправкою форми на сервер. Існують багато сторонніх бібліотек та фреймворків для розробки веб-додатків на JavaScript, які спрощують і прискорюють роботу розробників. JavaScript дозволяє створювати анімаційні ефекти на сторінці, що поліпшує користувацький досвід. Також JavaScript може використовуватися для розробки веб-додатків на різних операційних системах та пристроях.

Для вибору розглянуто такі критерії, як наявність об’єктно-орієнтовної парадигми, типізація, використання пам’яті, ефективність, взаємодія з платформами і галузь використання, оскільки вони є найактуальнішими при виборі мови.

Для об’єктивного порівняння вище наведених мов програмування по визначеним критеріям зведемо усю інформацію у таблицю 3.1.

Таблиця 3.1 – Порівняння мов програмування

Характеристика	C#	C++	JavaScript (JS)
Стиль мови	Об'єктно-орієнтований	Об'єктно-орієнтований і процедурний	Об'єктно-орієнтований і функціональний
Типізація	Сильна, статична	Сильна, статична	Слабка, динамічна
Використання пам'яті	Автоматичне керування пам'яттю	Вручну керування пам'яттю	Вручну керування пам'яттю (з можливістю використання автоматичного сбору сміття)
Ефективність	Вищий рівень ефективності	Висока ефективність, але важко підтримувати	Зазвичай менш ефективний порівняно з C# і C++
Платформи	Головним чином Windows	Крос-платформеність, але робота з Windows	Веб-програмування, включаючи веб-браузери та сервери
Використання	Розробка Windows-додатків, геймдевелопмент, серверні додатки	Системне програмування, геймдевелопмент, вбудовані системи	Веб-розробка, клієнтська і серверна сторони, розширення веб-браузерів

З точки зору призначення даного проекту, мова програмування JavaScript має вагому перевагу над мовами C# та C++, тому для реалізації серверу та проекту в цілому.

Для вибору середовища розробки створення веб-ресурсу було проаналізовано такі середовища розробки, як Visual Studio Code та WebStorm.

Visual Studio Code – це безкоштовне, відкрите та легковаге середовище розробки, розроблене компанією Microsoft. Воно є одним з найпопулярніших текстових редакторів та інтегрованих середовищ розробки (IDE) серед розробників програмного забезпечення, оскільки поєднує в собі потужність та розширюваність із зручністю використання та широким спектром доступних розширень. VS Code є повністю безкоштовним програмним забезпеченням, і ви можете завантажити та використовувати його на різних операційних системах, таких як Windows, macOS та Linux. Код самого редактора також відкритий, і ви можете приймати участь у розвитку його додатків. Середовище підтримує безліч мов програмування та технологій, включаючи JavaScript, Python, Java, C++, PHP, Ruby, HTML/CSS, і багато інших. Ви можете налаштовувати його під ваші потреби, додавши розширення для конкретних мов або фреймворків [26]. VS Code відомий своєю потужною системою розширень. Ви можете встановлювати розширення, які додають новий функціонал до редактора, роблячи його ідеальним для вашого стилю роботи. Наприклад, ви можете встановити розширення для роботи з Git, для підтримки певної мови програмування, для роботи з базами даних, тощо. Visual Studio Code пропонує автодоповнення коду, що робить розробку більш продуктивною. Воно розпізнає контекст і пропонує вам можливі варіанти для методів, змінних, ключових слів тощо. Також реалізована можливість відлажувати свій код прямо в середовищі. Середовище підтримує різні розширення для відлагодження різних мов, такі як Python, JavaScript та інші.
Загалом, Visual Studio Code є потужним та дуже популярним редактором для розробників програмного забезпечення, який надає безліч можливостей та

розширень для зручної та продуктивної роботи над проектами різного роду (рис. 3.1).

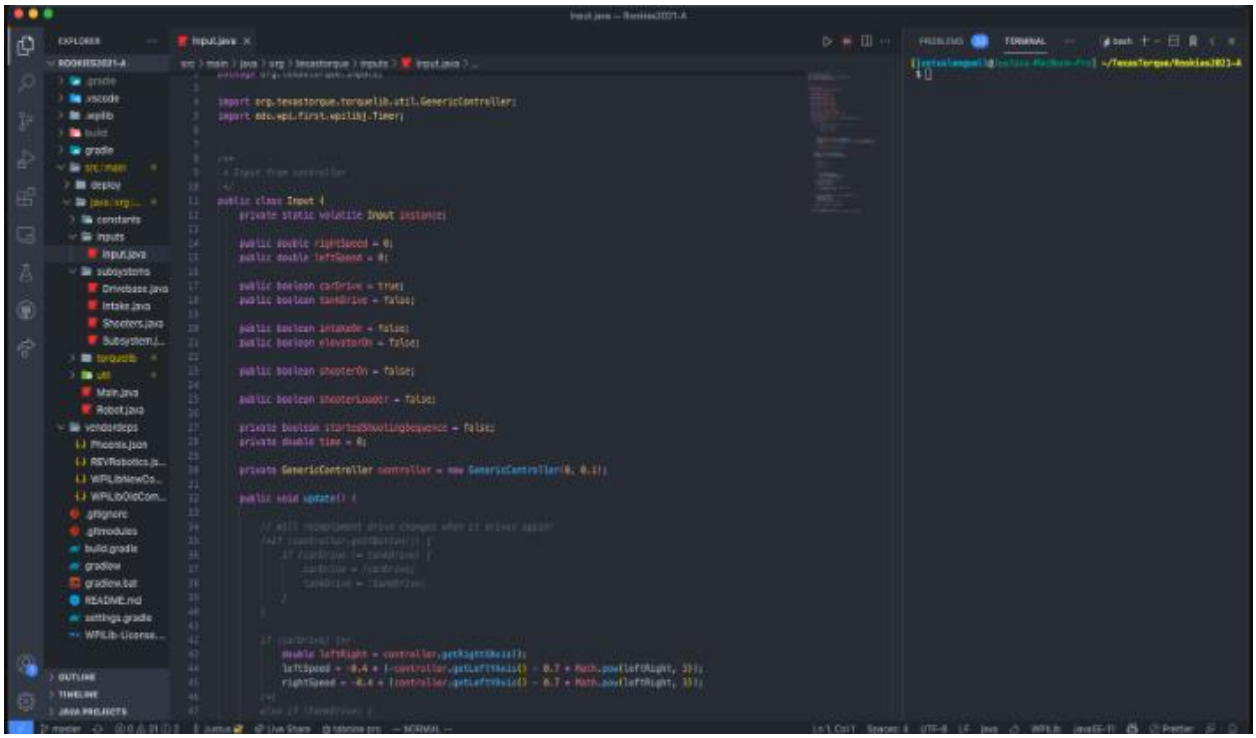


Рисунок 3.1 – Приклад роботи середовища програмування VS Code

WebStorm - це інтегроване середовищо розробки (IDE) для веб-розробки, розроблене компанією JetBrains. Ця IDE спеціалізується на підтримці різних мов та технологій веб-розробки, таких як HTML, CSS, JavaScript, TypeScript, Node.js, Angular, React, і багатьох інших. Вона є однією з найпопулярніших IDE для веб-розробки та використовується розробниками по всьому світу. Середовище програмування надає розширену підтримку HTML, CSS, JavaScript, TypeScript, і ECMAScript. Вона підтримує такі технології як JSX для реакту, Vue.js, і Angular. IDE надає підказки, автодоповнення, аналіз коду та багато інших корисних функцій для розробки веб-додатків [27]. WebStorm має потужний редактор тексту з можливістю розмітки синтаксису, відображення помилок, автодоповнення коду та підсвічування синтаксису. Це полегшує написання та редагування коду. Він має вбудовану підтримку систем контролю версій, таких як Git, SVN, і

Mercurial, дозволяє розробникам ведення та керування кодом проекту безпосередньо з інтерфейсу IDE, також надає потужні інструменти для налагодження JavaScript-коду, включаючи можливість встановлення точок зупинки, перегляду значень змінних та викликів функцій, а також відстеження викликів AJAX-запитів. Розробка серверної частини веб-додатків на Node.js також підтримується в WebStorm. IDE допомагає встановлювати залежності, виконувати сценарії npm, та налагоджувати серверну частину. WebStorm має можливість відлагодження JavaScript-коду в браузері за допомогою інтеграції з Chrome DevTools. Середовище інтегрується з популярними фронтенд-інструментами, такими як Webpack, Grunt, Gulp, і Bower, для автоматизації процесу розробки та збірки проекту. WebStorm дозволяє налагоджувати JavaScript-код на віддалених серверах через SSH, що дуже корисно для відладки в реальних середовищах. В цілому, WebStorm є потужним інструментом для веб-розробки, який надає широкий набір можливостей для створення веб-додатків та веб-сайтів. Її інтерфейс і функції можуть забезпечити зручну робочу обстановку та покращити продуктивність розробки (рис. 3.2).

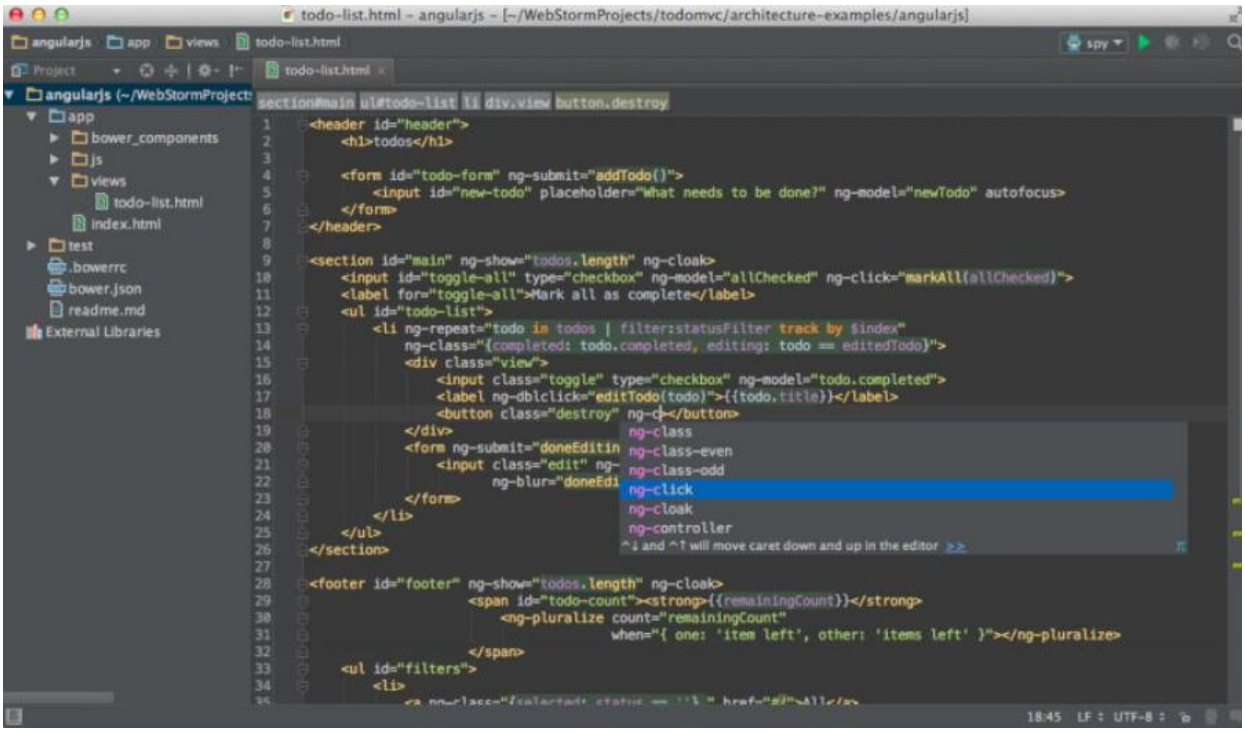


Рисунок 3.2 – Приклад роботи середовища програмування WebStorm

Порівняння розглянутих середовищ програмування за обраними критеріями наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння середовищ програмування

Характеристика	VS Code	WebStorm
Вартість	Безкоштовний з відкритим вихідним кодом	Платний, існує безкоштовна пробна версія
Розширюваність	Велика кількість розширень та плагінів	Включає в себе багато інтегрованих інструментів та плагінів
Мови програмування	Підтримує багато мов програмування, широкий вибір розширень	Основний фокус на підтримці JavaScript, але підтримує інші мови
Редагування коду	Добра підтримка автозаповнення коду, IntelliSense	Висока підтримка автозаповнення, інтеграція з інструментами від JetBrains
Відлагодження коду	Розширені засоби для відлагодження (за допомогою розширень)	Вбудований відлагоджувач з багатьма функціями
Інтеграція з системами керування версіями	Велика підтримка Git, плагіни для інших систем	Вбудована підтримка Git, Mercurial, та інших систем
Веб-розробка	Має багато розширень для веб-розробки, підтримка HTML, CSS, та JavaScript	Розроблений як інтегроване середовище для веб-розробки, з підтримкою Angular, React, інших фреймворків
Швидкість та ресурси	Легкий та швидкий, використовує менше ресурсів	Більше вимог до ресурсів, але відмінна продуктивність для великих проектів

Серед розглянутих середовищ програмування було обрано Visual Studio Code, оскільки воно безкоштовне, швидке та легке, о також має багато розширень для веб-розробки.

3.2 Розробка серверної частини веб-сервісу

Роботу серверу умовно можна поділити на кілька частин. Кожен файл проекту містить у собі один клас і кожен клас містить свій перелік методів.

У проекті сайту міститься файл конфігурації в якому вказані:

1) шлях до проекту;
2) шлях до згенерованого веб-сайту;
3) мод сесії;
4) хост;
5) номер порта;
6) назва локального сайту.

У головному файлі проекту відсутні класи, у ньому в основному зазначені роути та алгоритми роботи при різних запитах. Нижче представлено типи роутів та їх призначення (таблиця 3.3).

Таблиця 3.3 – Назва та призначення основних типів роутів

Приклад роуту	Тип і призначення
/render-pages	Викликає метод з іншого класу, який запускає метод для рендерингу відповідних сторінок (приймає в параметрах назву сайту та сторінки які потрібно згенерувати)(рис. 3.3)
/blog/games/top-games/	Звичайний роут без спеціальних параметрів для рендерингу, відкривається звичайна сторінка, яка була відрендерена раніше

Продовження таблиці 3.3 – Назва та призначення основних типів роутів

/admin/blog/	Роут до адмін панелі, алгоритм той же, що і при звичайному роуті, але відправляється до адмін панелі
/blog/games/top-games/?ssr=true	Роут із параметром для ререндерингу сторінки, викликає метод, який робить рендер сторінки по вказаному роуту
/blog/games/styles.css	Роут не на сторінку, роут веде до зображення, файлів стилів чи файлів скриптів

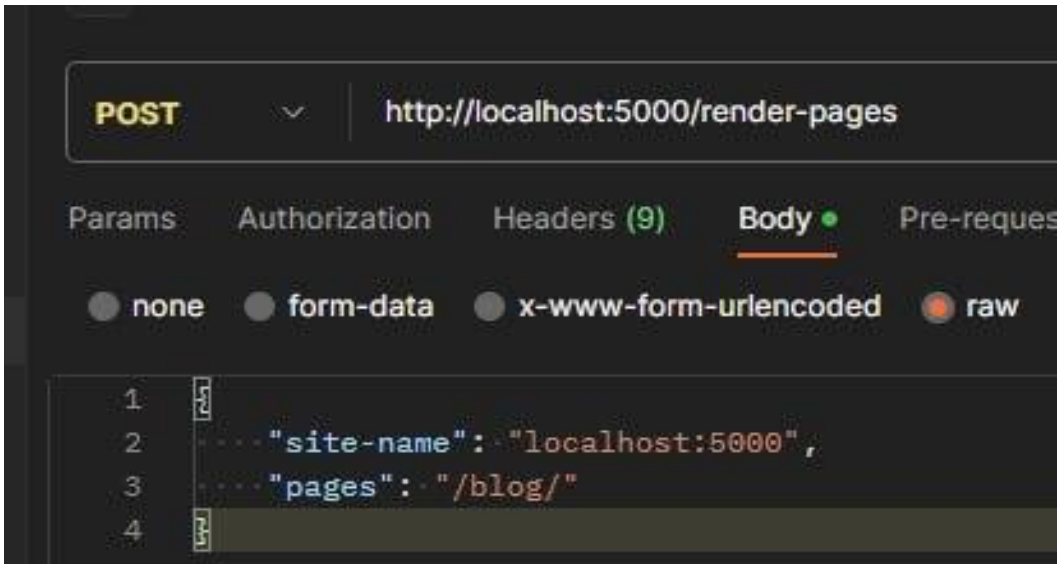


Рисунок 3.3 – Структура команди «/render-pages»

Алгоритм перевірки роутів відбувається за допомогою таких кроків:

1. Спочатку проводиться перевірка, чи є вказаний роут роутом для рендерингу сторінки. Якщо це так, то викликається метод для рендерингу звичайної сторінки. В іншому випадку аналіз продовжується.
2. Наступним етапом є превірка, чи це роут на сторінку, чи роут до адмін панелі. Якщо роут підходить під одни із двох варіантів, то на клієнта відправляється сайт, що був у запиті. В іншому випадку це означає, що це ніодин з цих варіантів, тоді це запит на файл.

3. Останньою є перевірка наявності файлу з запиту. Якщо файл є у наявності, то відбувається передача файлу на клієнтську частину. В іншому разі повертається сторінка 404, що свідчить про відсутність файлу із запиту.

Файл роутеру займається пошуком запитаного файлу та аналізом того що було запитано.

Один з методів в цьому файлі визначає що саме було запитано. Наприклад, це може бути роут на файл статті і виглядатиме він так: «/blog/games/top-games/». Оскільки вся технологія побудована на шаблонах, то у проекті немає такої структури файлів, але є реалізація у вигляді шляхів до шаблонів. Даний файл аналізує кожну частину роуту і шукає, які змінні йому підходять, тоді на виході цей роут буде виглядати так: «/blog/:category/:post/», у цьому випадку файл post і є шаблоном статті. Цей метод його знаходить та повертає. Також відбувається перевірка на наявність параметрів у роуті, і якщо ці параметри у наявності, то їх повертають у вигляді об’єкту.

Інший метод класу займається пошуком файлу у запиті. Якщо роут до файлу має параметр на ререндеринг, то відбувається перегенерація файлу, після чого він повертається. Якщо цей параметр відсутній, то повертається збережений раніше файл(рис. 3.4).

Файл зображень призначений для роботи з усіма зображеннями на сайті (рис. 3.5), і містить кілька методів:

1. Метод віддачі збереженого раніше зображення, цей метод включає перевірку наявності зображення та віддачу шляху до цього зображення.
2. Метод генерації зображення, що шукає зображення, що не було збережене раніше, в проекті та аналізує шлях по якому його потрібно зберегти та зберігає його. Після збереження файлу відбувається його конвертація у формат webp.
3. Метод зжимання зображень спрацьовую на кінцевому етапі для оптимізації, принцип полягає у тому, що якщо зображення за

розмірами більше за 600 або 1200 пікселів, то метод зберігає зберігає його копії з врізаними розмірами.

