

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**СТВОРЕННЯ ВЕБПЛАТФОРМИ З БРАУЗЕРНИМИ ІГРАМИ
ВИКОРИСТОВУЮЧИ REACT І CONSTRUCT 2**

**CREATING A WEB PLATFORM WITH BROWSER GAMES USING
REACT AND CONSTRUCT 2**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Миколайчук Ярослав Святославович

Керівник:
Ящук Андрій Анатолійович

Кваліфікаційну роботу
допущено до захисту
«__» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Миколайчуку Ярославу Святославовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Створення вебплатформи з браузерними іграми використовуючи React і Construct 2»

Керівник роботи: доцент Ящук А. А.

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: предметна область управління персоналом і задачами, технології розробки програмного забезпечення (React, TypeScript), засоби роботи з базами даних (SupaBase, CloudFlare), вимоги до інформаційних систем та забезпечення інформаційної безпеки.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз сучасного стану проблеми, постановка завдання на кваліфікаційну роботу, формування вимог до розроблюваної інформаційної системи, обґрунтування вибору технологій та засобів розробки, практична реалізація об'єкта проєктування, розробка структури бази даних, обґрунтування принципів захисту інформаційно-комп'ютерної системи, тестування та налагодження програмного забезпечення.

5. Перелік графічного матеріалу: 23 рисунки, 19 лістингів, 7 таблиць.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	Ящук А. А.		
Специфікація вимог до розробленої системи	Ящук А. А.		
Розробка об'єкта проектування	Ящук А. А.		
Нормоконтроль	Суринович О. М.		
Гарант ОП	Ліщина Н. М.		
Показник запозичень тексту	_____ %		
Академічна доброчесність	Ящук А. А.		

7. Дата видачі завдання «3» лютого 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об'єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Ярослав МИКОЛАЙЧУК

Керівник кваліфікаційної роботи _____

Андрій ЯЩУК

АНОТАЦІЯ

Миколайчук Я. С. Створення вебплатформи з браузерними іграми використовуючи React і Construct 2. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків та списку використаних джерел.

У першому розділі досліджено предметну область браузерних ігор, проаналізовано існуючі веб-платформи та обґрунтовано актуальність створення власного програмного рішення. У другому розділі визначено вимоги до веб-платформи, описано ролі користувачів, структуру застосунку та обґрунтовано вибір технологій розробки. У третьому розділі описано практичну реалізацію платформи, структуру бази даних, роботу з файлами ігор, створення ігор у Construct 2, тестування, налагодження та розгортання проєкту на Netlify.

У висновках підсумовано результати виконаної роботи та підтверджено досягнення поставленої мети.

Ключові слова: веб-платформа, браузерні ігри, React, TypeScript, Construct 2, HTML5, Supabase, Cloudflare R2, база даних, Netlify.

ABSTRACT

Mykolaichuk Ya. S. Creating a web platform with browser games using React and Construct 2. Manuscript. Bachelor's qualification thesis of the Educational Program "Software Engineering" in the specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, and a list of references.

The first chapter examines the subject area of browser games, analyzes existing web platforms, and substantiates the relevance of creating a custom software solution. The second chapter defines the requirements for the web platform, describes user roles, the application structure, and justifies the choice of development technologies. The third chapter describes the practical implementation of the platform, the database structure, working with game files, game development in Construct 2, testing, debugging, and deployment of the project on Netlify.

The conclusions summarize the results of the completed work and confirm the achievement of the stated goal.

Keywords: web platform, browser games, React, TypeScript, Construct 2, HTML5, Supabase, Cloudflare R2, database, Netlify.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ СТВОРЕННЯ ВЕБ-ПЛАТФОРМ З БРАУЗЕРНИМИ ІГРАМИ ТА ПОСТАНОВКА ЗАВДАННЯ	3 10
1.1 Аналіз сучасного стану проблеми	10
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	12
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ	14
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення	14
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання	22
РОЗДІЛ 3 РОЗРОБКА ВЕБ-ПЛАТФОРМИ З БРАУЗЕРНИМИ ІГРАМИ.....	30
3.1 Практична реалізація об'єкта проектування	30
3.2 Розробка бази даних.....	51
3.3 Розробка та експорт браузерних ігор у Construct 2	56
3.4 Тестування та налагодження веб-платформи.....	61
ВИСНОВКИ.....	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	68

ВСТУП

Актуальність теми. У сучасних умовах розвитку цифрових технологій браузерні ігри залишаються одним із доступних і поширених форматів інтерактивного контенту. Їхньою перевагою є можливість запуску без встановлення додаткового програмного забезпечення, оскільки користувачу достатньо мати браузер та доступ до мережі Інтернет. Завдяки цьому браузерні ігри можуть використовуватися не лише для розваг, а й як приклад практичної реалізації веб-технологій, інтеграції мультимедійного контенту та організації взаємодії користувача з веб-застосунком.