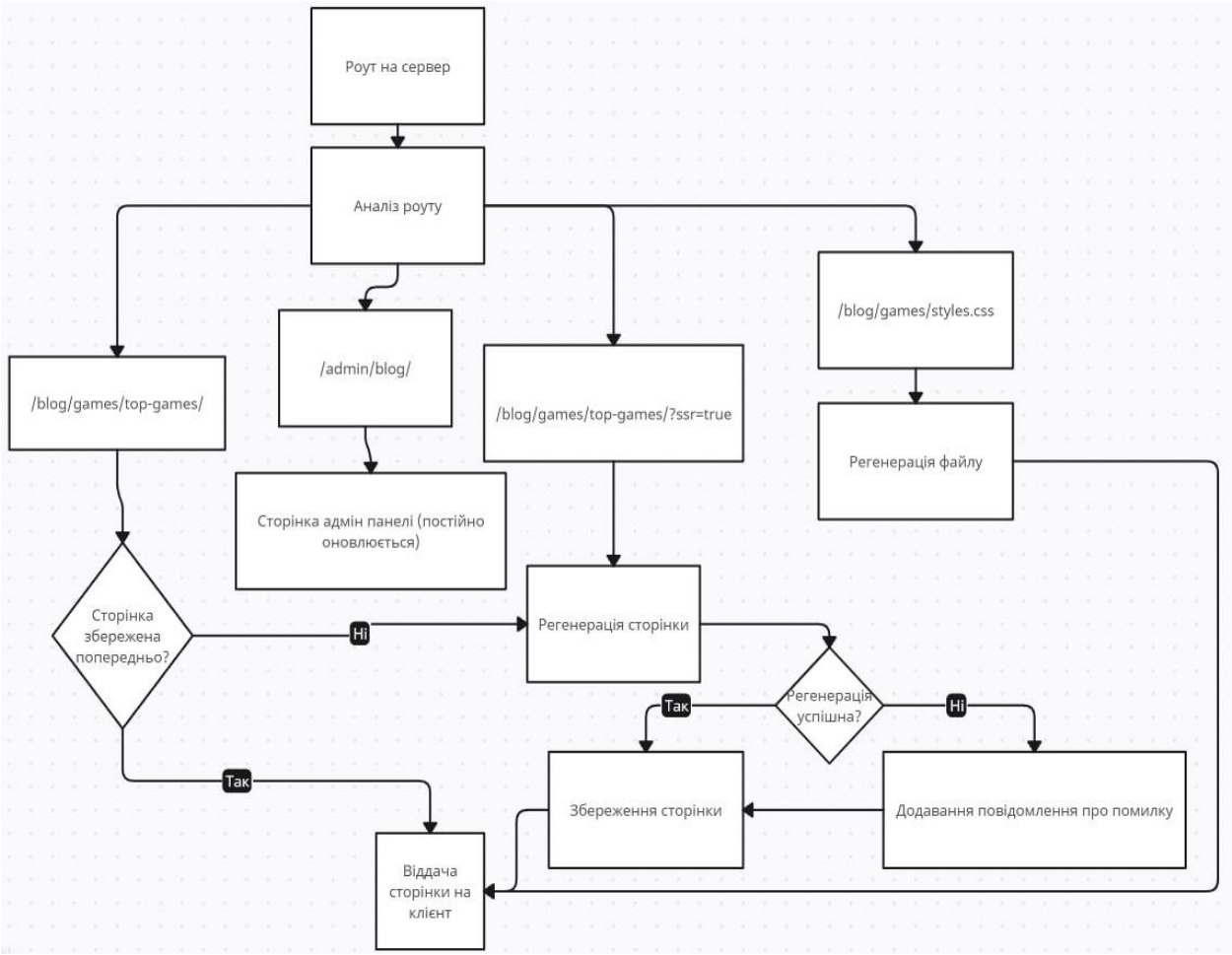


Рисунок 3.4 – Алгоритм роутингу та рендерингу сторінки

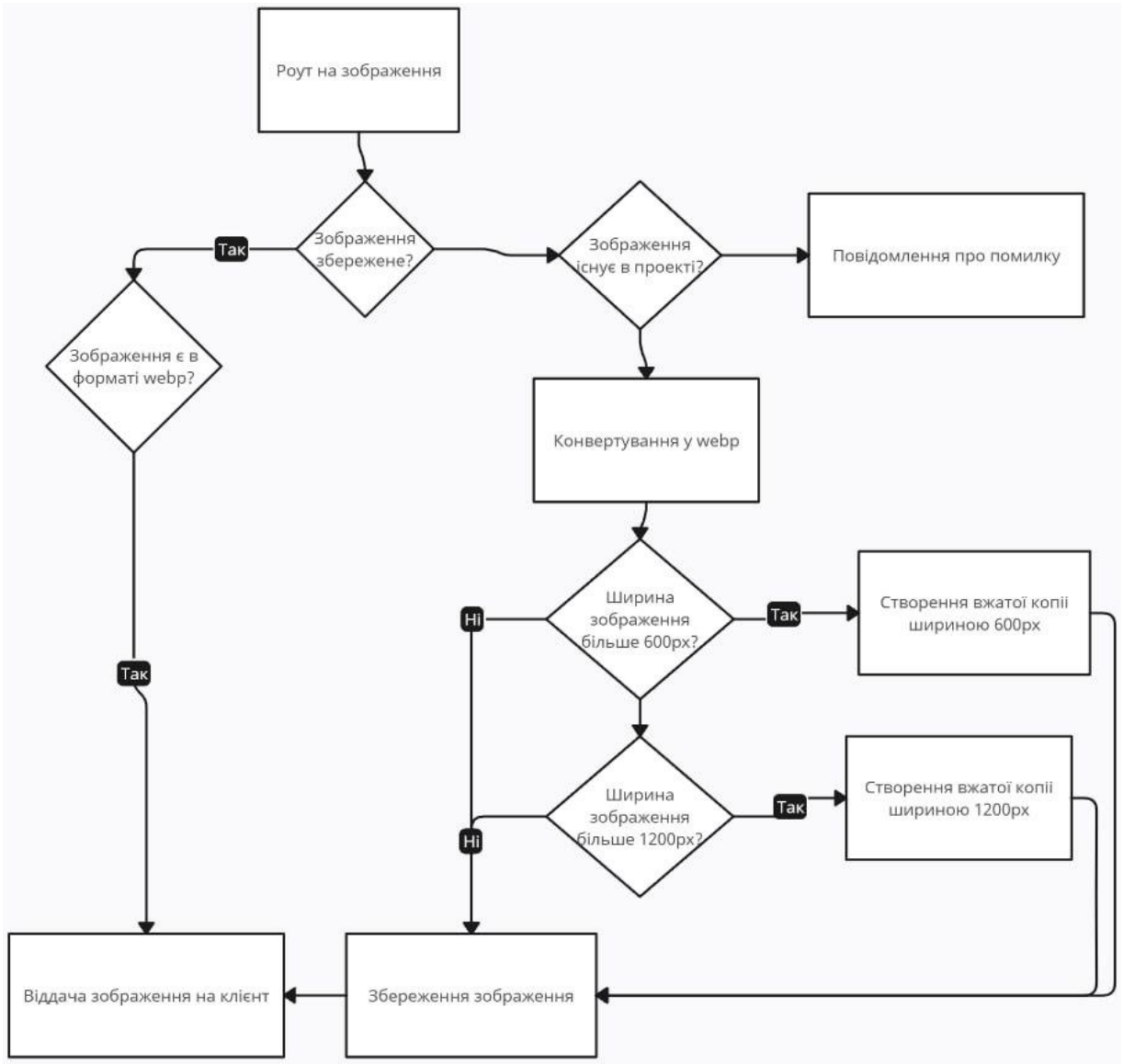


Рисунок 3.5 – Алгоритм роботи з зображеннями

WebP – це відкритий формат зображень, що призначений для стиснення зображень на веб-сторінках з метою поліпшення швидкості завантаження сторінок. WebP забезпечує високу стисненість та якість зображень, а також анімацію та велику кількість кольорів [28].

Файл SSR призначений для рендерингу, і включає в себе два методи:

1. Перший метод приймає назву сайту та масив з роутами, які потрібно перерендерити, після цього спрацьовує цикл, у якому до кожного роута шукається відповідний файл – шаблон, відправляється запит до бази даних для отримання даних по цій сторінці, а потім вставляє потрібний контент туди, куди необхідно. При успішному рендерингу

цей файл заміняє старий і зберігається, а у випадку виникнення помилки активною залишається стара сторінка, а у консоль виводиться повідомлення про помилку.

2. Другий же метод перевіряє чи існує директорія для файлу, який потрібно зберегти та зберігає його, а у випадку якщо такої директорії не існує, то створює її.

Файл SSG відповідає за віддачу попередньо збережених файлів за запитом, якщо такий файл відсутній, то повертає повідомлення про помилку.

Метод generatePage використовує бібліотеку “fs” – filesystem lib і приймає в себе аргумент “url”, сторінка генерується лише якщо параметр “ssr = true”. Також у цьому методі шлях до кореня проекту визначається функцією “path.resolve()”, і виглядає це так: “const __dirname = path.resolve()”. Крім цього визначається шлях від кореня проекту до кінцевого файлу функцією “path.join()”, що приймає агрументами корінь проекту та кінцеву URL-адресу файла, і виглядає так: “let filePath = path.join(__dirname, '/dist', url)”.

Функція “fs.readFile()” приймає агрументами шлях від кореня проекту– “filePath” та повідомлення результату виконання.

Функція “getData()” буферизує сторінку з формату html, у формат стрічки функцією “toString()”, а потім за допомогою функції “fs.writeFile()”, що приймає аргументами шлях до кінцевого файлу, кінцевий контент сторінки, кодування сторінки, і повідомлення при помилці.

3.3 Розробка клієнтської частини веб сервісу

В клієнтській частині веб сервісу створюється організована структура папок та файлів для управління сторінками веб-сайту. У даному випадку структура базуватиметься на директоріях і папках. Кореневою папкою буде папка “*src*”.

У папці “src” розташовані усі вихідні файли, в тому числі і файл «index.html». Цей файл є головною сторінкою веб-сайту, тобто стартовою сторінкою, яку бачать відвідувачі при першому відвідуванні (рис. 3.6).

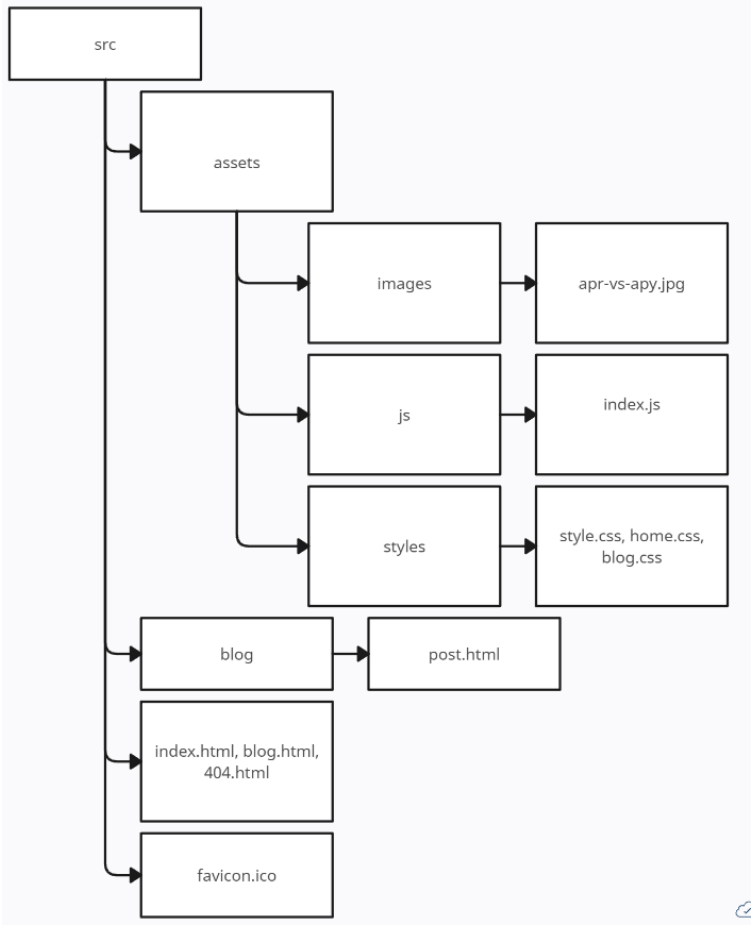


Рисунок 3.6 – Структура кореневої папки “src”

Структура директорій організована відповідно до роутів (маршрутів або URL-адрес). Це означає, що створені директорії відображають структуру маршрутів на веб-сайті [29].

Для прикладу, якщо роут (маршрут) представлений як “/blog/games/”, то структура, то структура директорій буде наступною: спочатку коренева папка “src”, далі папка «blog», наступною буде папка «category», і заключною ланкою маршруту буде файл «index.html», в цьому випадку файл «index.html» буде сторінкою, яку бачать користувачі за цим роутом. В результаті структура виглядатиме як “src > blog > category > index.html”

Кожен файл html проекту є шаблоном, в який поміщається вміст відповідно до конкретної сторінки

Якщо потрібно створити сторінку для роуту “/blog/”, то це можна зробити двома способами:

1. Створити окремий файл з назвою “blog.html”, який відображатиме сторінку “/blog/”. Тоді структура директорії для сторінки виглядатиме, як “src > blog.html”.
2. Створити папку “blog”, і уже в цій папці створити файл “index.html”. У цьому випадку “index.html” в папці «blog» відображатиме сторінку “/blog/”. Тоді структура директорії виглядатиме, як “src > blog > index.html”.

У проекті є прямі і не прямі шаблони. Прямі шаблони – це шаблони які створюються для кожної окремої сторінки, кожен такий шаблон має свій власний файл наприклад, шаблон для сторінки «Contact Us».

Непрямі шаблони – це більш універсальні шаблони, для прикладу, шаблон статей. Замість створення окремого html-файлу для кожної статті, використовується загальний шаблон, в який поміщаються дані з різних статей. У такому випадку один шаблон використовується для всіх статей, і дані у шаблони заповнюються динамічно.

В шаблонах використовується html розмітка, це означає, що в шаблонах визначена структура html-коду для сторінок, де буде відображатися контент. В html-коді шаблонів використовуються спеціальні атрибути, які вказують на те, куди вставляти контент. Для прикладу, “data-field” – це атрибут, який вказує, де розміщувати дані з бази даних.

Після створення необхідних шаблонів відбувається запуск команди “npm run build” – це команда для виконання збірки веб-ресурсу, вона запускає Webpack, інструмент для збірки веб-додатків.

Webpack об’єднує всі JavaScript-файли та стилі, а потім зберігає цей зібраний код і ресурси в папку “dist”, яка є папкою, що призначена для розповсюдження готового веб-ресурсу (рис. 3.7).

В результаті веб-сервер підчас своєї роботи переглядає папку “dist’ та використовує готовий код та ресурси для обслуговування запитів

користувачів. Якщо користувач запитує певну сторінку, то сервер надсилає відповідний html-файл і ресурси з папки “dist”.

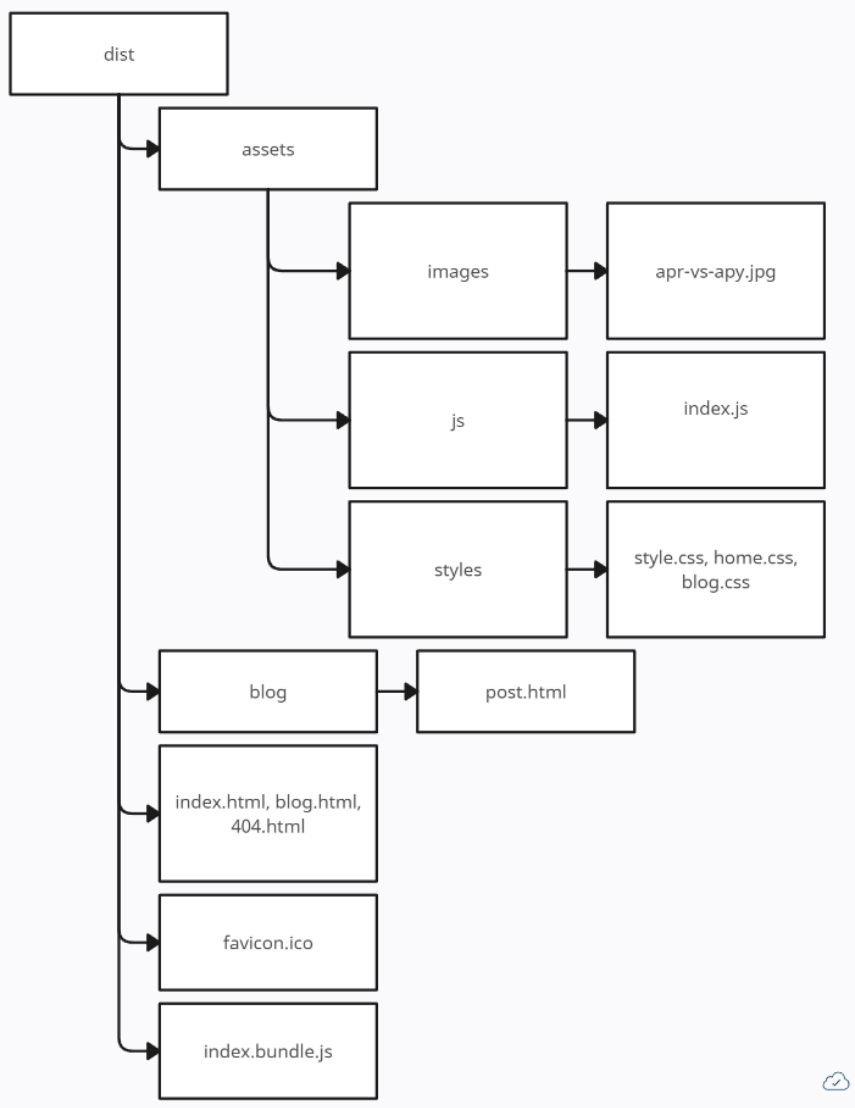


Рисунок 3.7 – Структура папки “dist”

3.4 Розробка адміністративної частини веб сервісу

Адміністративна частина веб-ресурсу розроблена з використанням фреймворку Vue.js. Vue.js підтримує розробку односторінкових додатків (SPA), а також може використовуватися для створення інтерфейсів на стороні сервера. Він інтегрується зі сучасними інструментами розробки, такими як Webpack та Babel, що дозволяє легко налаштувати робоче середовище [30].

Vue.js відповідає за оновлення DOM відповідно до заданого стану інтерфейсу, тобто розробнику не потрібно оновлювати DOM вручну. Цей

фреймворк побудований навколо концепції компонентів, які дозволяють розділити інтерфейс на невеликі і перевикористовувані частини. Це сприяє покращенню організації та підтримки коду. Також він надає можливість для оптимізації ресурсу такими методами як «lazy loading» та мемоїзація обчислень.

Lazy loading – це техніка оптимізації завантаження веб-сторінок або зображень. Замість того, щоб завантажувати всі ресурси (зображення, скрипти, стилі) одразу при завантаженні сторінки, ліниве завантаження дозволяє завантажувати ресурси тільки тоді, коли вони стають видимими на сторінці або активно використовуються користувачем. Це допомагає зменшити час завантаження сторінки і економить ресурси користувача, особливо на мобільних пристроях із повільним Інтернет-з'єднанням.

Адміністративна частина має головний файл, і є односторінковим додатком, тобто весь контент та навігація здійснюється без перенавантаження сторінки і весь вміст завантажується асинхронно. Цей файл містить компоненти, які забезпечують функціонал адміністративної панелі (рис. 3.8).

Компоненти панелі поділяються по призначенню на три типи:

1. Компоненти вкладки постів.
2. Компоненти для відображення вкладки ігор.
3. Компоненти звичайних сторінок.

Відображення ігор у вкладці ігри відбувається задопомогою умовного рендерингу. У Vue.js використовуються директиви умовного рендерингу компонентів та повторення їх в залежності від певних даних чи умов – v-if та v-for. Директива v-if та v-for - це два ключові атрибути, які використовуються в фреймворку Vue.js для управління відображенням та ітерацією елементів у веб-додатках.

Директива v-if - це конструкція в Vue.js, яка дозволяє умовно відображати або приховувати елементи в шаблоні на основі заданої умови. Застосовується директива v-if, коли потрібно визначити, чи має бути певний елемент відображений на сторінці в залежності від певної умови. Якщо умова

істинна, елемент буде видимим; в іншому випадку він буде прихованим (рис. 3.9).

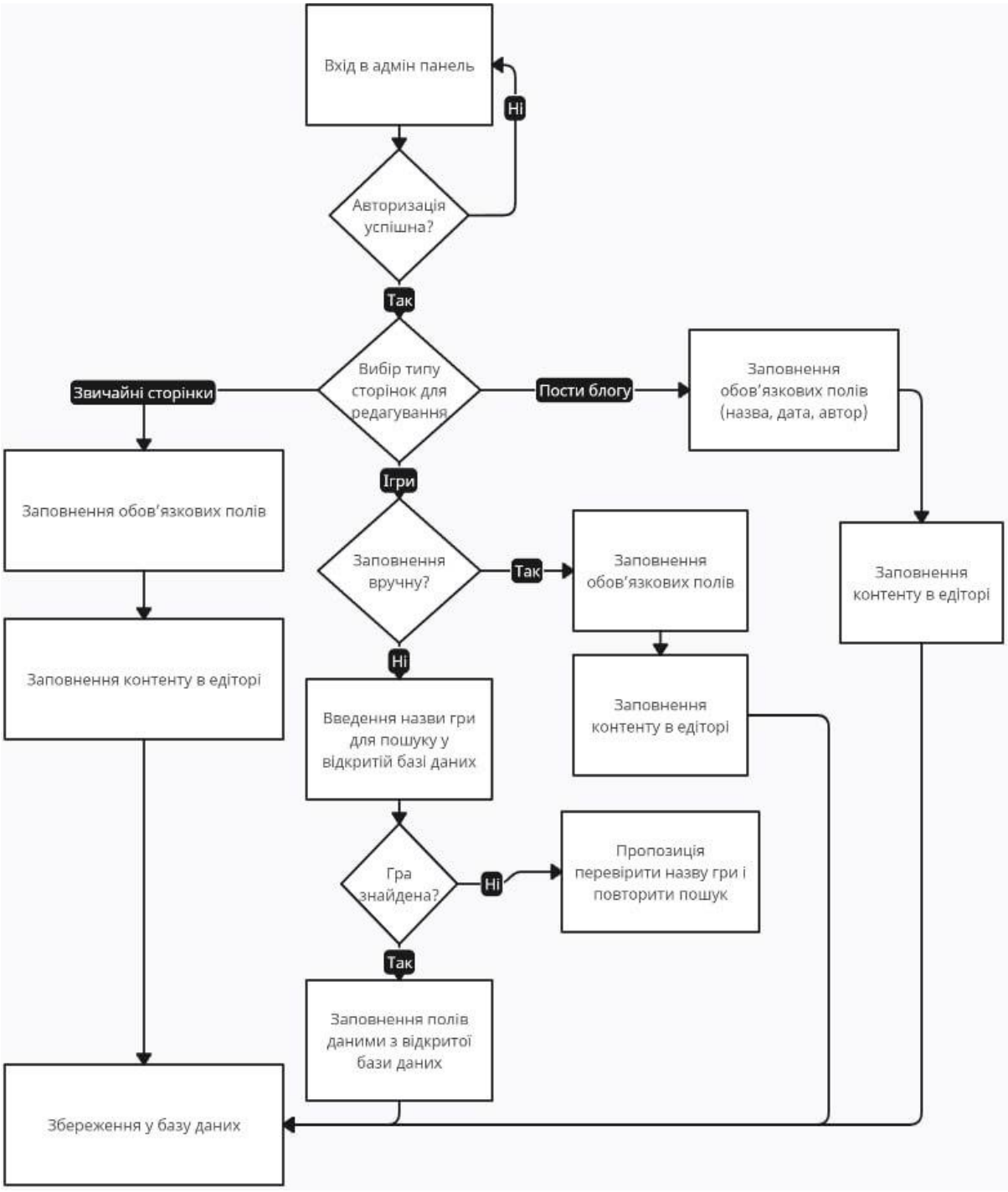


Рисунок 3.8 – Алгоритм роботи адмін-панелі

Директива v-for - це конструкція в Vue.js, яка використовується для ітерації по списку даних і створення повторюваних HTML-елементів з використанням цих даних. Застосовується ця директива, коли користувачу

потрібно створити багато однотипних елементів в шаблоні на основі даних, таких як масиви або об'єкти (рис. 3.10).

```
<template>
  <div>
    <p v-if="showElement">Цей елемент відображається</p>
  </div>
</template>
```

Рисунок 3.9 – Приклад використання директиви v-if

```
<template>
  <ul>
    <li v-for="item in items">{{ item }}</li>
  </ul>
</template>
```

Рисунок 3.10 – Приклад використання директиви v-for

Після входу в систему адмін-панелі, користувач має можливість перейти на будь-яку вкладку, наприклад вкладку з іграми. Коли користувач вибирає вкладку з іграми на сервер посилається запит, який взаємодіє з базою даних. Сервер надсилає запит до бази даних про наявність ігор та їх кількість, і тоді повертає об’єкт з усіма доступними іграми.

Отримавши об’єкт з іграми від сервера, дані циклічно обробляються в адмін-панелі та відображаються як список ігор.

При переході користувачем на якусь конкретну гру, відбувається запит на сервер, який в свою чергу надсилає запит до базу даних на отримання докладної інформації про цю гру. Дані повертаються з бази даних та відображаються на сторінці гри.

Якщо користувач зайшов на конкретну гру, або створює новий запис, то на сторінці з’являються поля для введення даних. У випадку якщо гра уже існує, то поля будуть заповненими, в іншому випадку поля будуть порожніми.

Коли користувач змінює дані, або додає нові дані у матеріали про гру та натискає кнопку «Зберегти», то до сервера відбувається запит з об’єктом, що містить дані з заповнених полів, і тоді сервер зберігає ці дані в базі даних для подальшого використання.

Метод “create()” аналізує тіло запиту на додавання поста до списку, перевіряє, чи заповнене тіло запиту даними, якщо ні, то система повідомляє про помилку. Також метод перевіряє чи заповнені усі поля поста, якщо не усі поля заповнені, то система повідомляє про відсутність параметрів. Якщо усі параметри відповідають вимогам, то метод перевіряє пост на наявність в базі по назві і перевіряє його по слагу. Якщо хоча б одна з перевірок визначила співпадіння, то відправляється повідомлення про існування такого посту, якщо ж такого поста ще немає, то створюється новий пост і зберігається у колекцію.

Метод “get()” дозволяє шукати пост у колекції по його назві, або слагу, у разі не коректного пошуку, система повідомить про відсутність даного поста, в іншому разі буде повернено сторінку з цим постом.

Метод “all()” повертає всю колекцію постів на даний момент.

Бібліотека “@tinymce/tinymce-vue дозволяє редагувати текст у тілі нового створюваного поста, наприклад шрифт, розмір шрифта, оформлення, колір шрифта, положення тексту.

3.5 Висновки

У третьому розділі було проведено аналіз різних мов програмування для розробки веб-ресурсу , а також середовищ програмування для його розробки.

У якості мови програмування було обрано мову Java Script для розробки веб-ресурсу, адже мова є об’єктно орієнтованою та забезпечить високу

швидкодію програмного рішення, також за допомогою асинхронності дозволить уникнути навантажень на систему.

В наслідок, було обрано використовувати середовище програмування Visual Studio Code в якості IDE. Дане середовище чудово підходить для програмування на Java Script, адже воно безкоштовне, використовує менше ресурсів ніж аналоги, тому є швидким та продуктивним, має хорошу підтримку автозаповнення коду та має велику кількість плагінів та розширень.

Було розроблено та реалізовано головний функціонал серверної частини веб-ресурсу, алгоритми роботи з даними та базами даних. Також було розроблено адмін-панель алгоритми роботи адмін-панелі, принципи роботи з даними і взаємодію адмін-клієнт і адмін-сервер. Розроблено клієнтську частину веб-ресурсу та принципи побудови сторінки на клієнті.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Опис методів тестування програмного забезпечення

Тестування програмного забезпечення призначено для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Аналіз результатів розробки ПЗ називається статичним тестуванням, а експлуатація програмного продукту називається динамічним тестуванням.

Розглянемо кілька типів тестування, які відрізняються рівнем знань про структуру програмного додатку.

Чорна скринька – це метод тестування, який не передбачає жодного знання про внутрішню архітектуру тестованого компонента або системи. Тестування з використанням техніки чорної скриньки передбачає написання тест-кейсів на основі аналізу функціональної чи нефункціональної специфікації компонента або системи, без знання її внутрішнього будови [31].

Метою цієї техніки є пошук помилок в таких категоріях:

1. неправильно реалізовані, відсутні функції;
2. помилки в структурах даних чи в доступі до зовнішніх ресурсів;
3. помилки інтерфейсу;
4. помилки поведінки, ініціалізації та завершення або недостатні характеристики системи.

Таким чином, користувач не має уявлення про структуру та внутрішній устрій системи. Потрібно концентруватися на тому, що програма робить, а не на тому, як вона це робить.

Переваги:

1. тестування проводиться з позиції кінцевого користувача;
2. той, хто проводить тестування, може не знати мов програмування і може не заглиблюватись у внутрішню реалізацію додатку;
3. тест-кейси пишуться лише на основі готової специфікації.

Недоліки:

1. тестування обмежене незнанням реалізації внутрішньої реалізації, важко підібрати критичні вхідні значення;
2. без чіткої специфікації важко скласти ефективні тест-кейси;
3. деякі тести можуть виявитися надмірними.

Біла скринька – це метод тестування програмного забезпечення, який передбачає, що внутрішня структура та реалізація системи відомі для тих, хто проводить тестування і використовуються для створення тестів. Вхідні значення вибираються, ґрунтуючись на знанні коду, який буде їх обробляти.

Переваги:

1. тестування може виконуватись до створення інтерфейсу користувача;
2. можливо підібрати більш ретельні тест-кейси, з покриттям великої кількості шляхів виконання програми;
3. покриття саме логіки (вихідний текст) програми.

Недоліки:

1. для виконання тестування необхідні спеціальні знання;
2. не можливо складати тест-кейси на основі лише специфікації;
3. при автоматизації тестування підтримка тестових скриптів може виявитися важкою, особливо при зміні тестованої програми.

Також можлива комбінація підходів білої та чорної скриньки. Такий метод називають сіра або напівпрозора скринька. Тобто, нам відомо внутрішній устрій програми, але саме тестування проводиться лише з позиції користувача.

4.2 Тестування роботи клієнтської частини веб-сервісу

Насамперед слід перевірити реєстрацію клієнта на веб-ресурсі для подальшої діяльності на ньому (рис.4.1), після чого у базі даних заповнюється запис з введених клієнтом даних (рис. 4.2)

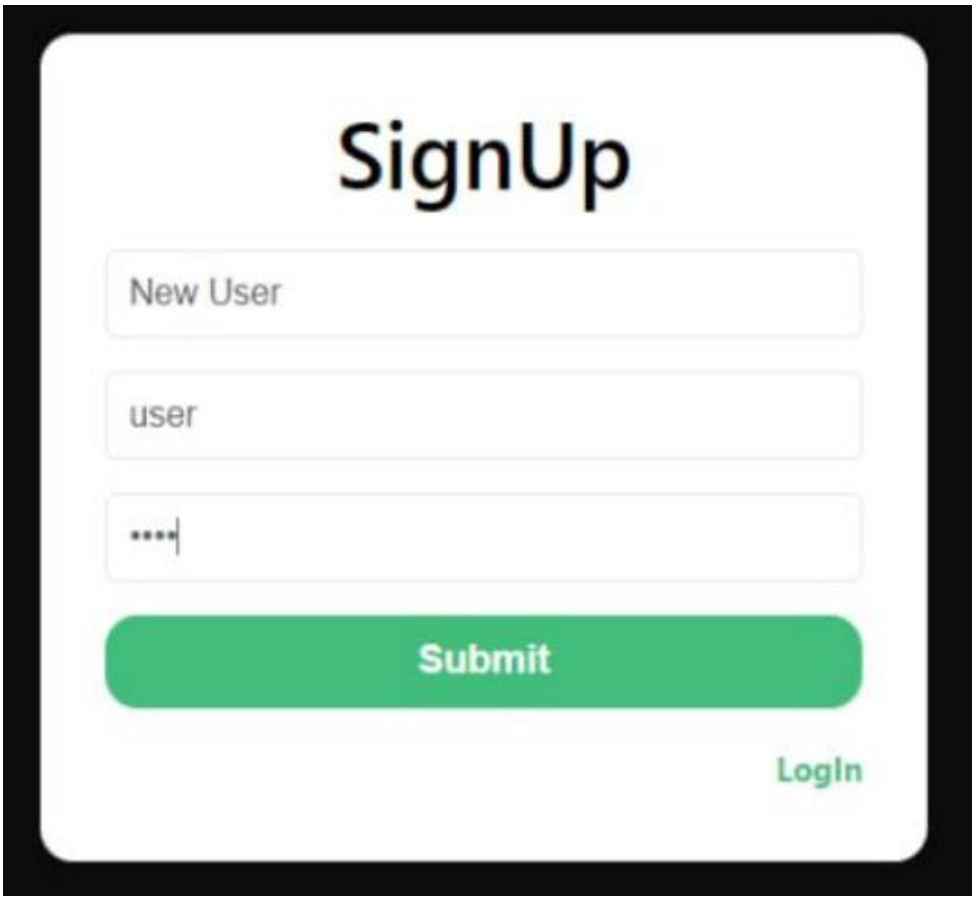


Рисунок 4.1 – Реєстрація клієнта на веб-ресурсі

Рисунок 4.2 – Запис даних клієнта у базі даних

У разі некоректного вводу даних при повторній авторизації, система викличе помилку (рис. 4.3)

Якщо ж авторизація пройшла коректно, то клієнт потрапляє на веб-ресурс і може виконувати дії на ньому. При пошуку певної гри на веб-ресурсі,

наприклад “The Witcher 3”, клієнт у разі не повної, або не завершеної сторінки, бачить на екрані такий контент (рис. 4.4).

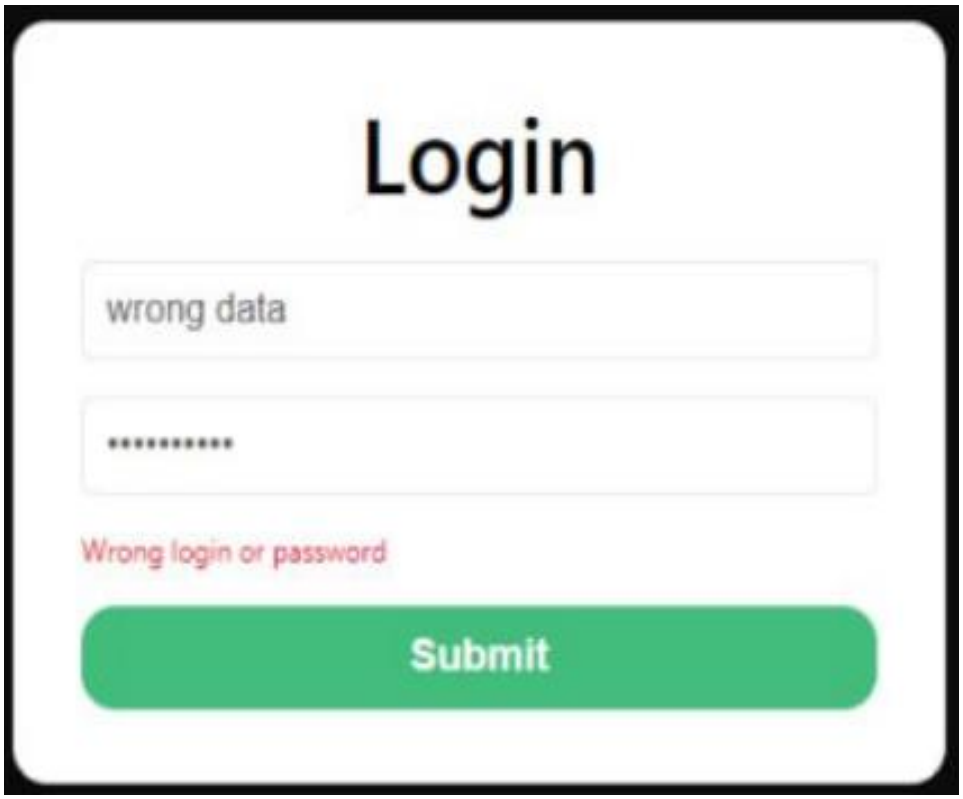


Рисунок 4.3 – Помилка при некоректному вводі даних

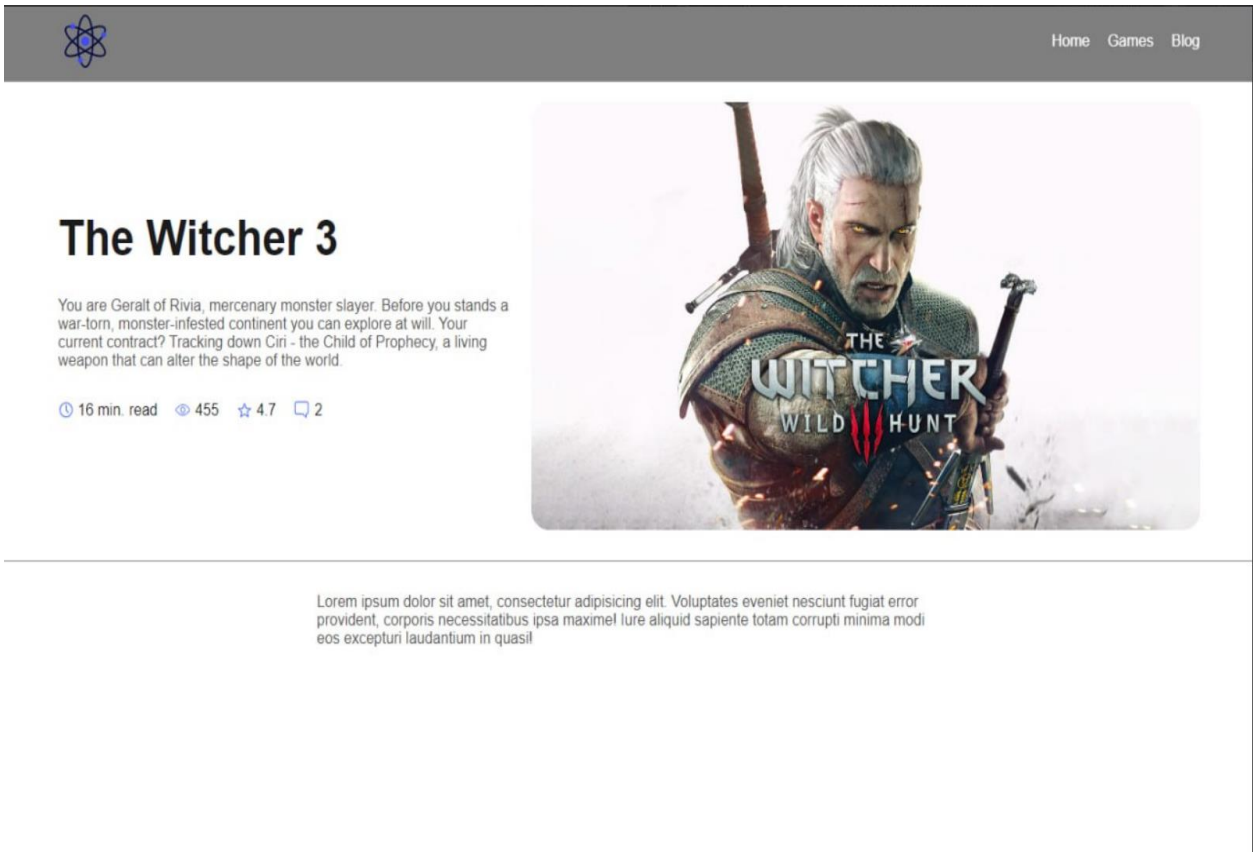


Рисунок 4.4 – Контент сторінки про гру до повної готовності

Коли адміністратор доповнить контент на сторінці гри і вона буде готова, то для отримання оновленої сторінки клієнту необхідно дописати у URL-адресу “?ssr=true”, тоді спрацює метод “generatePage” – на клієнтській сторінці контент оновиться і автоматично з’явиться готова інформація про гру (рис. 4.5).

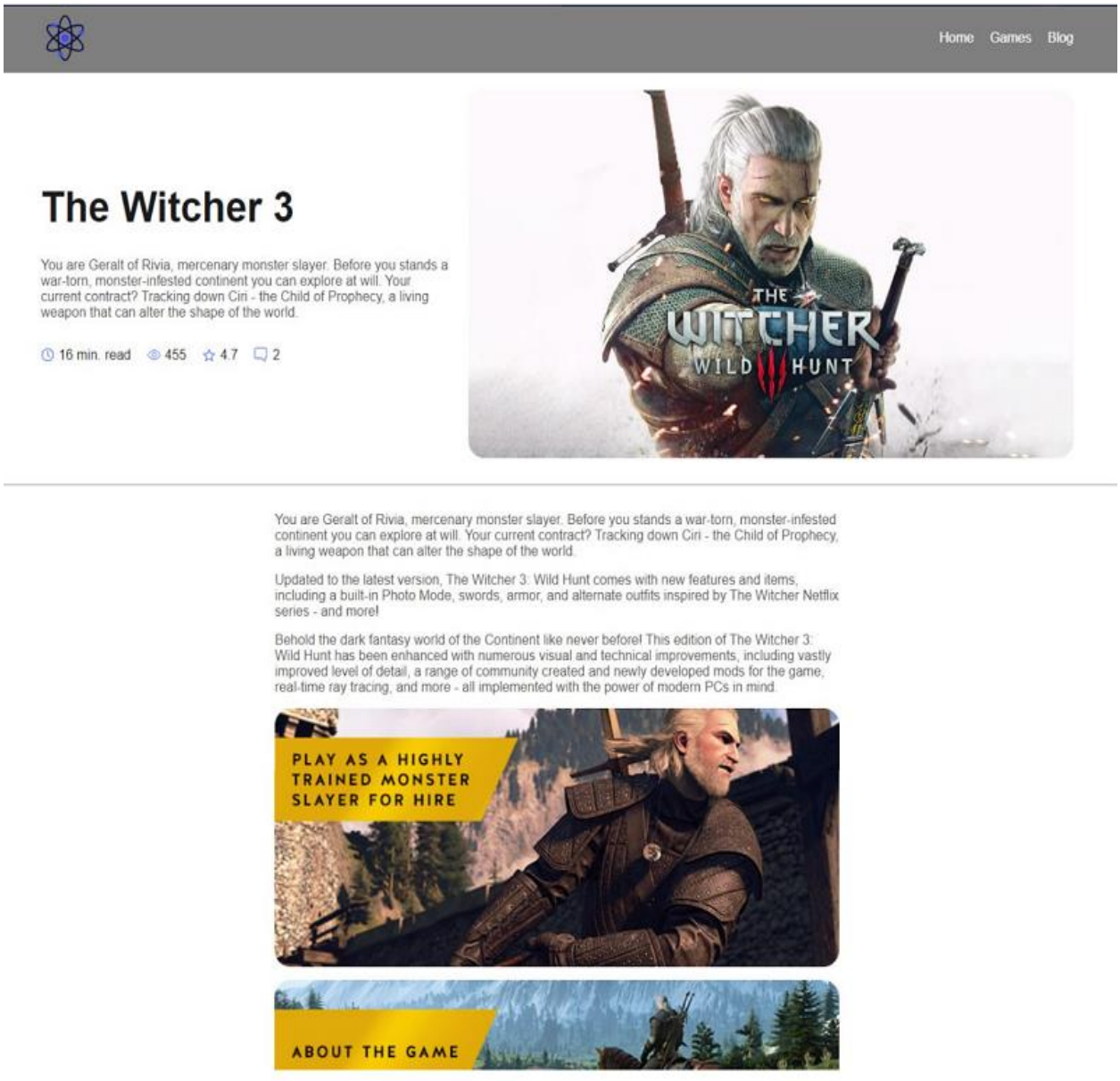


Рисунок 4.5 – Готовий контент після перегенерації сторінки

Тепер протестуємо перехід клієнта зі сторінки окремої гри на сторінку із загальним списком ігор “Games”та переглянемо її (рис. 4.6).

Після цього залишилося перевірити сторінку новин “Blog” з клієнтської частини (рис. 4.7) .