Розвиток HTML5, JavaScript та сучасних frontend-технологій значно розширив можливості створення браузерних ігор і платформ для їх розміщення. Сьогодні існує багато онлайн-сервісів, які надають доступ до ігрового контенту без необхідності встановлення ігор на пристрій користувача. Такі платформи зазвичай містять каталог ігор, систему пошуку, фільтрацію, сторінки окремих ігор та інші елементи взаємодії з користувачем. Водночас більшість готових платформ є закритими сервісами, не дають можливості дослідити їхню внутрішню архітектуру, а також не пристосовані для навчального або демонстраційного використання в межах власного програмного проєкту.

Актуальність теми кваліфікаційної роботи зумовлена необхідністю створення власної веб-платформи, яка дозволяє розміщувати, переглядати та запускати браузерні HTML5-ігри безпосередньо у браузері. Така платформа дає змогу поєднати сучасні засоби frontend-розробки, базу даних, хмарне зберігання файлів і інтеграцію ігор, створених за допомогою Construct 2 [1]. Розробка такого програмного продукту є практично доцільною, оскільки дозволяє дослідити повний цикл створення веб-застосунку: від проєктування інтерфейсу та структури даних до розгортання готового рішення на хостинговій платформі.

Метою кваліфікаційної роботи є створення веб-платформи з браузерними іграми з використанням React [2] і Construct 2, яка забезпечує перегляд каталогу

ігор, запуск HTML5-ігор у браузері, взаємодію користувачів із контентом та адміністрування ігрових матеріалів.

Об'єктом дослідження є процес створення та функціонування веб-платформи для розміщення браузерних ігор.

Предметом дослідження є методи, засоби та технології розробки веб-платформи з браузерними іграми з використанням React, TypeScript [3], Supabase [4], Cloudflare R2 [5] та Construct 2.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- здійснити аналіз сучасного стану розвитку браузерних ігор, проаналізувати існуючі аналоги веб-платформ з браузерними іграми, визначити їхні переваги та недоліки;
- визначити функціональні та нефункціональні вимоги до розроблюваної веб-платформи;
- спроектувати структуру веб-застосунку, визначити основні сторінки, компоненти, ролі користувачів та сценарії їх взаємодії із системою;
- розробити клієнтську частину веб-платформи з використанням React, TypeScript та сучасних засобів frontend-розробки;
- реалізувати засоби зберігання даних веб-платформи та інтегрувати рішення для онлайн зберігання файлів браузерних HTML5-ігор;
- реалізувати каталог браузерних ігор із можливістю перегляду, пошуку, фільтрації та запуску ігор безпосередньо у браузері, забезпечити інтеграцію HTML5-ігор, створених за допомогою Construct 2;
- реалізувати взаємодію користувачів із платформою, зокрема додавання ігор до обраного, оцінювання та коментування;
- розробити адміністративну частину для керування іграми, жанрами та користувацьким контентом;
- провести тестування розробленої веб-платформи, перевірити коректність роботи основних функцій, усунути виявлені помилки та розгорнути веб-застосунок на хостинговій платформі.

Методи дослідження включають аналіз предметної області, порівняння існуючих веб-платформ з браузерними іграми, проектування структури веб-застосунку, розробку клієнтської частини засобами React і TypeScript, використання бази даних та хмарних сховищ для зберігання даних і файлів, а також методи функціонального тестування програмного забезпечення.

Практичне значення отриманих результатів полягає у створенні програмного засобу, який забезпечує зручний доступ до браузерних ігор та сприяє популяризації цифрового ігрового контенту. Розроблена платформа може бути використана як основа для впровадження та розвитку сучасних розважальних ігор. Отримані результати підтверджують ефективність використання сучасних вебтехнологій для створення інтерактивних інформаційних систем та мають практичну цінність для сфери веброзробки й цифрових платформ.

РОЗДІЛ 1

АНАЛІЗ МЕТОДІВ СТВОРЕННЯ ВЕБ-ПЛАТФОРМ З БРАУЗЕРНИМИ ІГРАМИ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз сучасного стану проблеми

У сучасних умовах розвитку веб-технологій браузерні ігри залишаються популярним напрямом цифрових розваг завдяки своїй доступності, простоті запуску та відсутності потреби у встановленні додаткового програмного забезпечення. Користувачеві достатньо мати браузер та підключення до мережі Інтернет, щоб отримати доступ до гри. Такий підхід є зручним як для гравців, так і для розробників, оскільки дозволяє швидко поширювати ігровий контент серед широкої аудиторії.

Особливого значення набувають веб-платформи, які об'єднують браузерні ігри в одному середовищі. Такі платформи надають користувачам можливість переглядати каталог ігор, здійснювати пошук, фільтрувати ігри за жанрами, відкривати сторінки окремих ігор та запускати їх безпосередньо у браузері. Завдяки цьому користувач не витрачає час на пошук ігор на різних ресурсах, а отримує централізований доступ до ігрового контенту.

На практиці значна кількість браузерних ігор розміщується на окремих сайтах або у вигляді самостійних HTML5-проектів. Такий підхід має певні недоліки: відсутність єдиної структури каталогу, складність керування іграми, відсутність системи облікових записів користувачів, неможливість додавання ігор до обраного, обмежені можливості для коментування та оцінювання. Крім того, у випадку ручного додавання ігор до файлової структури сайту адміністрування платформи стає незручним і потребує технічного втручання розробника.