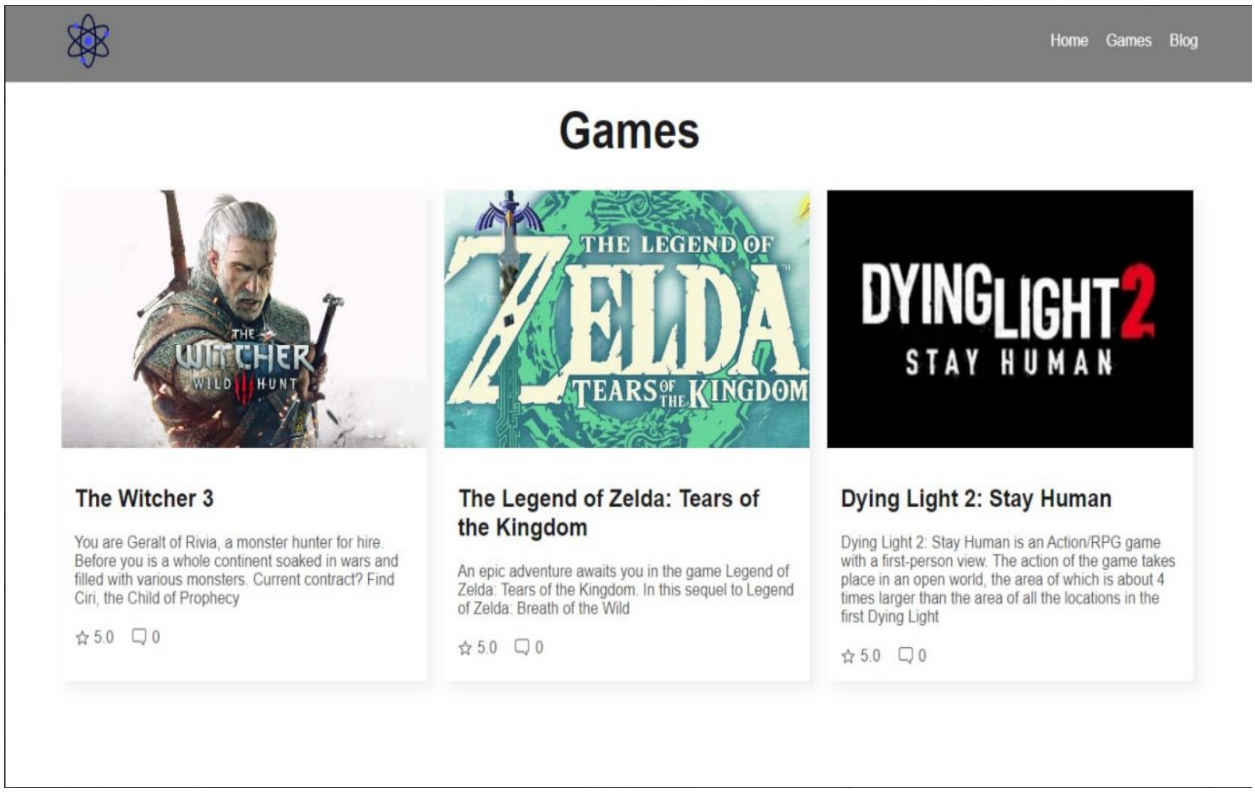


Рисунок 4.6 – Зовнішній вигляд сторінки “Games”

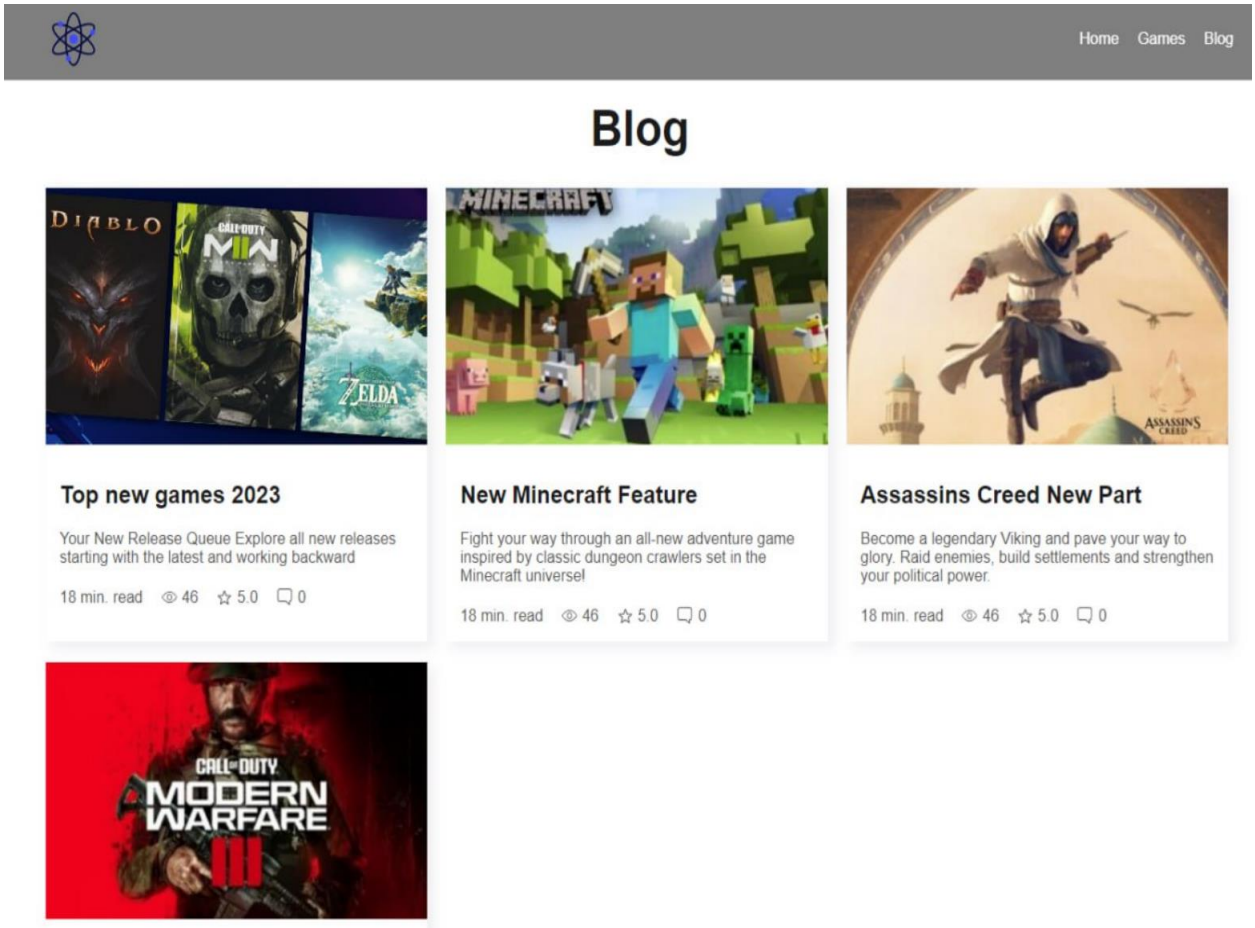


Рисунок 4.7 – Зовнішній вигляд сторінки “Blog”

4.3 Тестування роботи адміністративної частини веб-сервісу

Перевіримо спочатку авторизацію адміністратора для входу на сайт з привілеями адміністратора, адміністратору необхідно заповнити відповідні поля для подальшої роботи на веб-ресурсі (рис. 4.8), дані для авторизації клієнтів та адміністраторі знаходяться у колекції у базі даних mongoDB (рис. 4.9).

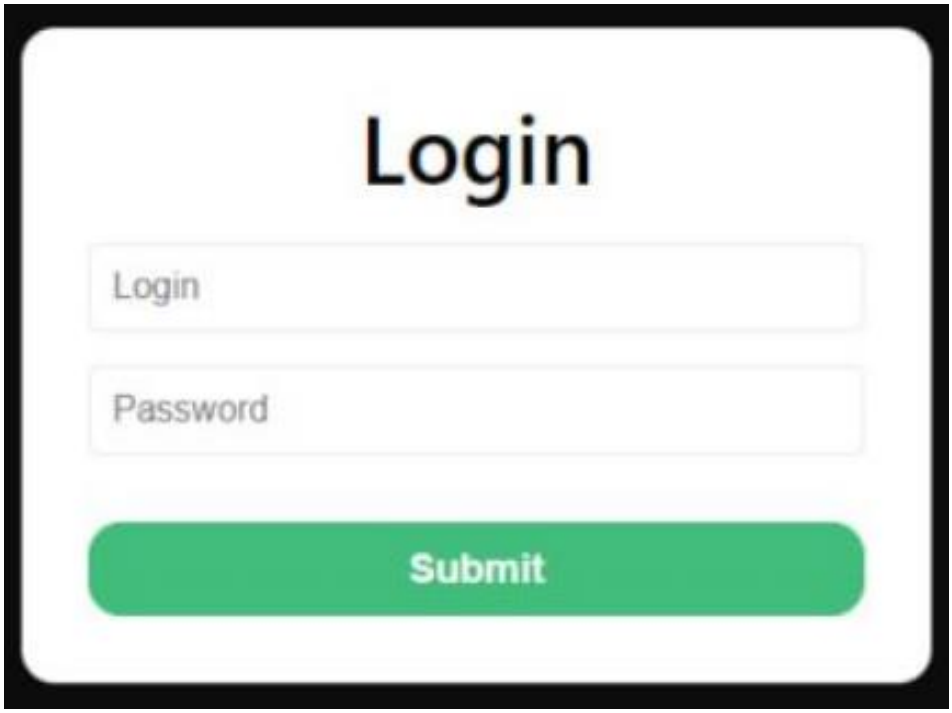


Рисунок 4.8 – Авторизація на сайті


```
_id: ObjectId('65665bba16ce5f6d070359ad')
role: "admin"
name: "John Doe"
login: "login"
password: "password"
__v: 0
```


Рисунок 4.9 – Запис даних адміністратора у базі даних

Після входу у ролі адміністратора, в’являється можливість корегувати та додавати певні пости, новини, або ігри, спочатку сторінка постів порожня (рис. 4.10)

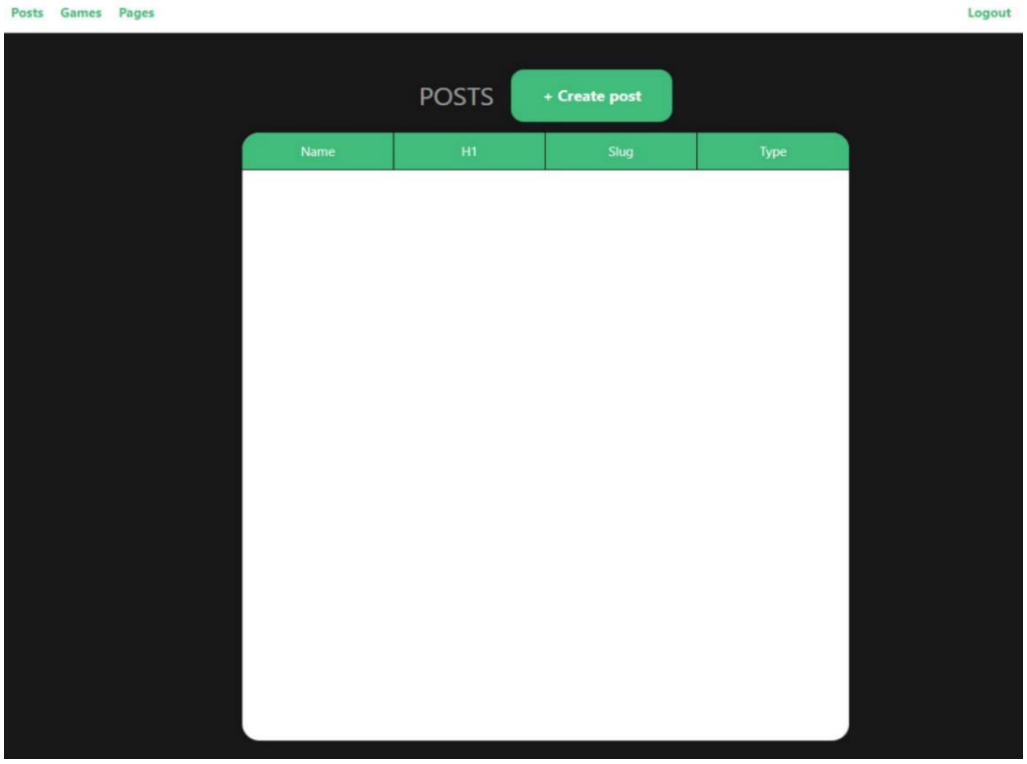


Рисунок 4.10 – Сторінка постів

Тепер адміністратор заповнює сторінку постів контентом (рис. 4.11), а дані записуються у поля бази даних (таблиця. 4.1, таблиця 4.2).

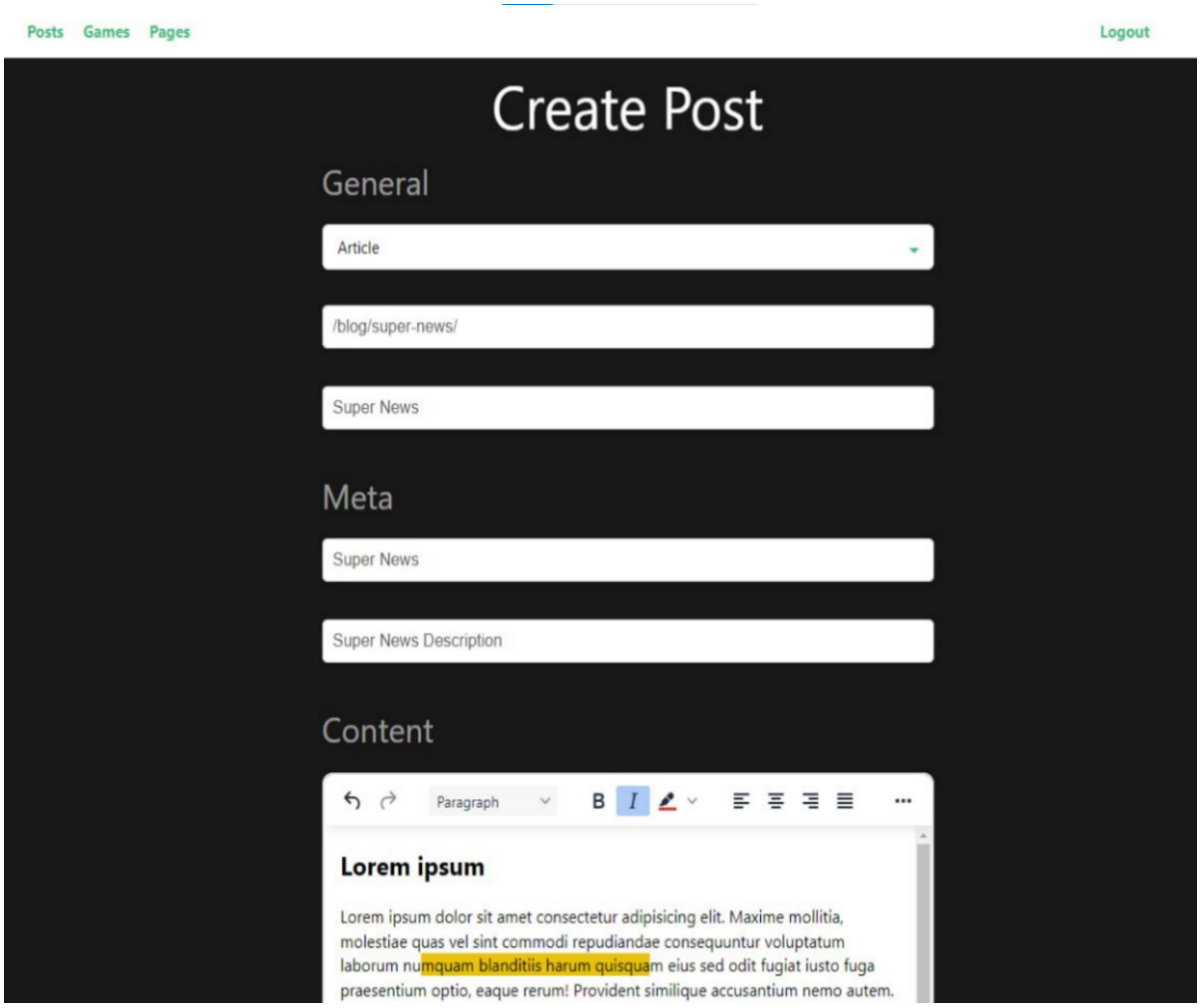


Рисунок 4.11 – Створення поста на сторінці

Таблиця 4.1 – Відповідний запис у mongoDB(стаття)

Назва	Значення
Type	Article
slug	/blog/top-20-games/
name	Top 20 Games
h1	Top 20 Games
meta	[title: “Top 20 Games”, description: “Post about Top 20 Games”]

Продовження таблиці 4.1 – Відповідний запис у mongoDB(стаття)

content	You are Geralt of Rivia, mercenary monster slayer. Before you stands a war-torn, monster-infested continent you can explore at will. Your current contract? Tracking down Ciri — the Child of Prophecy, a living weapon that can alter the shape of the world.
date	2023-11-29T22:06:37.147Z

Таблиця 4.2 – Відповідний запис у mongoDB(новина)

Назва	Значення
Type	News
slug	/blog/super-news/
name	Super News
h1	Super News
meta	[title: “Super News”, description: “Super News Description”]
content	Lorem ipsum Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime mollitia, molestiae quas vel sint commodi repudiandae consequuntur voluptatum laborum numquam...
date	2023-11-29T00:54:27.471Z

Тепер сторінка постів заповнилась контентом і має такий вигляд (рис. 4.12):

Рисунок 4.12 – Вигляд сторінки постів після заповнення

Перейдемо до перевірки додавання інформації про ігри у вкладку ігри (рис. 4.13) та запис у базу даних (таблиця 4.3).

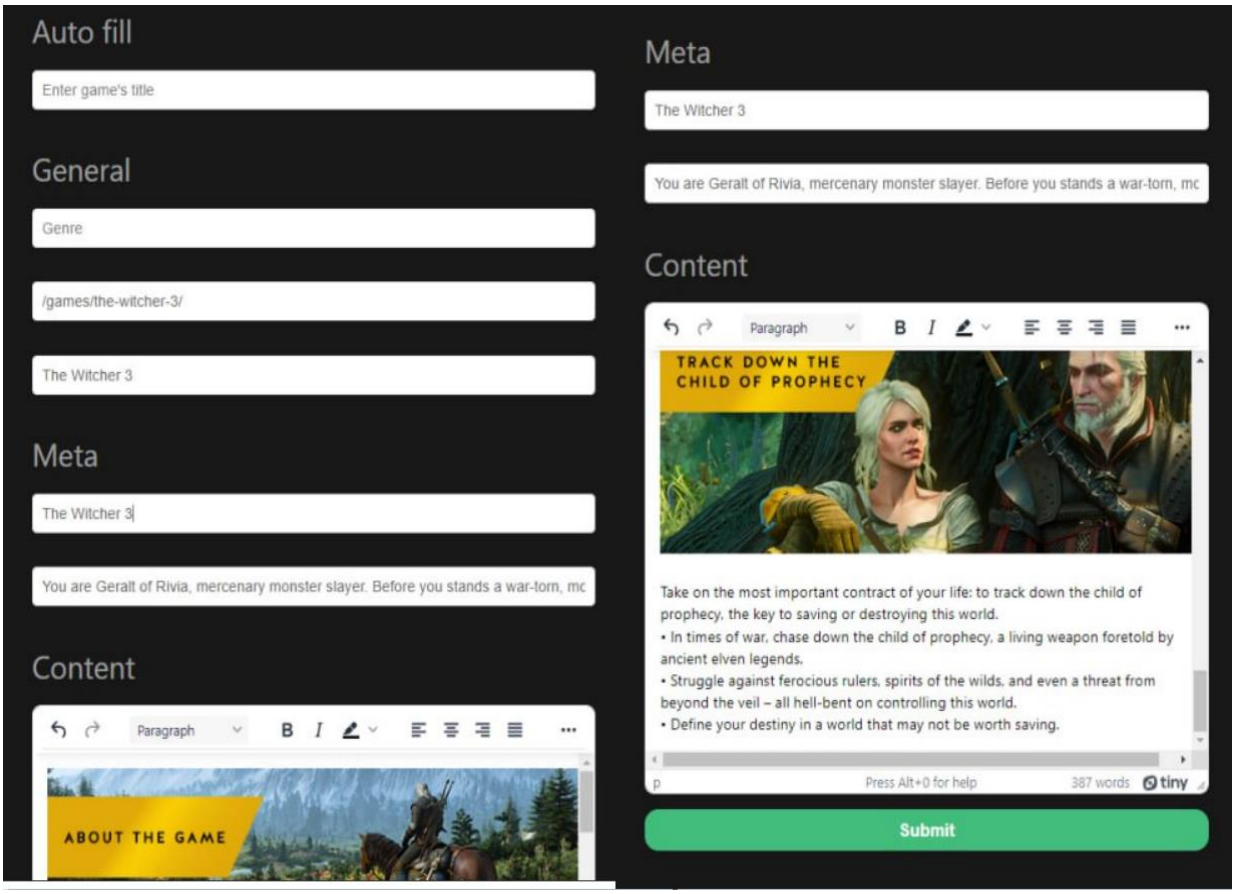


Рисунок 4.13 – Заповнення полів та додавання гри на сторінку ігри

Таблиця 4.3 – Запис гри у базі даних

Назва	Значення
genre	RPG
slug	/games/the-witcher-3/
name	The Witcher 3
meta	[title:” The Witcher 3”,description:” You are Geralt of Rivia, mercenary monster slayer. Before you stands a war-torn, monster-infested continent you can explore at will…”]
content	<img src="https://cdn.akamai.steamstatic.com/steam/apps/292030/extras/ABOUT_600x225_EN.png?t=1693590732"> You are Geralt of Rivia, mercenary monster slayer...
date	2023-11-29T21:54:15.908Z

Перевіримо функцію автозаповнення з бази даних mongoDB (рис. 4.14).

Тепер перевіримо запис нової сторінки у базу даних (таблиця 4.4) та можливість додавання сторінок на веб-ресурс (рис. 4.15)

Рисунок 4.14 – Функція автозаповнення даних про гру з бази даних

Таблиця 4.4 – Запис нової сторінки Homepage у колекцію

Назва	Значення
slug	/
name	Homepage

Продовження таблиця 4.4 – Запис нової сторінки Homepage у колекцію

meta	[title:” Homepage”, description:” Homepage Description”]
content	The standard Lorem Ipsum passage, used since the 1500s "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua… Section 1.10.32 of "de Finibus Bonorum et Malorum", written by Cicero in 45 BC …
date	2023-11-29T22:33:28.754Z

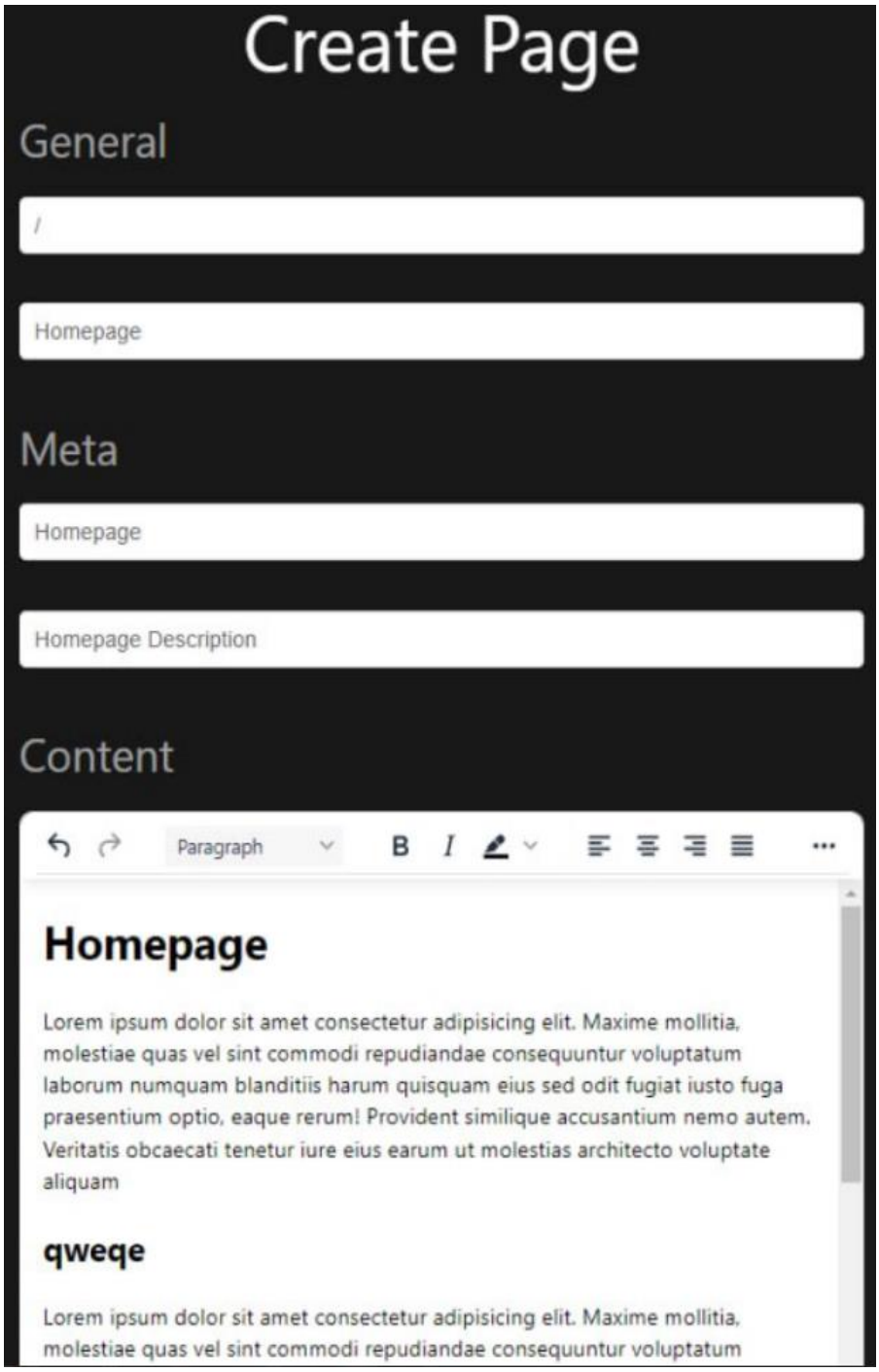


Рисунок 4.15 – Додавання нової сторінки Homepage

4.4 Висновки

Одним з важливих етапів розробки веб-ресурсу є тестування, так як коректне тестування та, відповідно, коректні результати тестування є гарантією якості розроблюваного ресурсу.

У четвертому розділі було проаналізовано основні методи тестування, описано методику тестування програмного додатку методом «чорної скриньки», «сірої» та «білої скриньки». Наведено результати тестування за таким методом, що не виявили помилок та підтвердили повну працездатність програмного продукту та відповідність технічному завданню.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу розробки методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету к.т.н., доцента Катєльнікова Д. І., к.т.н., доцента Ракитянську Г. Б., к.т.н., доцента Рейди О.М. з кафедри програмного забезпечення. Для проведення технологічного аудиту було використано таблицю 5.1, в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу розробки.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

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

Продовження таблиці 5.1

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження таблиці 5.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.2 – Рівні комерційного потенціалу розробки

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

У таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Катєльніков Д. І.	Ракитянська Г. Б.	Рейда О.М.
	Бали, виставлені експертами:		
1	4	4	4
2	2	2	2
3	3	3	3
4	3	4	3
5	2	2	2
6	3	3	3
7	2	3	3
8	3	4	3
9	3	2	1
10	3	2	3
11	3	4	4
12	4	3	3
Сума балів	$СБ_1$=35	$СБ_2$=36	$СБ_3$=34
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{35+36+34}{3} = 35$		

Розрахована нами на основі висновків експертів середньоарифметична сума балів склала 35 балів. Згідно таблиці 5.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Методи і програмні засоби веб-платформи про ігри для підвищення ефективності керування контентом, що розробляються в магістерській роботі, будуть цікаві геймерам і молодим людям, що цікавляться контентом ігор і

шукають цікаві ігри для себе і друзів. Враховуючи особливості цільової аудиторії та розважальну спрямованість проекту, дана розробка є некомерційною.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом. В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 5.4 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий	Новий		
Точність пошуку, бали	9	9	1	30%
Середній час відповіді на запит *(менше – краще)*, мс	180	100	1,8	30%
Рівень протидії неполадкам, бали	6	9	1,5	20%
Кількість мов та локалізацій, шт	5	5	1	10%
Використання ресурсів комп’ютера, %	20	20	1	10%

Порівняємо нову розробку, що розробляється в магістерській роботі, з аналогом, який існує на ринку.

В якості аналога для розробки було обрано інтерфейс управління контентом WordPress. Основним недоліком аналога є низький рівень заходів протидії збоям та високий середній час відповіді на запит, що ускладнює процес пошуку контенту.

У розробці дані проблеми вирішуються за рахунок використання комбінованого методу рендерингу та шаблонної системи на клієнті.

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.4.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{5.1}$$

$$q_i = \frac{P_{Бi}}{P_{Hi}} \tag{5.2}$$

де P_{Hi}, $P_{Бi}$ – числові значення i-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{9}{9} = 1;$$

$$q_2 = \frac{180}{100} = 1{,}8;$$

$$q_3 = \frac{9}{6} = 1{,}5;$$

$$q_4 = \frac{5}{5} = 1;$$

$$q_5 = \frac{20}{20} = 1.$$

Відносний рівень якості нової розробки визначаємо за формулою 5.3:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i \tag{5.3}$$

$$К_{я.в.} = 1 \cdot 0{,}3 + 1{,}8 \cdot 0{,}3 + 1{,}5 \cdot 0{,}2 + 1 \cdot 0{,}1 + 1 \cdot 0{,}1 = 1{,}34$$

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою 5.4:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.4}$$

де $I_{m.n.}$ – індекс технічних параметрів;

$I_{e.n.}$ – індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5):

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \tag{5.5}$$

де P_{Hei}, $P_{Бei}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$\mathrm{I_{е.п.}} = \frac{300}{300} = 1$$

$$\mathrm{К} = \frac{1{,}34}{1} = 1{,}34.$$

Виходячи з розрахунків можна зробити висновок, що нова розробка буде більш конкурентоспроможною, ніж конкурентний товар.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_0$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою (5.6):

$$З_0 = \frac{М}{Т_р} * t(грн) \quad (5.6)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$T_Р$ – число робочих днів в місяці; приблизно $T_Р \approx 21…23$ дні;

t – число робочих днів роботи дослідника.

Для розробки методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом необхідно залучити: форнт-енд розробник з посадовим окладом 20000 грн та бек-енд розробник з посадовим окладом 25000 грн; а також керівника роботи.

Кількість робочих днів у місяці складає 22, кількість робочих днів кожного програміста складає 33, а керівника 35 днів

Зведемо сумарні розрахунки до таблиці 5.6.

Таблиця 5.6 – Заробітна плата спеціалістів для розробки програмних засобів

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	12000	545,5	35	19 093
Фронт-енд розробник	20000	909,09	33	30 000
Бек-енд розробник	25000	1 136,36	33	37 500
Всього				86 593

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали участь в розробці нового технічного рішення розраховується як 10-12% від основної заробітної плати робітників за формулою (5.7).

На даному підприємстві додаткова заробітна плата начисляється в розмірі 11% від основної заробітної плати.

$$З_д = \frac{(З_о + З_р) * Н_{доп}}{100\%} \quad (5.7)$$

$$З_д = 0{,}11 \cdot 86593 = 9525{,}23(грн)$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.8):

$$Н_{ЗП} = \frac{(З_о + З_д) * \beta}{100} \quad (5.8)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, %

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{ЗП} = \frac{(86593 + 9525{,}23) \cdot 22}{100} = 21146(грн)$$

4. Витрати на матеріали М та комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою (5.9):

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \quad (5.9)$$

де $Н_i$ – витрати матеріалу *і*-го найменування, кг;

$Ц_i$ – вартість матеріалу *і*-го найменування, грн./кг.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

$В_i$ – маса відходів матеріалу *і*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *і*-го найменування, грн/кг;

n – кількість видів матеріалів.

Таблиця 5.7 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір	200	1	200
Ручка	35	1	35
CD-диск	40	1	40
Флешка	155	1	155
Всього			430
З врахуванням коефіцієнта транспортування			494,5

5. Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення необхідного для проведення дослідження. Для нової розробки використовувались безкоштовні програмні засоби.

6. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо за формулою (5.10).

$$A = \frac{Ц \cdot T}{Ткор \cdot 12} \quad [грн], \tag{5.10}$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункту 137.3.3 Податкового кодексу амортизація нараховується на основні засоби вартістю понад 6500 грн. В нашому випадку для написання магістерської роботи використовувалися три персональних комп'ютери кожен вартістю 40000 грн, а також оренда приміщення, вартість оренди 15000 грн.

$$А = А_к + А_п = 5250$$

$$А_к = \frac{120000 \cdot 2}{5 \cdot 12} = 4000$$

$$А_п = \frac{15000 \cdot 2}{2 \cdot 12} = 1250$$

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що використовуються з технологічною метою на проведення досліджень, та розраховуються за формулою (5.11).

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_і} \tag{5.11}$$

де W_{yt} – встановлена потужність обладнання, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

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

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

Для написання магістерської роботи використовувалися три персональних комп'ютери для яких розрахуємо витрати на електроенергію.

$$В_е = \frac{1,5 \cdot 360 \cdot 7,6 \cdot 0,5}{0,8} = 2565$$

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = \left(З_о + З_р\right) \cdot \frac{Н_{св}}{100} \quad (5.12)$$

де $Н_{св}$ – норма нарахування за статтею «Інші витрати».

$$В_{св} = (86593 + 9525,23) * 0,22 = 21146 \text{ грн}$$

До статті «Витрати на роботи, які виконують сторонні підприємства, установи і організації» належать витрати на проведення досліджень, що не можуть бути виконані штатними працівниками або наявним обладнанням організації, а виконуються на договірній основі іншими підприємствами, установами і організаціями незалежно від форм власності та позаштатними працівниками.

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{сп} = \left(З_о + З_р\right) \cdot \frac{Н_{сп}}{100} \quad (5.13)$$

де $Н_{пв}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$В_{сп} = 96118,23 * 0,4 = 38447,29 \text{ грн}$$

Накладні (загальновиробничі) витрати $В_{нзв}$ охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати $В_{нзв}$ можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, що розраховуються за формулою (5.13):

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.13)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 96118,23 \cdot \frac{100}{100\%} = 96118,23(\text{грн})$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР:

$$В = 96118,23 + 21146 + 494,5 + 5250 + 2565 + 21146 + 38447,29 + 96118,23 = 281285,25(\text{грн})$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою 5.13:

$$ЗВ = \frac{В}{\eta}, \quad (5.13)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії впровадження, то коефіцієнт $\eta = 0{,}9$. Звідси:

$$ЗВ = \frac{281285{,}25}{0{,}7} = 401836{,}07(\text{грн.})$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

Економічна ефективність дозволяє спрогнозувати чистий прибуток, який може бути отриманий від впровадження розробленої системи.

При розрахунку економічної ефективності потрібно обов'язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу.

Для розрахунку збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки використовується формула формулою (5.14):

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right) \quad (5.14)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження нової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

Цо – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333;

ρ – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

ϑ – ставка податку на прибуток. У 2023 році – 18%.

Впровадження результатів розробки дозволяє зручніше, якісніше і стійкіше до помилок генерувати сторінки на веб-сайті. Припустимо, що ціна розробленого програмного продукту зросте на 3000 грн за користувача. Кількість клієнтів, які користуються веб-сайтом збільшиться протягом першого року на 600 чоловік., протягом другого року – на 900 шт., а протягом третього року на 1300 шт. Кількість клієнтів до впровадження розробки складала 1 шт., а її ціна складала 300 грн.

$$\Delta П_1 = (3000 \cdot 1 + (3000 + 300) \cdot 600) \cdot 0.833 \cdot 0.25 \cdot \left(1 - \frac{18}{100}\right) = 338627 \text{ грн}$$

$$\Delta П_2 = (3000 \cdot 1 + (3000 + 300) \cdot (600 + 900)) \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 - \frac{18}{100}\right) = 845799 \text{ грн}$$

$$\Delta П_3 = \left(3000 \cdot 1 + (3000 + 300) \cdot (600 + 900 + 1300)\right) \cdot 0.833 \cdot 0.25 \cdot \left(1 - \frac{18}{100}\right) = 1578380 \text{ грн}$$

Далі за формулою (5.15) розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} \quad (5.15)$$

де ΔПі – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15. В Україні рівень інфляції за підсумком 2023 року склав 10.6%, прогнозований рівень на 2024 рік – 8.5% ;

t – період часу (в роках) від моменту початку впровадження розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$\text{ПП} = \frac{338627}{1+0.1} + \frac{845799}{(1+0.085)^2} + \frac{1578380}{(1+0.085)^3} = 307842 + 718468 + 1235726 = 2262036$$

За формулою (5.16) необхідно розрахувати величину початкових інвестицій PV , які потенційний інвестор має вкласти для впровадження і комерціалізації розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ} \qquad (5.16)$$

де $k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{\text{інв}}$ =2...5, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 3 \cdot 401836 = 1205508 \text{ грн}$$

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації розробки становитиме:

$$E_{\text{абс}} = \text{ПП} - PV \quad (5.17)$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

$$E_{\text{абс}} = 2262036 - 1205508 = 1056528 \text{ грн}$$

Оскільки величина економічного ефекту має велике додатне значення, це свідчить про потенційну зацікавленість інвесторів у впровадженні та комерціалізацію розробки.

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Для остаточного прийняття рішення про впровадження розробки та виведення її на ринок необхідно розрахувати внутрішню економічну дохідність $Е_в$ або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$. Для цього користуються формулою (5.18):

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.18)$$

де $Е_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн; PV – теперішня вартість початкових інвестицій, грн;

$Т_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$Е_в = \sqrt[3]{1 + \frac{1056528}{1205508}} - 1 = \sqrt[3]{1.8764} - 1 = 1{,}23341 - 1 = 0.23 = 23\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою (5.19):

$$\tau = d + f, \tag{5.19}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d= (0,14…0,2);

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

$$\tau_{min} = 0.16 + 0.06 = 0.22$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Далі розраховується період окупності інвестицій, вкладених у реалізацію проекту за формулою (5.17).

$$T_{ок} = \frac{1}{E_в} \tag{5.17}$$

де Ев – внутрішня економічна дохідність вкладених інвестицій.

$$Т_{ок} = \frac{1}{0.23} = 4{,}34 \text{ роки} = 52 \text{ міс} = 4 \text{ роки та } 4 \text{ міс}$$

Так як $Т_{ок} \leq$ 3…5-ти років, то це свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження цієї розробки та виведення її на ринок.

5.5 Висновки до економічного розділу

Було проведено оцінку комерційного потенціалу методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом, який є на рівні вище середнього. При порівнянні нової розробки з аналогом виявлено, що вона є якіснішою і більш конкурентоспроможною в порівнянні з аналогом, а також є кращою по технічним і економічним показникам.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 281 285,25 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 401 836,07грн.

Вкладені інвестиції в даний проект окупляться через 4 роки та 4 місяці при прогнозованому прибутку 2 262 036 грн. за три роки.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було розроблено веб-платформу для ігор з використанням комбінованого методу SSG та SSR. Завдяки цьому комбінованому методу забезпечується пришвидшення рендерингу та його гнічкість.

Перш за все, було розглянуто актуальність питання створення комбінованого методу рендерингу веб-сторінок та проаналізовано питання вибору інтерфейсу програмування додатків (API), з метою покращення умов використання клієнтом веб-ресурсів, та зручності взаємодії адміністратора з контентом веб-сторінки. Було проаналізовано можливі підходи до вирішення питання створення комбінованого методу рендерингу, шляхом порівняння систем-аналогів, окреслення їх сильних та слабких сторін, таких як SEO, швидкість, динамічність та автоматизація оновлення. Було визначено такі основні задачі для розробки, як розробка методу з високим SEO, високою швидкістю рендерингу та гнучкою зміною контенту. У роботі надано детальний опис алгоритму роботи серверної частини веб-ресурсу, розкрито принципи функціонування та структуру інших частин веб-ресурсу. Описана також модель системи для роботи з відкритою базою даних, що вказує на важливі аспекти зберігання та обробки інформації в системі. Було проведено аналіз різних мов програмування для розробки веб-ресурсу, а також середовищ програмування для його розробки та обрано мову Java Script для розробки веб-платформи, та обрано Visual Studio Code в якості IDE. Було розроблено та реалізовано головний функціонал серверної частини веб-платформи, алгоритми роботи з даними та базами даних, також було розроблено адмін-панель і клієнтську частину веб-платформи та принципи побудови сторінки на клієнті.

Для тестування було проаналізовано основні методи тестування програмного додатку методом «чорної скриньки», «сірої» та «білої скриньки». В результаті, помилок не було виявлено, що підтверджує працездатність програмного продукту та відповідність технічному завданню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SPA vs MPA [Electronic resource] – Access Mode: - https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58
2. SEO [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Glossary/SEO
3. Single Page Application [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Glossary/SPA
4. Advantages of each method [Electronic resource] – Access Mode: - https://www.makeuseof.com/csr-ssr-ssg-isr-pick-right-rendering-paradigm/
5. Server Side Rendering [Electronic resource] – Access Mode: - https://solutionshub.epam.com/blog/post/what-is-server-side-rendering
6. Methods of rendering web-pages [Electronic resource] – Access Mode: - https://www.educative.io/answers/ssr-vs-csr-vs-isr-vs-ssg
7. API [Electronic resource] – Access Mode: -

https://hostiq.ua/blog/ukr/what-is-api

8. Document Object Model (DOM) [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
9. HTTP Request [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
10. XML HTTP Request [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest
11. API Request [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Web/API/Request
12. Axios Library for Requests [Electronic resource] – Access Mode: - https://axios-http.com/docs/intro
13. Паляниця Д.Р., Катєльніков Д.І. Розробка методів та програмних засобів серверу на базі комбінованих технологій SSG та SSR для системи керування

контентом.// Матеріали III Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації - 2023», Одеса, 28-29 жовтня 2023 р. Одеса, Видавництво ОНТУ, 2023 р. С.108-109.

14. Паляниця Д.Р., Катєльніков Д.І. Розробка методів та програмних засобів серверу на базі комбінованих технологій SSG та SSR для системи керування контентом.// Матеріали Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (20-21 листопада, 2023, м.Суми/Вінниця). С.183-184.

15. MongoDB [Electronic resource] – Access Mode: - https://www.mongodb.com/docs/drivers/node/current/fundamentals/connection/

16. IGDB Open Database [Electronic resource] – Access Mode: - https://api-docs.igdb.com/#getting-started

17. CMS [Electronic resource] – Access Mode: - https://justpro.com.ua/blog/shcho-take-cms-ta-ikh-vydy/

18. Wordpress CMS [Electronic resource] – Access Mode: - https://codex.wordpress.org/

19. Реєстрація [Electronic resource] – Access Mode: - https://uk.wikipedia.org/wiki/Реєстрація

20. Автентифікація [Electronic resource] – Access Mode: - https://www.microsoft.com/uk-ua/security/business/security-101/what-is-authentication

21. Авторизація [Electronic resource] – Access Mode: - https://introserv.com/ua/blog/u-chomu-rizniczya-mizh-autentifikaczieyu-ta-avtorizaczieyu/#авторизація

22. Користувацький інтерфейс [Electronic resource] – Access Mode: - https://kr-labs.com.ua/blog/shho-take-frontend-i-backend/

23. Introduction to C# [Electronic resource] – Access Mode: - https://www.w3schools.com/cs/cs_intro.php

24. Introduction to C++ [Electronic resource] – Access Mode: - https://www.w3schools.com/cpp/cpp_intro.asp

25. What is JavaScript [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

26. Visual Studio Code [Electronic resource] – Access Mode: - https://uk.wikipedia.org/wiki/Visual_Studio_Code

27. WebStorm [Electronic resource] – Access Mode: - https://uk.wikipedia.org/wiki/WebStorm

28. WebP [Electronic resource] – Access Mode: - https://www.adobe.com/ua/creativecloud/file-types/image/raster/webp-file.html

29. Express.js routing [Electronic resource] – Access Mode: - https://expressjs.com/en/starter/basic-routing.html

30. Vue.js [Electronic resource] – Access Mode: - https://vuejs.org/

31. Методи тестування [Electronic resource] – Access Mode: - https://qagroup.com.ua/publications/vydy-testuvannya-ta-vidminnosti-mizh-nymy/

32. IGDB Open Database [Electronic resource] – Access Mode: - https://api-docs.igdb.com/#getting-started

33. Інформаційне забезпечення [Electronic resource] – Access Mode: - https://uk.wikipedia.org/wiki/Інформаційне_забезпечення

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доц. каф. ПЗ Катєльніков Д. І.

"19" 09 2023 р.

Виконав:

_____ студент гр.2ПІ-22м Д.Р. Паляниця

"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом».

Галузь застосування – веб-платформи та веб-ресурси.

2. Підстава для розробки.

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

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності веб-серверу для швидшого завантаження веб-сторінок, а також підвищення зручності керування контентом.

Призначення роботи – розробка методів і програмних засобів рендерингу веб-сторінок на сервері із комбінованими технологіями.

3 Вихідні дані для проведення НДР

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

1. MongoDB [Electronic resource] – Access Mode: - https://www.mongodb.com/docs/drivers/node/current/fundamentals/connection/
2. API Request [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Web/API/Request
3. Server Side Rendering [Electronic resource] – Access Mode: - https://solutionshub.epam.com/blog/post/what-is-server-side-rendering
4. HTTP Request [Electronic resource] – Access Mode: - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

4. Технічні вимоги

Комбінація технологій рендерингу та генерації сторінок SSG та SSR; швидка генерація сторінок; стійкість до помилок; зберігання згенерованих сторінок в кеш; зручність роботи; простота архітектури серверу.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання розробки серверної частини та задач дослідження	20.09.2023 – 25.09.2023	Вик.
2	Розробка методу і моделі роботи серверної частини	25.09.2023 – 04.10.2023	Вик.
3	Вибір середовища та мови розробки	04.10.2023 – 07.10.2023	Вик.
4	Розробка програмного забезпечення серверної частини	07.10.2023 – 27.10.2023	Вик.
5	Тестування роботи системи	27.10.2023 – 08.11.2023	Вик.
6	Економічна частина	08.11.2023 – 22.11.2023	Вик.
7	Оформлення матеріалів до захисту МКР	22.11.2023 – 01.12.2023	Вик.