У межах аналізу сучасного стану проблеми також було проведено патентний пошук за ключовими словами «browser game platform», «HTML5 game platform», «online game platform», «web-based gaming portal», «платформа браузерних ігор» та «веб-платформа для ігор». У результаті пошуку було

виявлено окремі технічні рішення, пов'язані з онлайн-ігровими платформами, хмарними технологіями та доступом до ігрового контенту через браузер. Водночас ці рішення не повністю відповідають задачам даної роботи, оскільки розроблювана система орієнтована на створення навчальної веб-платформи з каталогом HTML5-ігор, ролями користувачів, коментарями, оцінюванням, обраними іграми та адміністративним керуванням контентом.

Існуючі ігрові веб-платформи, зокрема Poki, CrazyGames, itch.io та Kongregate, надають користувачам доступ до великої кількості браузерних ігор і мають зручні засоби пошуку, категоризації та запуску ігрового контенту. Такі платформи демонструють актуальність централізованого розміщення ігор у веб-середовищі. Водночас більшість із них орієнтована на масову аудиторію, комерційне використання, рекламну монетизацію або публікацію великої кількості стороннього контенту. Для навчального або невеликого проєкту така модель є надмірною, оскільки не дає можливості повністю контролювати структуру системи, ролі користувачів, механізми адміністрування та спосіб інтеграції власних HTML5-ігор.

Окремою проблемою є інтеграція браузерних HTML5-ігор у веб-платформу. На відміну від звичайного медіаконтенту, HTML5-гра складається з набору взаємопов'язаних файлів, серед яких основний файл `index.html`, JavaScript-файли, стилі, зображення, аудіо та інші ресурси. Для коректного запуску гри необхідно забезпечити правильне зберігання цих файлів, доступ до основного файлу запуску та коректну видачу пов'язаних ресурсів браузеру. У разі неправильного зберігання або некоректної видачі файлів можуть виникати проблеми із запуском гри, завантаженням ресурсів або взаємодією користувача з ігровим вмістом.

Важливим аспектом таких платформ є не лише сам факт запуску гри, а й загальний користувацький досвід. Зручна навігація, зрозумілий інтерфейс, візуальне представлення ігор, короткий опис, жанрова класифікація та можливість швидко повернутися до вподобаного контенту значно впливають на якість взаємодії користувача із системою. Через це сучасна ігрова веб-платформа

повинна розглядатися не просто як набір посилань на ігри, а як повноцінне середовище для організації ігрового контенту.

Сучасні веб-технології дають змогу реалізувати таку платформу без розробки складного серверного застосунку з нуля.

Таким чином, сучасний стан проблеми свідчить про актуальність створення веб-платформи для браузерних ігор, яка забезпечує централізований каталог ігор, запуск HTML5-проектів у браузері, авторизацію користувачів, систему ролей, обране, коментарі, оцінювання та адміністративне керування контентом. Розробка такої системи з використанням React і Construct 2 є доцільною, оскільки дозволяє поєднати сучасний веб-інтерфейс із можливістю інтеграції браузерних ігор, створених у спеціалізованому середовищі для розробки ігор.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою кваліфікаційної роботи є розробка веб-платформи з браузерними іграми з використанням бібліотеки React та середовища Construct 2 для створення ігрового контенту. Розроблювана система має забезпечувати централізований доступ до HTML5-ігор, їх запуск у браузері, перегляд інформації про ігри, взаємодію користувачів із платформою та адміністрування ігрового контенту.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- здійснити аналіз сучасного стану розвитку браузерних ігор, проаналізувати існуючі аналоги веб-платформ з браузерними іграми, визначити їхні переваги та недоліки;
- визначити функціональні та нефункціональні вимоги до розроблюваної веб-платформи;
- спроектувати структуру веб-застосунку, визначити основні сторінки, компоненти, ролі користувачів та сценарії їх взаємодії із системою;

- розробити клієнтську частину веб-платформи з використанням React, TypeScript та сучасних засобів frontend-розробки;
- реалізувати засоби зберігання даних веб-платформи та інтегрувати рішення для онлайн зберігання файлів браузерних HTML5-ігор;
- реалізувати каталог браузерних ігор із можливістю перегляду, пошуку, фільтрації та запуску ігор безпосередньо у браузері, забезпечити інтеграцію HTML5-ігор, створених за допомогою Construct 2;
- реалізувати взаємодію користувачів із платформою, зокрема додавання ігор до обраного, оцінювання та коментування;
- розробити адміністративну частину для керування іграми, жанрами та користувацьким контентом;
- провести тестування розробленої веб-платформи, перевірити коректність роботи основних функцій, усунути виявлені помилки та розгорнути веб-застосунок на хостинговій платформі.

Результатом виконання кваліфікаційної роботи має стати функціональна веб-платформа, яка дозволяє користувачам переглядати каталог браузерних ігор, запускати їх у браузері, взаємодіяти з ігровим контентом, а адміністратору керувати наповненням платформи через відповідні засоби адміністрування.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Розробка веб-платформи з браузерними іграми потребує попереднього визначення цілей системи, функцій користувачів, вимог до програмного забезпечення та моделей його основних елементів. На етапі аналізу необхідно встановити, які задачі повинна виконувати система, які категорії користувачів будуть з нею взаємодіяти, які дані потрібно зберігати та які процеси мають бути автоматизовані.