9. Порядок контролю та прийняття.

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

Додаток Б – Протокол перевірки роботи на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Розробка методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 2ПІ – 22м

Науковий керівник: Катєльніков Д. І.

Unicheck	
Оригінальність	**88,2**
Схожість	11,8

Аналіз звіту подібності

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

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

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Паляниця Д. Р.

Керівник роботи _____ Катєльніков Д. І.

Додаток В – Лістинг коду

Клієнтська частина

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon">
   <title>Home</title>
   <link rel="stylesheet" href="/assets/styles/style.css">
   <link rel="stylesheet" href="/assets/styles/home.css">
</head>
<body>
   <header>
      <div class="container">
         <div class="logo">
            <a href="/">
               <svg xmlns="http://www.w3.org/2000/svg" width="800px" height="800px"
viewBox="0 0 1024 1024" class="icon" version="1.1"><path d="M512 960c-92.8 0-160-200-160-
448S419.2 64 512 64s160 200 160 448-67.2 448-160 448z m0-32c65.6 0 128-185.6 128-416S577.6
96 512 96s-128 185.6-128 416 62.4 416 128 416z" fill="#050D42"/><path d="M124.8 736c-48-80
92.8-238.4 307.2-363.2S852.8 208 899.2 288 806.4 526.4 592 651.2 171.2 816 124.8 736z m27.2-
16c33.6 57.6 225.6 17.6 424-97.6S905.6 361.6 872 304 646.4 286.4 448 401.6 118.4 662.4 152
720z" fill="#050D42"/><path d="M899.2 736c-46.4 80-254.4 38.4-467.2-84.8S76.8 368 124.8
288s254.4-38.4 467.2 84.8S947.2 656 899.2 736z m-27.2-16c33.6-57.6-97.6-203.2-296-318.4S184
246.4 152 304 249.6 507.2 448 622.4s392 155.2 424 97.6z" fill="#050D42"/><path d="M512
592c-44.8 0-80-35.2-80-80s35.2-80 80-80 80 35.2 80 80-35.2 80-80 80zM272 312c-27.2 0-48-
20.8-48-48s20.8-48 48-48 48 20.8 48 48-20.8 48-48 48zM416 880c-27.2 0-48-20.8-48-48s20.8-48
48-48 48 20.8 48 48-20.8 48-48 48z m448-432c-27.2 0-48-20.8-48-48s20.8-48 48-48 48 20.8 48
48-20.8 48-48 48z" fill="#2F4BFF"/></svg>
            </a>
         </div>
         <nav>
            <ul>
               <li>
                  <a href="/">Home</a>
               </li>
               <li>
                  <a href="/">Games</a>
               </li>
               <li class="with_child">
                  <a href="/blog/">Blog</a>
                  <div class="child">
                     <div class="container">
                        <ul>
                           <li>
                              <a href="/blog/article/">Post</a>
                           </li>
```

```
                    </ul>
                  </div>
                </div>
              </li>
            </ul>
          </nav>
        </div>
      </header>
      <main>
          <div class="container margin-bottom">
            <div class="flex-wrapper">
              <div class="text">
                <h1>Title</h1>
                <div class="desc">
                  <p>Description</p>
                </div>
                <div class="meta_left">
                  <div class="stats">
                    <div class="time">
                      <svg xmlns="http://www.w3.org/2000/svg" width="17" height="18"
viewBox="0 0 17 18" fill="none"><path d="M8.5 16.0298C7.029 16.0298 5.59104 15.5936
4.36795 14.7763C3.14486 13.9591 2.19158 12.7975 1.62865 11.4385C1.06572 10.0795 0.918435
8.58404 1.20541 7.1413C1.49239 5.69857 2.20074 4.37333 3.2409 3.33318C4.28105 2.29303
5.60629 1.58468 7.04902 1.2977C8.49175 1.01072 9.98719 1.15801 11.3462 1.72093C12.7052
2.28386 13.8668 3.23714 14.6841 4.46023C15.5013 5.68333 15.9375 7.12129 15.9375
8.59229C15.9375 10.5648 15.1539 12.4566 13.7591 13.8514C12.3643 15.2462 10.4726 16.0298
8.5 16.0298ZM8.5 2.21729C7.23915 2.21729 6.00661 2.59118 4.95824 3.29167C3.90988 3.99216
3.09278 4.9878 2.61027 6.15268C2.12776 7.31756 2.00152 8.59936 2.2475 9.83599C2.49348
11.0726 3.10064 12.2085 3.9922 13.1001C4.88376 13.9917 6.01967 14.5988 7.2563
14.8448C8.49293 15.0908 9.77473 14.9645 10.9396 14.482C12.1045 13.9995 13.1001 13.1824
13.8006 12.134C14.5011 11.0857 14.875 9.85314 14.875 8.59229C14.875 6.90153 14.2034
5.28003 13.0078 4.08448C11.8123 2.88894 10.1908 2.21729 8.5 2.21729V2.21729Z"
fill="#2B60FA"></path><path d="M10.9384 11.7798L7.96875
8.8101V3.81104H9.03125V8.36916L11.6875 11.0307L10.9384 11.7798Z"
fill="#2B60FA"></path></svg>
                      <span>16 min.</span>
                      <span>read</span>
                    </div>
                    <div class="view">
                      <svg xmlns="http://www.w3.org/2000/svg" width="17" height="20"
viewBox="0 0 17 20" fill="none"><g clip-path="url(#clip0_1357_3724)"><path d="M16.5385
9.36474C16.4078 9.1449 13.2662 3.99609 8.29996 3.99609C4.03853 3.99609 0.241214 9.11533
0.0814369 9.33356C-0.0271456 9.48225 -0.0271456 9.70048 0.0814369 9.84995C0.241214
10.0682 4.03853 15.1874 8.29996 15.1874C12.5614 15.1874 16.3587 10.0682 16.5185
9.84995C16.6188 9.71247 16.6278 9.51342 16.5385 9.36474ZM8.29996 14.3881C4.88378
14.3881 1.63564 10.623 0.810487 9.59177C1.63427 8.55979 4.87895 4.7955 8.29996
4.7955C12.2972 4.7955 15.1185 8.55499 15.8074 9.57019C15.012 10.5686 11.7452 14.3881
8.29996 14.3881Z" fill="#2B60FA"></path><path d="M8.29795 6.39471C6.77212 6.39471
5.53125 7.8288 5.53125 9.59222C5.53125 11.3556 6.77212 12.7897 8.29795 12.7897C9.82378
12.7897 11.0646 11.3556 11.0646 9.59222C11.0646 7.8288 9.82378 6.39471 8.29795
6.39471ZM8.29795 11.9904C7.15392 11.9904 6.22292 10.9144 6.22292 9.59226C6.22292 8.2701
7.15392 7.19412 8.29795 7.19412C9.44197 7.19412 10.373 8.2701 10.373 9.59226C10.373
10.9144 9.44197 11.9904 8.29795 11.9904Z" fill="#2B60FA"></path></g><defs><clipPath
```

```
id="clip0_1357_3724"><rect width="16.6" height="19.1848"
fill="white"></rect></clipPath></defs></svg>
                        <span>455</span>
                    </div>
                    <div class="like">
                        <svg xmlns="http://www.w3.org/2000/svg" width="17" height="16"
viewBox="0 0 17 16" fill="none"><path d="M8.5 1.33363L10.2448 5.64798L10.3618
5.93741L10.6733 5.95929L15.3156 6.28547L11.7516 9.27806L11.5125 9.47882L11.5879
9.78178L12.7123 14.2977L8.76482 11.8329L8.5 11.6675L8.23518 11.8329L4.28772
14.2977L5.41207 9.78178L5.4875 9.47882L5.2484 9.27806L1.68438 6.28547L6.32674
5.95929L6.63817 5.93741L6.75522 5.64798L8.5 1.33363Z" stroke="#2B60FA"></path></svg>
                        <span>4.7</span>
                    </div>
                    <div class="comment">
                        <svg xmlns="http://www.w3.org/2000/svg" width="18" height="18"
viewBox="0 0 18 18" fill="none"> <g clip-path="url(#clip0_1357_3737)"><path d="M2.82444
0.184875H14.1085C15.289 0.184875 16.248 1.29321 16.248 2.65752V10.6381C16.248 11.9996
15.2915 13.1051 14.1158 13.1107V16.7311L9.61292 13.1107H2.82444C1.64394 13.1107 0.68494
12.0024 0.68494 10.6381V2.65752C0.68494 1.29321 1.64638 0.184875 2.82444
0.184875ZM1.5977 10.6381C1.5977 11.4201 2.14778 12.0558 2.82444
12.0558H9.90013L13.2055 14.7141V12.0558H14.111C14.7876 12.0558 15.3377 11.4201 15.3377
10.6381V2.65752C15.3377 1.8755 14.7876 1.23976 14.111 1.23976H2.82687C2.15022 1.23976
1.60013 1.8755 1.60013 2.65752V10.6381H1.5977Z" fill="#2B60FA"></path></g><defs>
<clipPath id="clip0_1357_3737"><rect width="17" height="17" fill="white" transform="matrix(-1
0 0 1 17.0938 0.0924072)"></rect></clipPath></defs></svg>
                        <span>2</span>
                    </div>
                </div>
            </div>
            <div class="image">
                <img src="src">
            </div>
        </div>
    </div>
    <hr>
    <div class="container">
        <div class="content">
            Content
                    </div>
    </div>
</main>
</body>
</html>
```

404.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon">
    <title>404</title>
</head>
<body>
    <header>
        <div class="container">
            <div class="logo">
                <a href="/">
                    <svg xmlns="http://www.w3.org/2000/svg" width="800px" height="800px"
viewBox="0 0 1024 1024" class="icon" version="1.1"><path d="M512 960c-92.8 0-160-200-160-
448S419.2 64 512 64s160 200 160 448-67.2 448-160 448z m0-32c65.6 0 128-185.6 128-416S577.6
96 512 96s-128 185.6-128 416 62.4 416 128 416z" fill="#050D42"/><path d="M124.8 736c-48-80
92.8-238.4 307.2-363.2S852.8 208 899.2 288 806.4 526.4 592 651.2 171.2 816 124.8 736z m27.2-
16c33.6 57.6 225.6 17.6 424-97.6S905.6 361.6 872 304 646.4 286.4 448 401.6 118.4 662.4 152
720z" fill="#050D42"/><path d="M899.2 736c-46.4 80-254.4 38.4-467.2-84.8S76.8 368 124.8
288s254.4-38.4 467.2 84.8S947.2 656 899.2 736z m-27.2-16c33.6-57.6-97.6-203.2-296-318.4S184
246.4 152 304 249.6 507.2 448 622.4s392 155.2 424 97.6z" fill="#050D42"/><path d="M512
592c-44.8 0-80-35.2-80-80s35.2-80 80-80 80 35.2 80 80-35.2 80-80 80zM272 312c-27.2 0-48-
20.8-48-48s20.8-48 48-48 48 20.8 48 48-20.8 48-48 48zM416 880c-27.2 0-48-20.8-48-48s20.8-48
48-48 48 20.8 48 48-20.8 48-48 48z m448-432c-27.2 0-48-20.8-48-48s20.8-48 48-48 48 20.8 48
48-20.8 48-48 48z" fill="#2F4BFF"/></svg>
                </a>
            </div>
            <nav>
                <ul>
                    <li>
                        <a href="/">Home</a>
                    </li>
                    <li class="with_child">
                        <a href="/blog/">Blog</a>
                        <div class="child">
                            <div class="container">
                                <ul>
                                    <li>
                                        <a href="/blog/article/">Post</a>
                                    </li>
                                </ul>
                            </div>
                        </div>
                    </li>
                </ul>
            </nav>
        </div>
    </header>
    <main>
        <h1>404</h1>
    </main>
</body>
</html>
```

Адмін-панель

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="icon" href="/favicon.ico">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vite App</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```

package.json

```
{
  "name": "admin",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "lint": "eslint . --ext .vue,.js,.jsx,.cjs,.mjs --fix --ignore-path .gitignore",
    "format": "prettier --write src/"
  },
  "dependencies": {
    "@tinymce/tinymce-vue": "^5.1.1",
    "axios": "^1.6.2",
    "tinymce": "^6.7.2",
    "vue": "^3.3.4",
    "vue-router": "^4.2.5"
  },
  "devDependencies": {
    "@rushstack/eslint-patch": "^1.3.3",
    "@vitejs/plugin-vue": "^4.4.0",
    "@vitejs/plugin-vue-jsx": "^3.0.2",
    "@vue/eslint-config-prettier": "^8.0.0",
    "eslint": "^8.49.0",
    "eslint-plugin-vue": "^9.17.0",
    "prettier": "^3.0.3",
    "sass": "^1.69.5",
    "vite": "^4.4.11"
  }
```

```
}
```

main.js

```
import './assets/main.css'

import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

const app = createApp(App)

app.use(router)

app.mount('#app')
```

App.vue

```
<script setup>
import { RouterLink, RouterView } from 'vue-router'
import Header from './components/Header.vue'

</script>

<template>
  <Header />
  <RouterView />
</template>
```

HomeView.vue

```
<script setup>
import TheWelcome from '../components/TheWelcome.vue'
import Authorization from '../components/Authorization.vue'
import Signup from '../components/Signup.vue'
</script>

<template>
  <main>
    <Signup />
    <Authorization />
    <TheWelcome />
  </main>
</template>
```

router/index.js

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'
```

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView
    },
    {
      path: '/posts',
      component: () => import('@/components/blog/posts/Main.vue'),
      meta: {
        title: 'Posts'
      },
    },
    {
      path: '/post',
      component: () => import('@/components/blog/posts/Post.vue'),
      meta: {
        title: 'Posts'
      },
    },
    {
      path: '/games',
      component: () => import('@/components/games/Main.vue'),
      meta: {
        title: 'Games'
      },
    },
    {
      path: '/game',
      component: () => import('@/components/games/Game.vue'),
      meta: {
        title: 'Game'
      },
    },
    {
      path: '/pages',
      component: () => import('@/components/pages/Main.vue'),
      meta: {
        title: 'Pages'
      },
    },
    {
      path: '/page',
      component: () => import('@/components/pages/Page.vue'),
      meta: {
        title: 'Page'
      },
    },
  ]
})
```

```
export default router;
```

Signup.vue

```
<script setup>
import { ref, reactive } from "vue";
import { useRouter } from "vue-router";
import axios from "axios";

defineProps({
  msg: {
    type: String,
    required: true
  }
})
setTimeout(() => {

  console.log(localStorage.getItem("logged"));
  if (localStorage.getItem("logged") == "true") {
    console.log(1);
    console.log(document.querySelector('.login'));
    document.querySelector('.login') ? document.querySelector('.login').classList.add('logged') : '';
  }
}, 0);
const submit = async () => {
    let nameValue = document.querySelector('.signup form [name="name"]').value;
    let loginValue = document.querySelector('.signup form [name="login"]').value;
    let passwordValue = document.querySelector('.signup form [name="password"]').value;
    document.querySelector('.signup').classList.remove('error');
    if (loginValue.length > 0 && nameValue.length > 0 && passwordValue.length > 0 ) {
      const { data } = await axios.post('http://localhost:5500/api/user', {
        "user" : {
            "role": "user",
            "name": nameValue,
            "login": loginValue,
            "password": passwordValue
        }
      });
      console.log('data',data);
    } else {
      document.querySelector('.signup form').classList.add('error');
    }
  }
</script>

<template>
  <div class="signup">
    <div class="wrapper">
      <div class="h1">SignUp</div>
        <div class="form_wrapper">
          <form action="#" ref="myDiv">
            <input type="text" name="name" placeholder="Name">
```

```
          <input type="text" name="login" placeholder="Login">
          <input type="password" name="password" placeholder="Password">
          <span class="error">Fill all fields</span>
          <button @click.prevent="submit()" class="btn" type="submit">Submit</button>
        </form>
        <div class="login">LogIn</div>
      </div>
    </div>
  </div>
</template>
```

Authorization.vue

```
<script setup>
import { ref, reactive } from "vue";
import { useRouter } from "vue-router";
import axios from "axios";

defineProps({
  msg: {
    type: String,
    required: true
  }
})
setTimeout(() => {

  console.log(localStorage.getItem("logged"));
  if (localStorage.getItem("logged") == "true") {
    console.log(1);
    console.log(document.querySelector('.login'));
    document.querySelector('.login') ? document.querySelector('.login').classList.add('logged') : '';
  }
}, 0);
const submit = async () => {
    let loginValue = document.querySelector('.login form [name="login"]').value;
    if (loginValue.length > 0) {
      try {
        document.querySelector('.login form').classList.remove('error');
        const response = await axios.get(`http://localhost:5500/api/user?login=${loginValue}`);
        console.log('response:', response);
        if (response.status == 200) {
          document.querySelector('.login form').classList.remove('error');
          if (document.querySelector('.login form [name="password"]').value ==
response.data.user.password) {
            localStorage.setItem("logged", "true");
            localStorage.setItem("role", response.data.user.role);
            document.querySelector('.login').classList.add('logged');
          } else {
            document.querySelector('.login form').classList.add('error');
          }
        } else {
          document.querySelector('.login form').classList.add('error');
        }
```

```
    } catch (error) {
      document.querySelector('.login form').classList.add('error');
    }

  }
 }
</script>

<template>
 <div class="login">
  <div class="wrapper">
   <div class="h1">Login</div>
    <div class="form_wrapper">
     <form action="#" ref="myDiv">
      <input type="text" name="login" placeholder="Login">
      <input type="password" name="password" placeholder="Password">
      <span class="error">Wrong login or password</span>
      <button @click.prevent="submit()" class="btn" type="submit">Submit</button>
     </form>
     <div class="signup">SignUp</div>
    </div>
  </div>
 </div>
</template>
```

Header.vue

```
<script setup>
const logout = async () => {
 localStorage.removeItem("logged");
 localStorage.removeItem("role");
 if (document.querySelector('.login')) {
  console.log(document.querySelector('.login'));
  document.querySelector('.login').classList.remove('logged');
 }
}
</script>

<template>
 <header>
  <div class="container">
   <ul class="menu">
    <li>
      <router-link to="/posts/"> Posts </router-link>
    </li>
    <li>
      <router-link to="/games/"> Games </router-link>
    </li>
    <li>
      <router-link to="/pages/"> Pages </router-link>
    </li>
   </ul>
   <div class="logout"  @click="logout()">
```

```
      Logout
    </div>
  </div>
 </header>
</template>
```

Main.vue

```
<template>
  <div class="all_pages">
    <div class="top">
      <h1>PAGES</h1>
      <router-link to="/page/" class="btn"> + Create page </router-link>
    </div>
    <div class="pages_list">
      <div class="page names">
        <div>Name</div>
        <div>Slug</div>
        <div>Date</div>
      </div>
      <template v-for="page in data.value" :key="page.h1" class="pages_list">
        <div class="page">
          <div>{{ page.name }}</div>
          <div>{{ page.slug }}</div>
          <div>{{ new Date(page.date).toDateString() }}</div>
        </div>
      </template>
    </div>
  </div>
</template>

<script>
import { ref, reactive, onMounted } from "vue";
import { useRoute, useRouter } from "vue-router";
import axios from 'axios';

export default {
   setup () {
      const router = useRouter();
      const route = useRoute();
      let data = reactive([]);
      onMounted(async () => {
        console.log('1:', 1)
        const response = await axios.get('http://localhost:5500/api/pages/all');
        console.log('response:', response)
        data.value = response.data.pages;
        console.log('data',data.value);
```

```
        });

        return {data}
    },
};
</script>
```

Game.vue

```
<template>
  <div class="wrapper">
    <h1>Create Game</h1>
  <form action="#">

    <div class="title loader">Auto fill</div>

    <input type="text" placeholder="Enter game's title" name="gamesTitle">

    <div class="title">General</div>

    <input type="text" placeholder="Genre" name="genre">
    <input type="text" placeholder="Slug" name="slug">
    <input type="text" placeholder="Name" name="name">

    <div class="title">Meta</div>
    <input type="text" placeholder="Meta title" name="metaTitle">
    <input type="text" placeholder="Meta description" name="metaDescription">

    <div class="title">Content</div>

    <div>
      <editor id="editor"
        api-key="no-api-key"
        :init="{
        height: 500,
        menubar: false,
        plugins: [
          'a11ychecker','advlist','advcode','advtable','autolink','checklist','export',
          'lists','link','image','charmap','preview','anchor','searchreplace','visualblocks',
          'powerpaste','fullscreen','formatpainter','insertdatetime','media','table','help','wordcount'
        ],
        toolbar:
          'undo redo | casechange blocks | bold italic backcolor | \
          alignleft aligncenter alignright alignjustify | \
          bullist numlst checklist outdent indent | removeformat | a11ycheck code table help'
        }"
        initial-value="Welcome to TinyMCE Vue"
      />
    </div>

    <button type="submit" class="btn" @click.prevent="submit()">Submit</button>
  </form>
```

```
    </div>
</template>

<script setup>
    import axios from 'axios';
    const submit = async () => {
        let editorContent = tinymce.get("editor").getContent();
        console.log(editorContent);
        const { data } = await axios.post('http://localhost:5500/api/game', {
            "game" : {
                "genre": document.querySelector('form input[name="genre"]').value,
                "slug": document.querySelector('form input[name="slug"]').value,
                "name": document.querySelector('form input[name="name"]').value,
                "meta": {
                    "title": document.querySelector('form input[name="metaTitle"]').value,
                    "description": document.querySelector('form input[name="metaDescription"]').value
                },
                "content": {
                    "html": editorContent
                },
                "date": new Date(Date.now())
            }
        });
        console.log("data", data);
  }
</script>

<script>
import Editor from '@tinymce/tinymce-vue'

export default {
name: 'app',
components: {
  'editor': Editor
},

props: {

    tabindex: {
      type: Number,
      required: false,
      default: 0,
    },
  },
  data() {
    this.options = ["Article", "News"];
    this.default = "Article";
    return {
      selected: this.default
        ? this.default
        : this.options.length > 0
        ? this.options[0]
        : null,
```

```
      open: false,
    };
  },
  mounted() {
    this.$emit("input", this.selected);
  },
}
</script>

<style scoped lang="scss">
.wrapper {
    width: 100%;
    max-width: 700px;
    padding: 50px 0;
}
h1 {
    color: #fff;
    text-align: center;
    font-size: 55px;
}

form, form input, form button {
    width: 100%;
    max-width: 100%;
}
.title {
    font-size: 32px;
}
input {
    width: 100%;
    display: block;
    height: 40px;
    margin-bottom: 20px;
}

.custom-select {
  position: relative;
  width: 100%;
  text-align: left;
  outline: none;
  height: 40px;
  line-height: 40px;
  margin-bottom: 20px;
}

.custom-select .selected {
  background-color: #fff;
  border-radius: 6px;
  border: 1px solid #E8EAEB;
  color: #000;
  padding-left: 1em;
  cursor: pointer;
  user-select: none;
```

```
}

.custom-select .selected.open {
  border: 1px solid #00bd7e;
  border-radius: 6px 6px 0px 0px;
}

.custom-select .selected:after {
  position: absolute;
  content: "";
  top: 22px;
  right: 1em;
  width: 0;
  height: 0;
  border: 5px solid transparent;
  border-color: #00bd7e transparent transparent transparent;
}

.custom-select .items {
  color: #000;
  border-radius: 0px 0px 6px 6px;
  overflow: hidden;
  border-right: 1px solid #00bd7e;
  border-left: 1px solid #00bd7e;
  border-bottom: 1px solid #00bd7e;
  position: absolute;
  background-color: #fff;
  left: 0;
  right: 0;
  z-index: 1;
}

.custom-select .items div {
  color: #000;
  padding-left: 1em;
  cursor: pointer;
  user-select: none;
}

.custom-select .items div:hover {
  background-color: #00bd7e;
}

.selectHide {
  display: none;
}
.loader {
  position: relative;
  &::after {
    content: '';
    position: absolute;
    right: -20px;
    top: 10px;
```

```
  width: 25px;
  height: 25px;
  border-radius: 50%;
  border: 4px solid #00bd7e;
  border-top-color: #fff;
  transform: rotate(45deg);
 }
}
</style>
```

Games/Main.vue

```
<template>
  <div class="all_games">
    <div class="top">
      <h1>GAMES</h1>
      <router-link to="/game/" class="btn"> + Create game </router-link>
    </div>
    <div class="games_list">
      <div class="game names">
        <div>Name</div>
        <div>Genre</div>
        <div>Slug</div>
        <div>Date</div>
      </div>
      <template v-for="game in data.value" :key="game.h1" class="games_list">
        <div class="game">
          <div>{{ game.name }}</div>
          <div>{{ game.genre }}</div>
          <div>{{ game.slug }}</div>
          <div>{{ new Date(game.date).toDateString() }}</div>
        </div>
      </template>
    </div>
  </div>
</template>

<script>
import { ref, reactive, onMounted } from "vue";
import { useRoute, useRouter } from "vue-router";
import axios from 'axios';

export default {
   setup () {
      const router = useRouter();
      const route = useRoute();
      let data = reactive([]);
      onMounted(async () => {
        const response = await axios.get('http://localhost:5500/api/games/all');
        console.log('response:', response)
        data.value = response.data.games;
        console.log('data',data.value);
```

```
        });

        return {data}
    },
};
</script>

<style scoped lang="scss">
.all_games {
    display: flex;
    flex-direction: column;
    margin-top: 70px;
    width: 700px;
}
.top {
    display: flex;
    gap: 20px;
    align-items: center;
    justify-content: center;
    height: 50px;
}

.games_list {
    height: calc(100vh - 200px);
    border-radius: 20px;
    margin-top: 20px;
    background-color: #fff;
    overflow: auto;
    .game {
        display: flex;
        color: #000;
        border-bottom: 1px solid #000;
        background-color: #fff;
        &:nth-child(2n-1) {
            background-color: #efefef;
        }
        &.names {
            background-color: #00bd7e;
            & > div {
                color: #fff;
            }
        }
        &:last-child {
            border-right: none;
        }
        & > div {
            width: 33%;
            color: #000;
            text-align: center;
            padding: 10px;
            border-right: 1px solid #000;
            &:last-child {
                border-right: none;
```

```
        }
      }
    }
}

</style>
```

Серверна частина

index.js

```
import express from 'express';
import bodyParser from 'body-parser';
import cors from 'cors';
import path from 'path';
import mongoose from 'mongoose';

import fsSync from 'fs';
const fs = fsSync.promises;

import pagesRoutes from './routes/pages.js';
import gamesRoutes from './routes/games.js';
import postsRoutes from './routes/posts.js';
import usersRoutes from './routes/users.js';

import { SSR } from './SSR.js';
const ssr = new SSR();

import { RouteHandler } from './RouteHandler.js';
const routeHandler = new RouteHandler();

import { ImageHandler } from './ImageHandler.js';
const imageHandler = new ImageHandler();

import axios from 'axios';

import { PORT, routes } from './config.js';
import { log } from 'console';
mongoose.Promise = global.Promise;

const __dirname = path.resolve();

const app = express()
const port = 5500

app.use( bodyParser.json() );

app.use(bodyParser.urlencoded({
 extended: true}));
app.use(cors())

const router = express.Router();
```

```
router.use(pagesRoutes);
router.use(gamesRoutes);
router.use(postsRoutes);
router.use(usersRoutes);

app.use('/api', router);

app.listen(port, ()=>{
   console.log(`Server is runing on port ${port}`)
})

app.get('*', async (req, res) => {
   console.log('req',req.params[0],req.query);
   let ssrQuery = req.query['ssr'] ? req.query.ssr : false;
   let url = req.params[0];
   console.log('ssrQuery',ssrQuery);
   if (ssrQuery) {
      await ssr.generatePage(url)
   }
   let MODE = 'production'
   let currentUrl = new URL('http://localhost:5000' + url);

   if(currentUrl.pathname.indexOf('.') === -1 && currentUrl.pathname.lastIndexOf('/') !==
currentUrl.pathname.length - 1) {
      return res.redirect(301, currentUrl.pathname + '/' + currentUrl.search);
   }
   console.log(url);
   const srcFolder = url.split('/')[1] === 'admin' ? '/admin' : '/dist';
   if (srcFolder === '/admin' && !url.split('/')[2]) {
      res.sendFile(`${__dirname}/admin/index.html`)
   } else if (url === '/') {
      res.sendFile(`${__dirname}/dist/index.html`)
   } else {
      const route = routeHandler.findSlugTemplate(routes, url);
      console.log('asd', route.found)

      if (route.found === true) {
         res.sendFile(`${__dirname}${srcFolder}${route.index}`)
      } else {
         const file = await routeHandler.findStaticFile(url, srcFolder);

         if(file) {

            if(file === true) {
               return res.send('OK');
            }

            if(file.type === 'asset') {
               if(imageHandler.detectImage(req.originalUrl) && MODE === 'production') {
                  res.removeHeader('access-control-allow-origin');
               }
               return res.sendFile(file.path);
```

```
            } else if(file.type === 'html') {
                route.index = file.path;
                route.path = url;
                route.route = url;
                route.queryParams = `?path=${url}`;
            } else if(file.type === 'error') {
                return res.status(404).send(file.error);
            }

        } else {
            if(fsSync.existsSync('http://localhost:5000' + url)) {
                return res.sendFile('http://localhost:5000' + url);
            }
            let errorPage;
            const errorPageExists = fsSync.existsSync('http://localhost:5000' + '/404.html');

            if(errorPageExists) {
                errorPage = await ssr.generate('http://localhost:5000', website, { index: '/404.html',
route: '/404/', queryParams: '?path=/404/', paramsObject: {} }, chapters, auth_key, 'dist',
disableImagesRegeneration, preview, disableBundling)
            } else {
                errorPage = '404';
            }

            return res.status(404).send('<!DOCTYPE html>' + errorPage);
        }

    }
  }

})

mongoose.connect('mongodb://127.0.0.1:27017/game-cms');

mongoose.connection
  .once('open', () => {
    console.log('Mongoose - successful connection...');
    app.listen(PORT, () => {
      console.log(`Server started on port ${PORT}`);
    });
  })
  .on('error', error => console.error(error));
```

RouteHandler.js

```
import fsSync from 'fs';

import { path_to_website } from './config.js';

import { ImageHandler } from './ImageHandler.js';
import { log } from 'console';
const imageHandler = new ImageHandler();
```

```
export class RouteHandler {
   constructor () {

   }
   findSlugTemplate(routes, url) {
      let urlArr = url.split('/');
      urlArr = urlArr.flatMap(item => {
         if (item !== '') {
            return [item];
         }
         return [];
      });

      let result = {
         found: false,
         paramsObject: {},
         queryParams: '?'
      }

      if(urlArr[urlArr.length - 1]?.indexOf('.') !== -1) {
         return result;
      }

      for (let route of routes) {
         result = {
            found: false,
            paramsObject: {},
            queryParams: '?'
         }
         let schemeArr = route.route.split('/');
         schemeArr = schemeArr.flatMap(item => {
            if (item !== '') {
               return [item];
            }
            return [];
         });

         for (let index in schemeArr) {
            if (schemeArr.length !== urlArr.length) {
               result.found = false;
               break;
            }
            if (schemeArr[index].indexOf(':') === -1) {
               if (schemeArr[index] === urlArr[index]) {
                  result.found = true;
                  result = { ...route, ...result };
                  result.paramsObject.routeObject = JSON.stringify(route);
               } else {
                  result.found = false;
                  break;
               }
            } else {
               if (schemeArr[index] && urlArr[index]) {
```

```
                result.found = true;
                result = { ...route, ...result };
                let property = schemeArr[index].substring(1, schemeArr[index].length);
                let value = urlArr[index];
                result.paramsObject[property] = value;
            } else {
                result.found = false;
                break;
            }
        }
    }

    if (result.found === true) {
        result.paramsObject.path = '/' + urlArr.join('/') + '/';
        break;
    }
  }

  if (result.found) {

    for (let param in result.paramsObject) {
        result.queryParams += `${param}=`;
        result.queryParams += `${result.paramsObject[param]}&`;
    }

    result.queryParams = result.queryParams.substring(0, result.queryParams.length - 1);
  }

  return result;
 }

 async findStaticFile(url, srcFolder, mode = 'ssg') {
  const isFile = url.split('/')[url.split('/').length - 1].indexOf('.') !== -1;
  url = url.lastIndexOf('/') === url.length - 1 ? url.substring(0, url.length - 1) : url

  const filePath = path_to_website + srcFolder + url;

  if(isFile) {

    if(imageHandler.detectImage(filePath)) {
        if(mode === 'ssg' && fsSync.existsSync(path_to_website + srcFolder + url)) {
            return await imageHandler.getFromCache('http://localhost:5000', url, path_to_website,
srcFolder);
        } else {
            return await imageHandler.generateImages('http://localhost:5000', url, srcFolder,
path_to_website);
        }
    }

    let isFileExists = fsSync.existsSync(filePath);

    if(isFileExists) {
        return {
```

```
                type: 'asset',
                path: filePath
            }
        }
    } else {
        let isNamedIndexFileExists = fsSync.existsSync(filePath + '.html')
        let isIndexFileExists = fsSync.existsSync(filePath + '/index.html');

        if(isNamedIndexFileExists) {
            return {
                type: 'html',
                path: url + '.html'
            }
        } else if(isIndexFileExists) {
            return {
                type: 'html',
                path: url + '/index.html'
            }
        }
    }
  }
}
```

SSR.js

```
import fs from 'fs';
import path from 'path';

import { parse } from 'node-html-parser';

import { path_to_website } from './config.js';
import { log } from 'console';

export class SSR {
  constructor () {

  }
  async generatePage(url) {
    console.log('ssr url', url);
    console.log('ssr path_to_website', path_to_website + '/dist' + url);
    const __dirname = path.resolve();
    let filePath = path.join(__dirname, '/dist', url);
    console.log('filePath:', filePath)
    let data;
    fs.readFile(filePath, function (error, html) {

      if (error) {
        throw error;
      }

      const user = "Node JS";
      getData(html)

    });
```

```
        function getData (html) {
            let root = Buffer.from(html).toString();
            console.log(root);
            parse(root).querySelector('h1').innerHTML = 'asd';
            fs.writeFile(filePath, Buffer.from(parse(root).querySelector(dbSelector).innerHTML =
dbContent, "utf-8"), 'utf-8', function (err) {
                if (err) throw err;
                console.log('filelistAsync complete');
              });
        }
    }
}
```

ImageHandler.js

```
import axios from 'axios';

import fsSync from 'fs';
const fs = fsSync.promises;
import sharp from 'sharp';

export class ImageHandler {
    constructor() {
        this.imagesGenerationQueue = [];
    }

    detectImage(path) {
        return /\.(jpe?g|tiff?|png|webp|bmp)/g.test(path);
    }

    detectExternalImage(url, urlParams) {
        const fullUrl = new URL('https://example.com' + url + urlParams);
        return fullUrl.searchParams.get('source');
    }

    async getFromCache(website, url, path_to_website, srcFolder) {
        let cachedFilePath = path_to_website + srcFolder + url;
        let isCachedFileExists = fsSync.existsSync(cachedFilePath);

        if (isCachedFileExists) {
            return {
                type: 'asset',
                path: cachedFilePath
            }
        } else {
            return await this.generateImages(website, url);
        }
    }

    generateImages(website, url, buildFolder, path_to_website) {
        const queuedImage = this.imagesGenerationQueue.find(image => image.url === url &&
image.website === website);

        if (queuedImage) {
            return queuedImage.promise;
```

```
    }

    const promise = new Promise(async (resolve, reject) => {

        let filePath = path_to_website + buildFolder + url;
        let cachedFilePath = path_to_website + url;
        if (cachedFilePath.indexOf('&') !== -1) {
            cachedFilePath = cachedFilePath.substring(0, cachedFilePath.indexOf('&'));
        }

        if (this.detectImage(filePath)) {
            return resolve({
                type: 'asset',
                path: cachedFilePath
            });
        }

        let cachedFolderPath = cachedFilePath.split('/').slice(0, cachedFilePath.split('/').length -
1).join('/');

        if (!fsSync.existsSync(cachedFilePath)) {
            await fs.mkdir(cachedFolderPath, { recursive: true });
        }

        if (!fsSync.existsSync(filePath)) {
            if (fsSync.existsSync(cachedFilePath)) {
                return resolve({
                    type: 'asset',
                    path: cachedFilePath
                })
            }
        }

        try {

            await fs.copyFile(filePath, cachedFilePath);

        } catch (err) {

            return resolve({
                type: 'error',
                error: 'File was not found'
            })

        }

        await this.prepareSources(cachedFilePath);

        return resolve({
            type: 'asset',
            path: cachedFilePath
        });
```

```
      }).then(value => {
         const index = this.imagesGenerationQueue.findIndex(image => image.url === url &&
image.website === website);
         if (index != -1) {
            this.imagesGenerationQueue.splice(index, 1);
         }
         return value;
      });

      this.imagesGenerationQueue.push({
         url,
         website,
         promise
      });

      return promise;

   }

   async prepareSources(path) {
      const cachedFilePath = path;
      const image = sharp(cachedFilePath);
      const imageMetadata = await image.metadata();

      if (imageMetadata.width > 600) {
         const extension = cachedFilePath.substring(cachedFilePath.lastIndexOf('.'),
cachedFilePath.length);
         const nameWithoutExtendsion = cachedFilePath.substring(0, cachedFilePath.length -
extension.length);
         const smallSizePath = nameWithoutExtendsion + '-600' + extension;
         try {
            await image.resize(600).toFile(smallSizePath);
         } catch (err) {
            console.log('Error while image resizing: ', err);
         }
      }

      if (imageMetadata.width > 1200) {
         const extension = cachedFilePath.substring(cachedFilePath.lastIndexOf('.'),
cachedFilePath.length);
         const nameWithoutExtendsion = cachedFilePath.substring(0, cachedFilePath.length -
extension.length);
         const smallSizePath = nameWithoutExtendsion + '-1200' + extension;
         try {
            await image.resize(1200).toFile(smallSizePath);
         } catch (err) {
            console.log('Error while image resizing: ', err);
         }
      }

      return true;

   }
```

```
  async downloadExternalImage(website, url, urlParams) {
    const fullUrl = new URL('https://example.com' + url + urlParams);
    const savePath = PATH_TO_CACHE_FOLDER + website + (fullUrl.pathname.indexOf('&')
!== -1 ? fullUrl.pathname.substring(0, fullUrl.pathname.indexOf('&')) : fullUrl.pathname);

    const splittedPath = savePath.split('/');
    splittedPath.pop();
    const pathToFolder = splittedPath.join('/') + '/';

    if (!fsSync.existsSync(pathToFolder)) {
      await fs.mkdir(pathToFolder, { recursive: true });
    }

    const downloadImage = (url, imagePath) =>
      axios({
        url,
        responseType: 'stream',
      }).then(
        response =>
          new Promise((resolve, reject) => {
            response.data
              .pipe(fsSync.createWriteStream(imagePath))
              .on('finish', () => resolve())
              .on('error', e => reject(e));
          }),
      );

    await downloadImage(fullUrl.searchParams.get('source'), savePath);
    await this.prepareSources(savePath);
    return savePath;
  }

}
```

config.js

```
import process from "process";
import { config } from "dotenv";

config();

export const path_to_website = process.cwd()

export const PORT = process.env.PORT;
export const SERVER_PORT = 5000;
export const HOST = '127.0.0.1';

export const routes = [
  {
    route: '/blog/',
    index: '/blog.html'
  },
  {
```

```
        route: '/blog/:post/',
        index: '/blog/post.html'
    },
    {
        route: '/games/',
        index: '/games.html'
    },
    {
        route: '/games/:game/',
        index: '/games/game.html'
    },
]
```

routes/games.js

```
import express from 'express';

import { GamesController } from './../controllers/GameControllers.js';

const router = express.Router();

router.post('/game', GamesController.create);

router.get('/games', GamesController.get);

router.get('/games/all', GamesController.all);

export default router;
```

models/Game.js

```
import mongoose from 'mongoose';
import mongooseSequence from 'mongoose-sequence';

const Schema = mongoose.Schema;

const GameSchema = new Schema({
    genre: { type: String, required: true },
    slug: { type: String, required: true },
    name: { type: String, required: true },
    meta: { type: mongoose.SchemaTypes.Mixed },
    content: {
        html: { type: String }
    },
    date: { type: String, required: true }
});

const GameModel = mongoose.model('games', GameSchema);

export default GameModel;
```

GameControllers.js

```
import Game from './../models/Game.js';

export class GamesController {
```

```
static async create(req, res) {
    const { game } = req.body;

    if(!game) {
        return res.status(400).json({
            message: 'Missing game object'
        });
    }

    if( !game.name || !game.slug) {
        return res.status(400).json({
            message: 'Missing game parameters'
        });
    }

    if(await Game.findOne({ name: game.name }) || await Game.findOne({ name: game.slug })) {
        return res.status(400).json({
            message: 'Game with this name or slug already exists'
        });
    }

    const gameModel = new Game({...game});

    await gameModel.save();

    return res.status(200).json({
        message: 'Success'
    });
}

static async get(req, res) {
    const { id, slug } = req.query;

    if(!id && !slug) {
        return res.status(400).json({
            message: 'Missing id or slug'
        });
    }

    const game = await Game.findOne({ $or: [ { name: id }, { slug } ] });
    if(!game) {
        return res.status(404).json({
            message: 'Game not found'
        });
    }

    return res.status(200).json({
        message: 'Success',
        game
    });
}
```

```
    static async all(req, res) {

        const games = await Game.find();

        return res.status(200).json({
            message: 'Success',
            games
        });
    }

}
```

Додаток Г – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

Розробка методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом

Розробка методів і програмних засобів веб-платформи про відеоігри для підвищення ефективності керування контентом

АВТОР: СТ.ГР 2ПІ-22М ПАЛЯНИЦЯ Д. Р.

КЕРІВНИК: К.Т.Н. ДОЦЕНТ КАТЄЛЬНІКОВ Д. І.

Рисунок Г.1 – Титульний слайд

Мета та завдання дослідження

Метою роботи є підвищення продуктивності клієнт-серверної взаємодії при використанні веб-платформи без втрати можливості індексації сторінок веб-ресурсу роботами пошукових систем.

Основними задачами дослідження є:

- провести аналіз існуючих методів і засобів рендерингу та генерації веб сторінок;
- запропонувати новий метод, що буде поєднувати переваги існуючих методів, та перекривати їх недоліки;
- розробити програмні компоненти та систему візуалізації на основі запропонованого методу;
- провести експериментальні дослідження розроблених засобів рендерингу та генерації веб-сторінок.

Рисунок Г.2 – Мета та завдання дослідження

Опис дослідження

Об’єкт дослідження: процес розробки веб-платформ для відеоігор та інших видів мультимедійного контенту.

Предмет дослідження: методи та програмні засоби серверної частини веб-ресурсу для отримання користувачем статичних сторінок та можливістю вибіркового перерендерення сторінок адміністратором.

Методи дослідження. У процесі дослідження використовувались: теорія баз даних при організації зберігання контенту, теорія розподілених систем для побудови розподіленої клієнт-серверної веб-платформи, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень, методи об’єктно-орієнтованого програмування та шаблони програмування для розробки архітектури класів та сервісів програмного забезпечення.

Рисунок Г.3 – Опис дослідження

Наукова новизна

Подальшого розвитку отримав метод рендерингу веб-сторінок, у якому, на відміну від існуючих методів рендерингу, використовується комбінація SSG- та SSR- генерації, що дозволяє, з одного боку, підвищити продуктивність клієнт-серверної взаємодії при використанні веб-платформи, і, з іншого боку, забезпечити можливість кращої індексації сторінок веб-ресурсу роботами пошукових систем.