Для виявлення вимог до веб-платформи було використано аналіз предметної області, огляд існуючих аналогів, визначення ролей користувачів і опис основних сценаріїв роботи з системою. Такий підхід дозволяє сформулювати специфікацію вимог до програмного забезпечення та визначити функціональні можливості, необхідні для роботи платформи.

Для моделювання програмної системи обрано UML-нотацію, оскільки вона дає змогу формалізовано представити взаємодію користувачів із системою, структуру даних та логіку роботи основних компонентів. У межах проєктування доцільно використати діаграму варіантів використання для опису сценаріїв взаємодії користувачів із платформою та модель основних сутностей для відображення структури даних системи.

Розроблювана веб-платформа призначена для розміщення, перегляду та запуску браузерних HTML5-ігор у веб-середовищі. Її створення спрямоване на вирішення проблеми розрізненого доступу до браузерних ігор, складності їх централізованого представлення, а також необхідності забезпечення зручної взаємодії користувачів із ігровим контентом.

Ціллю системи є створення веб-платформи, яка забезпечує централізований каталог браузерних ігор, запуск ігор безпосередньо у браузері,

роботу з обліковими записами користувачів, взаємодію з контентом та адміністративне керування іграми.

Основними задачами системи є: організація каталогу браузерних ігор; забезпечення пошуку та фільтрації ігор; реалізація запуску HTML5-ігор у межах сторінки веб-платформи; забезпечення реєстрації та авторизації користувачів; реалізація розмежування прав доступу; зберігання інформації про користувачів, ігри, жанри, коментарі, оцінки та обрані ігри; надання адміністратору засобів керування ігровим контентом.

У розроблюваній веб-платформі передбачено три основні ролі користувачів: гість, авторизований користувач та адміністратор. Кожна роль має власний рівень доступу до функцій системи. Узагальнений розподіл можливостей між ролями наведено в таблиці 2.1.

Таблиця 2.1 – Ролі користувачів веб-платформи

Роль користувача	Основні можливості
Гість	Перегляд каталогу ігор, пошук і фільтрація, перегляд сторінки гри, запуск гри, перегляд коментарів, можливість реєстрації та авторизації.
Авторизований користувач	Усі можливості гостя, додавання ігор до обраного, коментування, оцінювання, редагування профілю, перегляд статистики обраних ігор.
Адміністратор	Усі можливості авторизованого користувача, доступ до адміністративної панелі, додавання, редагування та видалення ігор, керування файлами ігор, жанрами та коментарями.

На основі визначених ролей користувачів сформовано функціональні вимоги до веб-платформи. Вони визначають перелік основних функцій та можливостей системи, необхідних для забезпечення ефективної взаємодії між користувачами та адміністратором. Функціональні вимоги описують дії, які система повинна виконувати під час обробки запитів, керування даними, надання доступу до інформації та підтримки основних бізнес-процесів. Їх формування здійснювалося з урахуванням потреб кожної категорії користувачів, що дозволяє забезпечити зручність використання платформи, безпеку даних та коректне виконання всіх передбачених операцій. Детальний перелік функціональних вимог наведено у таблиці 2.2.

Таблиця 2.2 – Функціональні вимоги проекту

Назва вимоги	Опис вимоги
Перегляд каталогу ігор	Система повинна дозволяти користувачам переглядати каталог браузерних ігор із короткою інформацією про кожну гру.
Пошук ігор	Система повинна дозволяти користувачам здійснювати пошук ігор за назвою.
Фільтрація ігор	Система повинна дозволяти користувачам фільтрувати ігри за жанрами.
Перегляд інформації про гру	Система повинна дозволяти користувачам відкривати сторінку окремої гри та переглядати її назву, опис, жанр, складність, рейтинг та коментарі інших користувачів.
Запуск гри	Система повинна забезпечувати запуск браузерної HTML5-гри безпосередньо на сторінці веб-платформи.
Керування відображенням гри	Система повинна дозволяти користувачам відкривати гру в повноекранному режимі та регулювати звук гри.
Реєстрація користувача	Система повинна дозволяти новим користувачам створювати обліковий запис.
Авторизація користувача	Система повинна дозволяти користувачам входити до облікового запису за допомогою email та пароля.
Розмежування доступу відповідно до ролі	Система повинна надавати доступ до функцій відповідно до ролі користувача: гість, авторизований користувач або адміністратор.
Керування особистим кабінетом	Система повинна дозволяти авторизованому користувачу змінювати ім'я та аватар профілю.
Керування обраними іграми	Система повинна дозволяти авторизованому користувачу додавати ігри до списку обраного та видаляти їх із цього списку.
Коментування ігор	Система повинна дозволяти авторизованому користувачу залишати коментарі до ігор.
Оцінювання ігор	Система повинна дозволяти авторизованому користувачу оцінювати ігри.
Доступ до адміністративної панелі	Система повинна дозволяти доступ до адміністративної панелі лише користувачам із роллю адміністратора.
Керування каталогом ігор	Система повинна дозволяти адміністратору додавати нові ігри із зазначенням назви, опису, складності, початкового рейтингу, кількості гравців та зображення, а також видаляти існуючі ігри з каталогу.
Редагування інформації про гру	Система повинна дозволяти адміністратору змінювати інформацію про існуючу гру.
Керування файлами гри	Система повинна дозволяти адміністратору очищувати або замінювати файли гри.
Керування жанрами	Система повинна дозволяти адміністратору додавати та видаляти жанри ігор.

Нефункціональні вимоги визначають якісні характеристики системи, зокрема вимоги до зручності, продуктивності, стабільності, безпеки та подальшого розширення програмного забезпечення. Перелік нефункціональних вимог наведено в таблиці 2.3.

Таблиця 2.3 – Нефункціональні вимоги

Назва вимоги	Опис вимоги
Адаптивність	Інтерфейс системи повинен коректно відображатися на різних розмірах екранів.
Зручність інтерфейсу	Інтерфейс повинен бути зрозумілим, послідовним і зручним для навігації.
Продуктивність	Сторінки платформи повинні завантажуватися без помітних затримок для користувача.
Стабільність запуску ігор	HTML5-ігри повинні стабільно запускатися у браузері без встановлення додаткового програмного забезпечення.
Безпека доступу	Адміністративні функції повинні бути недоступними для гостей та звичайних користувачів, а доступ до них надається лише адміністратору.
Обробка помилок	Система повинна коректно обробляти помилки авторизації, завантаження даних, запуску гри та роботи з адміністративними функціями.
Віддалене зберігання файлів	Файли ігор повинні зберігатися у віддаленому хмарному сховищі, а не локально у структурі клієнтського застосунку.
Розширюваність	Структура програмного забезпечення повинна дозволяти подальше додавання нових функцій і розширення платформи.
Кросбраузерність	Система повинна коректно працювати в основних сучасних браузерах.

Сформовані функціональні та нефункціональні вимоги охоплюють основні характеристики розроблюваної системи: введення та зміну даних про ігри, жанри й користувацький контент, пошук і фільтрацію ігор за запитом користувача, виведення знайденої інформації в каталозі та на сторінці гри, розмежування прав доступу між гостем, авторизованим користувачем і адміністратором, а також захист адміністративних функцій від несанкціонованого доступу. Цілісність даних забезпечується через логічні зв'язки між користувачами, іграми, жанрами, коментарями, оцінками, обраними іграми та файлами запуску.

Після визначення функціональних і нефункціональних вимог було виконано моделювання взаємодії користувачів із веб-платформою. Для цього використано діаграму варіантів використання (рис. 2.1), яка належить до UML-моделей поведінки та дозволяє відобразити основні сценарії роботи користувачів із системою.

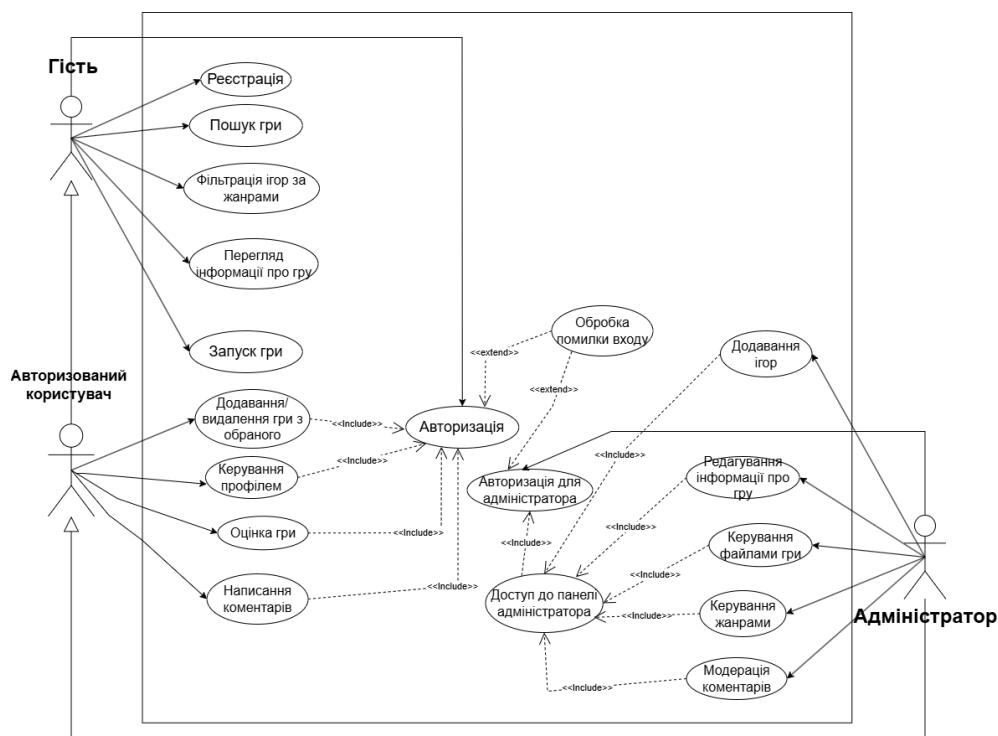


Рисунок 2.1 – Діаграма варіантів використання веб-платформи

На діаграмі варіантів використання зображено трьох основних акторів системи: гостя, авторизованого користувача та адміністратора. Гість має доступ до базових функцій платформи: реєстрації, авторизації, пошуку ігор, фільтрації ігор за жанрами, перегляду інформації про гру та запуску гри. Ці функції не потребують наявності облікового запису, тому доступні будь-якому відвідувачу веб-платформи.