Рисунок Г.4 – Наукова новизна

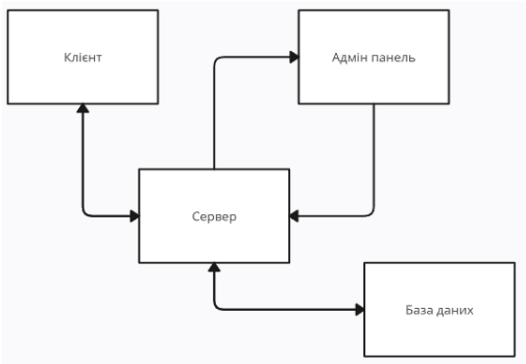
Апробація матеріалів магістерської кваліфікаційної роботи

Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях:

- III Всеукраїнська науково – технічна конференція молодих вчених, аспірантів та студентів «Комп'ютерні ігри і мультимедіа як інноваційний підхід до комунікації - 2023»(Одеса, 2023);
- Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (20-21 листопада, 2023, м.Суми/Вінниця).

Рисунок Г.5 – Апробація матеріалів магістерської кваліфікаційної роботи

Складові системи



- Клієнтська частина;
- Адмін панель;
- Серверна частина.

Рисунок Г.6 – Складові системи

Клієнтська частина

Цю частину системи являє собою сукупність HTML, CSS та JavaScript файлів. Ці файли мають свою структуру, ця структура файлів визначає і структуру сайту. HTML файли у системі – це по суті шаблони, вони є «прямі» та «не прямі». Різниця у тому, що «прямі» шаблони – це файли, що називаються так, як slug сторінки («games.html» -> «/games/»), а «непрямі» – це більш універсальні шаблони, для прикладу, шаблон статей. Замість створення окремого html-файлу для кожної статті, використовується загальний шаблон, в який поміщаються дані з різних статей. У такому випадку один шаблон використовується для всіх статей, і дані у шаблони заповнюються динамічно. (в папці «games» файл «game.html» -> «/games/the-witcher-3/» або «/games/minecraft/»).

Білдиться проект за допомогою Webpack (бандлер) у папку «dist» із такою ж структурою як і в корні проекту. І вже з файлами папки «dist» взаємодіє сервер.

Рисунок Г.7 – Клієнтська частина

Адмін панель

Адмін панель має розподілення користувачів по ролям при авторизації, що відповідно надає привілеї певним користувачам, також присутня авторизація користувача та ряд функцій керування контентом, такі як створення, редагування та видалення постів та ігор, а також сторінок. У вікні додавання нової гри присутня функція автоматичного заповнення полів за допомогою REST API до відкритої бази даних IGDB за персональним API-ключем.

Для маніпулюванням даними в базі даних використовуються різні API-реквести на сервер.

(create, get, all)

Для введення контенту використовується едітор tinymce

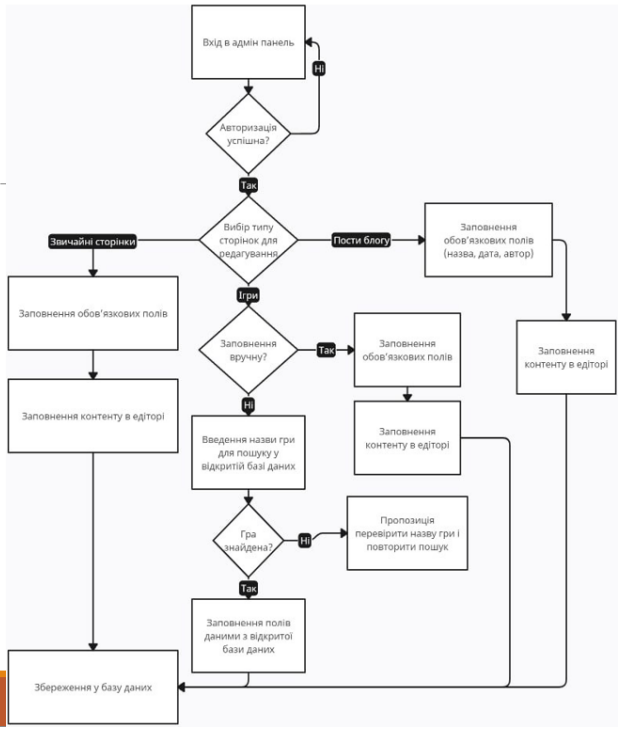


Рисунок Г.8 – Адмін панель

Серверна частина (роути)

Алгоритм перевірки роутів відбувається за допомогою таких кроків:

1. Спочатку проводиться перевірка, чи є вказаний роут роутом для рендерингу сторінки. Якщо це так, то викликається метод для рендерингу звичайної сторінки. В іншому випадку аналіз продовжується.
2. Наступним етапом є превірка, чи це роут на сторінку, чи роут до адмін панелі. Якщо роут підходить під один із двох варіантів, то на клієнт відправляється сайт, що був у запиті. В іншому випадку це означає, що це ні один з цих варіантів, тоді це запит на файл.
3. Останньою є перевірка наявності файлу з запиту. Якщо файл є у наявності, то відбувається передача файлу на клієнтську частину. В іншому разі повертається сторінка 404, що свідчить про відсутність файлу із запиту.

Сервер відрізняє роути до «прямих»(/blog/games/top-games/) та «не прямих»(/blog/:category/:post/) шаблонів

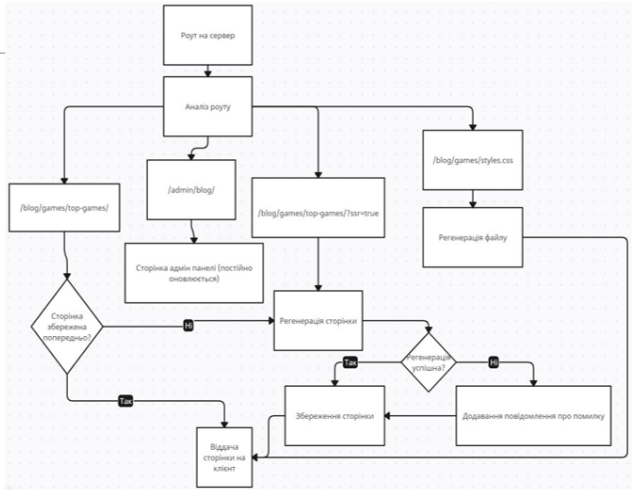


Рисунок Г.9 – Серверна частина (роути)

Серверна частина (зображення)

Файл зображень призначений для роботи з усіма зображеннями на сайті, і містить кілька методів:

1. Метод віддачі збереженого раніше зображення, цей метод включає перевірку наявності зображення та віддачу шляху до цього зображення.
2. Метод генерації зображення, що шукає зображення, що не було збережене раніше, в проекті та аналізує шлях по якому його потрібно зберегти та зберігає його. Після збереження файлу відбувається його конвертація у формат webp.
3. Метод зжимання зображень спрацьовую на кінцевому етапі для оптимізації, принцип полягає у тому, що якщо зображення за розмірами більше за 600 або 1200 пікселів, то метод зберігає зберігає його копії з врізаними розмірами.

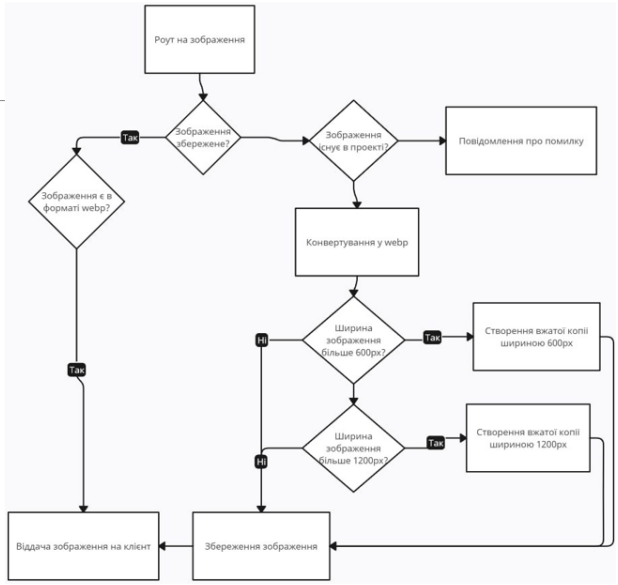


Рисунок Г.10 – Серверна частина (зображення)

Серверна частина (генерація сторінок)

Головна перевага комбінованого методу – зберігання згенерованих сторінок у серверний кеш (SSR) та швидка віддача готових сторінок на клієнт з кешу (SSG). Якщо сервер отримав роут за яким не знайшов потрібного файлу у кеші, то виконується пошук файлу за роутом у папці «dist», та відбувається його генерація. Файл, що виконує генерацію містить два методи:

1. Перший метод приймає назву сайту та масив з роутами, які потрібно перерендерити, після цього спрацьовує цикл, у якому до кожного роута шукається відповідний файл – шаблон, відправляється запит до бази даних для отримання даних по цій сторінці, а потім вставляє потрібний контент туди, куди необхідно. При успішному рендерингу цей файл заміняє старий і зберігається, а у випадку виникнення помилки активною залишається стара сторінка, а у консоль виводиться повідомлення про помилку.
2. Другий же метод перевіряє чи існує директорія для файлу, який потрібно зберегти та зберігає його, а у випадку якщо такої директорії не існує, то створює її.

Рисунок Г.11 – Серверна частина (генерація сторінок)

Використані програмні засоби

Програмні засоби та технології

API

Рисунок Г.12 – Використані програмні засоби

Адмін панель. Реєстрація та авторизація

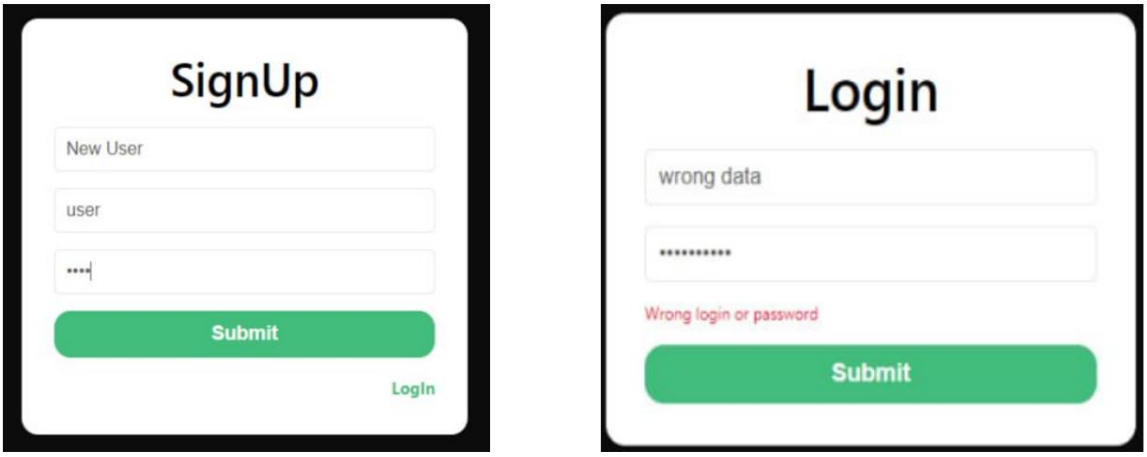


Рисунок Г.13 – Адмін панель. Реєстрація та авторизація

Адмін панель. Блог

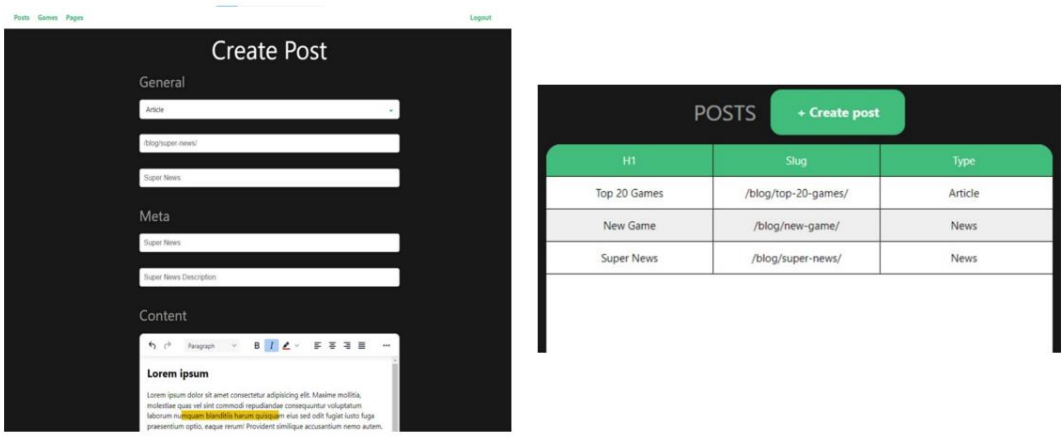


Рисунок Г.14 – Адмін панель. Блог

Адмін панель. Ігри

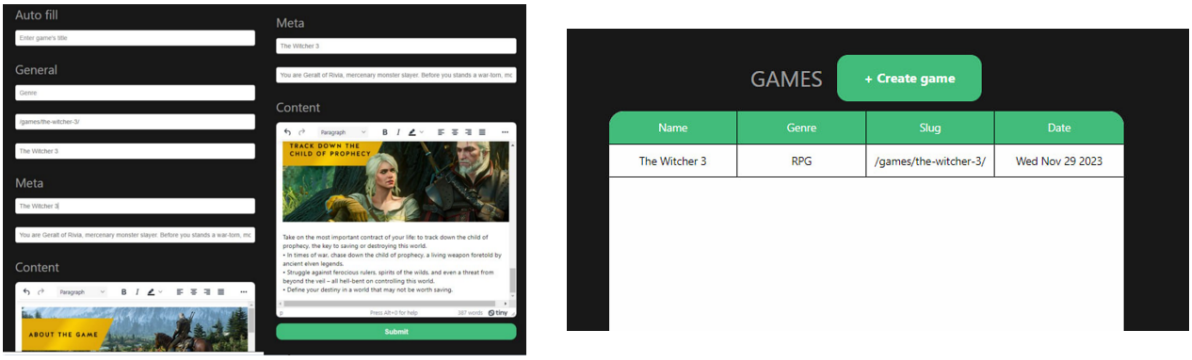


Рисунок Г.15 – Адмін панель. Ігри

Клієнтська частина. Ігри

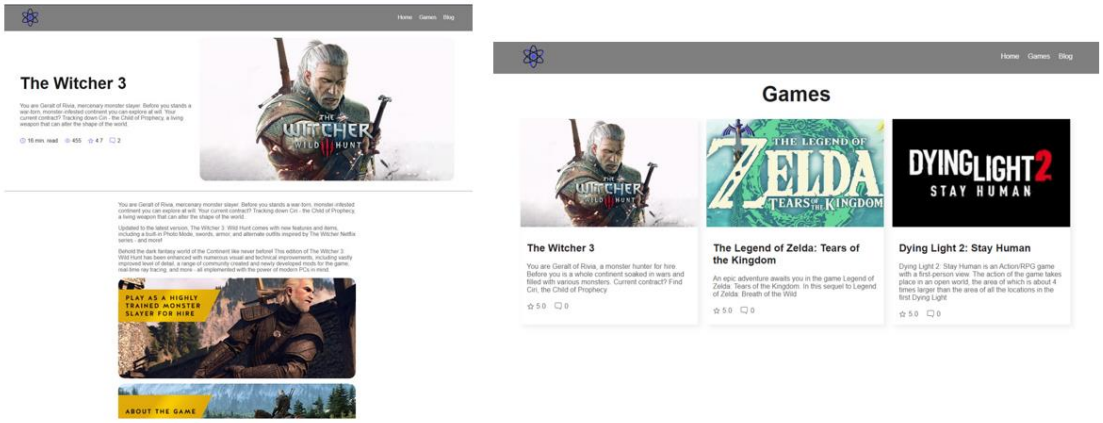


Рисунок Г.16 – Клієнтська частина. Ігри

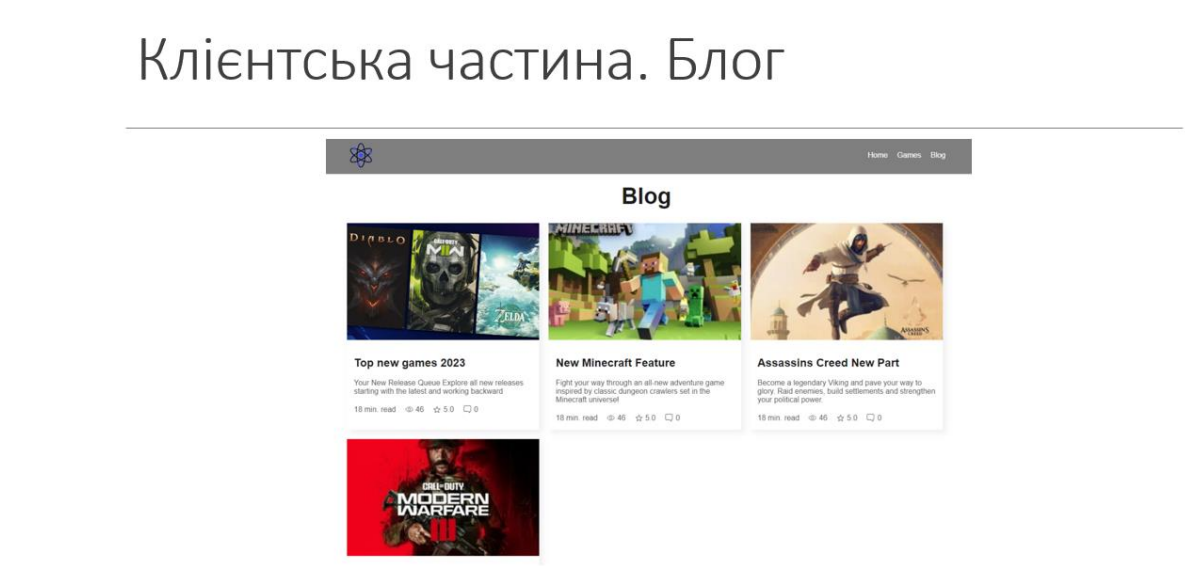


Рисунок Г.17 – Клієнтська частина. Блог

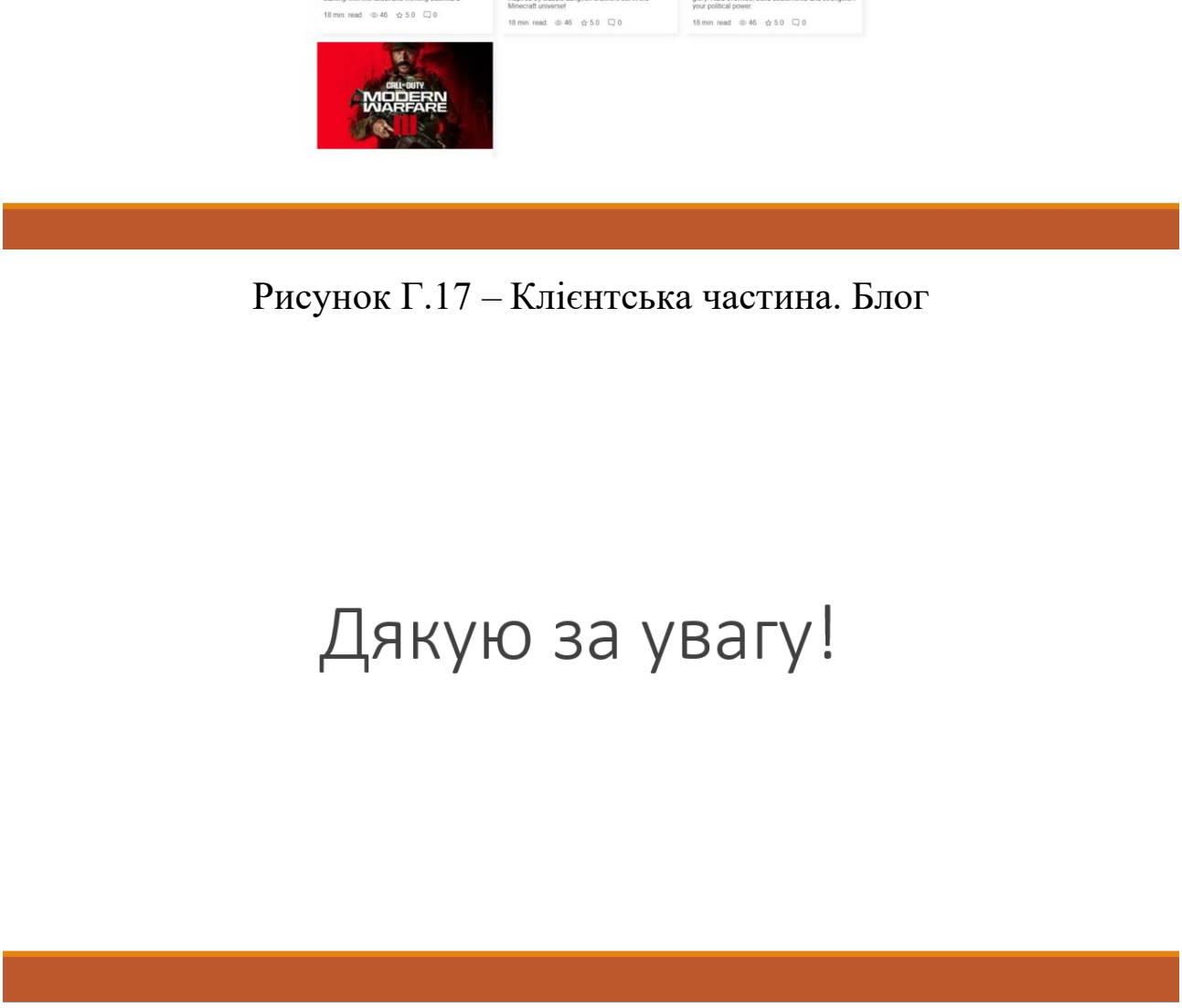


Рисунок Г.18 – Дякую за увагу