Авторизований користувач має розширені можливості порівняно з гостем. До них належать додавання або видалення гри з обраного, керування профілем, оцінювання гри та написання коментаря. Ці дії пов'язані з персональними даними користувача або його взаємодією з контентом, тому вони виконуються

лише після авторизації в системі. На діаграмі це відображено за допомогою зв'язку “include” між відповідними прецедентами та прецедентом авторизації.

Адміністратор є користувачем із розширеними правами доступу. Для нього передбачено окрему авторизацію адміністратора та доступ до панелі адміністратора. Через адміністративну панель він може додавати ігри, редагувати інформацію про ігри, керувати файлами гри, керувати жанрами та модерувати коментарі користувачів.

Отримана діаграма показує розмежування прав доступу, залежність окремих дій від авторизації та логіку взаємодії різних ролей із системою. Також необхідно описати основні елементи системи, з якими працює веб-платформа, та зв'язки між ними.

З цією метою було побудовано діаграму класів основних сутностей веб-платформи. Використання UML-діаграми [6] класів є доцільним, оскільки вона дозволяє формалізовано представити логічну структуру програмного додатку, визначити основні об'єкти системи, їхні атрибути та взаємозв'язки. Діаграма класів основних сутностей веб-платформи GamletLand наведена на рисунку 2.2.

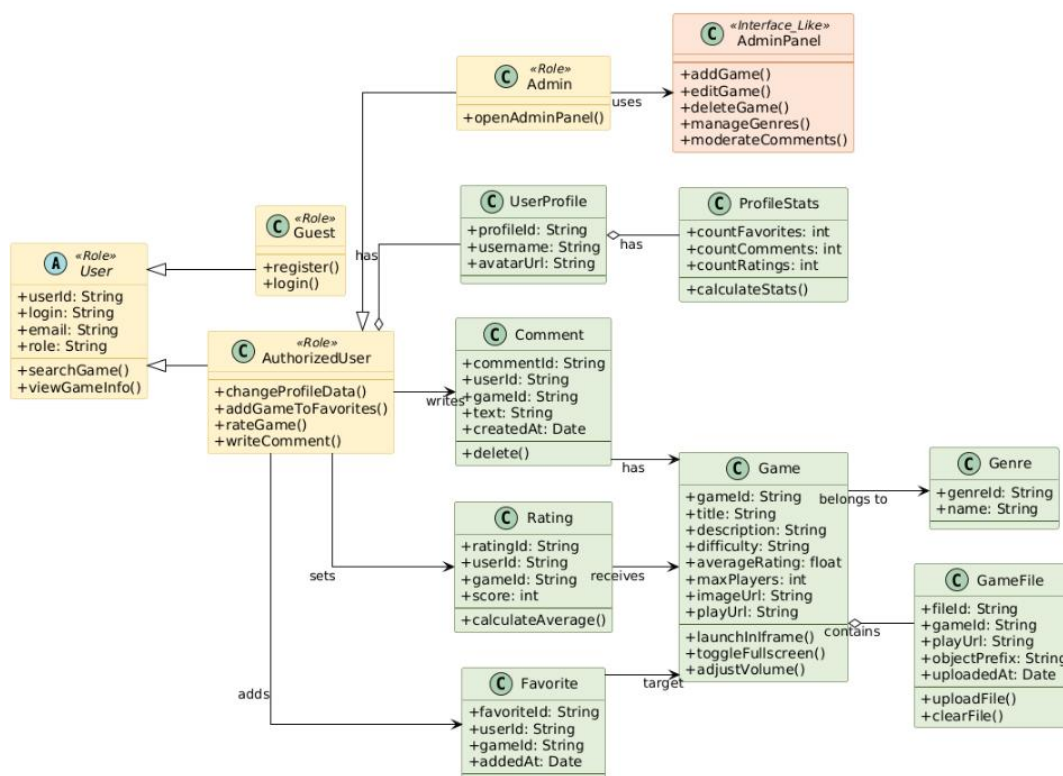


Рисунок 2.2 – Діаграма класів веб-платформи

На діаграмі показано базовий клас User, який описує загальні властивості користувача системи, зокрема ідентифікатор, логін, електронну пошту та роль. Від нього наслідуються класи Guest і AuthorizedUser, що дозволяє розділити можливості неавторизованого та авторизованого користувача. Клас Guest відображає користувача, який може переглядати інформацію про ігри, здійснювати пошук, реєструватися та виконувати вхід у систему. Клас AuthorizedUser розширює ці можливості та додає функції взаємодії з контентом: зміну даних профілю, додавання гри до обраного, оцінювання гри та написання коментаря.

Окремо на діаграмі виділено клас Admin, який є різновидом авторизованого користувача з розширеними правами доступу. Він використовує клас AdminPanel, що містить основні адміністративні функції: додавання, редагування та видалення ігор, керування жанрами й модерацію коментарів. Таке представлення дозволяє показати, що адміністратор має не лише можливості звичайного авторизованого користувача, а й доступ до окремого набору функцій керування платформою.

Клас UserProfile описує профіль користувача та містить інформацію про ім'я користувача й аватар. З ним пов'язаний клас ProfileStats, який використовується для відображення статистики активності користувача: кількості обраних ігор, коментарів та оцінок. Це відповідає функціональним вимогам до особистого кабінету користувача.

Центральною сутністю предметної області є клас Game, який описує гру на платформі. Він містить основні дані про гру: назву, опис, складність, середній рейтинг, максимальну кількість гравців, зображення та посилання для запуску. Методи класу Game відображають дії, пов'язані із запуском HTML5-гри в браузері, переходом у повноекранний режим і регулюванням звуку.

Клас Genre використовується для класифікації ігор за жанрами. Зв'язок між Game і Genre показує, що гра належить до певного жанру, що забезпечує можливість фільтрації ігор у каталозі платформи.

Взаємодія користувача з ігровим контентом представлена класами Comment, Rating і Favorite. Клас Comment зберігає коментарі до гри, клас Rating відповідає за оцінювання гри користувачем, а клас Favorite описує додавання гри до списку обраного. Через ці класи реалізуються основні функції персоналізованої взаємодії авторизованого користувача з платформою.

Клас GameFile описує файли гри, необхідні для її запуску в браузері. Він містить інформацію про ідентифікатор файлу, ідентифікатор гри, шлях до файлу та дату завантаження. Наявність цього класу є важливою, оскільки браузерні ігри, експортовані з Construct 2, складаються з HTML5-файлів, які мають бути пов'язані з відповідною грою на платформі.

Отримана діаграма класів дозволяє проаналізувати структуру основних елементів веб-платформи та пояснити їхню роль у реалізації функціональних вимог. Вона показує, що система складається з користувацьких ролей, ігрового контенту, профілю користувача, механізмів оцінювання, коментування, додавання до обраного та адміністративного керування.

Після визначення основних функцій системи та побудови UML-моделей доцільно описати структуру користувацького інтерфейсу веб-платформи. UX/UI-дизайн системи повинен забезпечувати зрозумілу навігацію, швидкий доступ до каталогу ігор, зручний запуск гри та просте керування контентом в адміністративній частині. Основні сторінки платформи наведено в таблиці 2.4.

Таблиця 2.4 – Основні сторінки веб-платформи GamletLand

Сторінка	Призначення
Головна сторінка	Представлення веб-платформи та перехід до основних розділів.
Каталог ігор	Перегляд списку браузерних ігор, пошук і фільтрація за жанрами.
Сторінка гри	Перегляд інформації про гру, запуск гри, перегляд коментарів та оцінювання.
Реєстрація та авторизація	Створення облікового запису та вхід користувача в систему.
Особистий кабінет	Керування профілем, перегляд статистики та списку обраних ігор.
Адміністративна панель	Додавання, редагування та видалення ігор, керування жанрами, файлами гри й коментарями.
Сторінка про платформу	Відображення загальної інформації про веб-платформу.

Наведена структура сторінок відображає основні сценарії взаємодії користувача з веб-платформою. Такий поділ інтерфейсу дозволяє розмежувати відкриті функції платформи, персональні можливості авторизованого користувача та адміністративне керування контентом.

Невід’ємною частиною проектування є аналіз інформаційних потоків веб-платформи, які пов’язані з передаванням даних між користувачем, інтерфейсом системи, базою даних та сховищем файлів ігор. Користувач через інтерфейс надсилає запити на перегляд каталогу, пошук або фільтрацію ігор, відкриття сторінки гри, авторизацію, додавання коментаря, оцінювання або додавання гри до обраного. Система обробляє ці запити, отримує або змінює відповідні дані та повертає результат у вигляді оновленого інтерфейсу.

Для адміністратора інформаційний потік передбачає введення або зміну даних про гру, керування жанрами, коментарями та файлами гри. Дані про користувачів, ігри, жанри, коментарі, оцінки та обрані ігри зберігаються в базі даних, а файли браузерних HTML5-ігор – у віддаленому сховищі. Детальні особливості технічної реалізації цих потоків доцільно розглядати в наступному розділі під час опису розробки веб-платформи.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для реалізації веб-платформи з браузерними іграми необхідно обрати засоби розробки, які забезпечать створення клієнтської частини, роботу з даними, авторизацію користувачів, зберігання файлів ігор, розгортання веб-застосунку та інтеграцію HTML5-ігор. Оскільки розроблювана система повинна працювати у браузері, мати інтерактивний інтерфейс, каталог ігор, особистий кабінет користувача та адміністративну панель, особливу увагу потрібно приділити вибору frontend-технологій, засобів роботи з даними та інструментів розгортання.

Під час вибору засобів розробки було враховано такі критерії: зручність створення frontend-застосунку з динамічною навігацією між розділами, підтримка компонентного підходу, можливість роботи з типізованими даними, простота інтеграції з базою даних і зовнішніми сервісами, зручність розгортання та підтримка подальшого розширення функціональності платформи.

Розробка веб-платформи здійснюється з використанням ітераційного підходу. Це означає, що система створюється поступово: спочатку реалізуються базові функції, зокрема відображення каталогу ігор і сторінки окремої гри, після чого додаються авторизація, особистий кабінет, взаємодія користувача з ігровим контентом та адміністративні можливості. Такий підхід дозволяє перевіряти працездатність окремих частин платформи на різних етапах розробки, виявляти помилки раніше та поступово розширювати функціональність системи.

Для підтримки цього підходу використовується система контролю версій Git і сервіс GitHub [7]. Git дозволяє фіксувати окремі етапи розробки, відстежувати зміни в коді, працювати з різними версіями проєкту та за потреби повертатися до попереднього стану файлів. GitHub використовується як віддалене сховище репозиторію, що забезпечує централізоване зберігання коду та спрощує подальше розгортання веб-застосунку. Для створення клієнтської частини веб-платформи було розглянуто кілька сучасних frontend-технологій: React, Vue та Angular. Їх порівняння наведено в таблиці 2.5.

Як основне середовище розробки використовується Visual Studio Code. Це середовище забезпечує зручну роботу з файлами React- і TypeScript-проєкту, підтримує підсвічування синтаксису, автодоповнення коду, інтеграцію з терміналом, Git і розширеннями для frontend-розробки. Використання Visual Studio Code дозволяє працювати з компонентами, сторінками, сервісами, файлами стилів, конфігурацією проєкту та SQL-файлами в межах одного робочого середовища. Також Visual Studio Code надає багато розширень для кастомізації та оптимізації процесу розробки програмного забезпечення, що є великим плюсом.

Таблиця 2.5 – Порівняння frontend-технологій

Технологія	Переваги	Недоліки
React	Компонентний підхід, велика кількість бібліотек, гнучкість, зручність створення SPA, активна спільнота	Не є повноцінним фреймворком, тому структуру застосунку, навігацію та окремі архітектурні рішення розробник визначає самостійно
Vue	Простий синтаксис, швидке освоєння, зручна структура компонентів, добра придатність для невеликих і середніх веб-застосунків	Має власний підхід до організації компонентів і екосистеми, тому потребував би адаптації проєкту під інший технологічний стек
Angular	Повноцінний фреймворк із вбудованими засобами маршрутизації, формами та сервісами	Має вищу складність, більший обсяг початкової структури та є надмірним для порівняно невеликої веб-платформи

На основі порівняння для реалізації клієнтської частини веб-платформи було обрано бібліотеку React. Її використання є доцільним, оскільки React підтримує компонентний підхід до побудови інтерфейсу, що добре відповідає структурі розроблюваної платформи. Окремими компонентами можуть бути представлені картки ігор, каталог, сторінка гри, форми авторизації, особистий кабінет користувача та адміністративна панель.

Після вибору React для розробки клієнтської частини використовується TypeScript. Його застосування пояснюється тим, що веб-платформа працює з різними типами даних: інформацією про ігри, користувачів, жанри, коментарі, оцінки та обрані ігри. TypeScript дозволяє описувати структуру таких об'єктів, контролювати типи даних під час розробки та зменшувати кількість помилок у коді. Це особливо важливо для системи, у якій дані передаються між інтерфейсом, базою даних і зовнішніми сервісами.

Для створення структури frontend-проєкту та його збірки використовується Vite [8]. Він забезпечує зручну початкову організацію React-застосунку, швидкий запуск проєкту в режимі розробки та автоматичне оновлення сторінки після внесення змін до коду. Також Vite виконує збірку проєкту, під час якої вихідні файли застосунку обробляються, оптимізуються та перетворюються на набір статичних файлів, придатних для розгортання на хостингу. Завдяки цьому спрощується процес розробки,

Також важливим чинником вибору є практичний досвід роботи з React та особиста зручність використання цієї бібліотеки під час розробки. Це дозволяє ефективніше реалізувати поставлені задачі, швидше налагоджувати інтерфейс і підтримувати структуру проєкту.

Для побудови інтерфейсу також використовується бібліотека MUI, яка надає готові компоненти інтерфейсу: кнопки, поля введення, картки, діалогові вікна, вкладки та інші елементи. Це дозволяє швидше створювати узгоджений і зручний інтерфейс користувача, зберігаючи єдиний стиль сторінок платформи.

Маршрутизація в розроблюваній системі реалізована без використання React Router. Для переходу між основними розділами веб-платформи використовується власна hash-навігація через `window.location.hash`. Такий підхід є достатнім для даного проєкту, оскільки структура сторінок є відносно простою, а основні переходи виконуються між головною сторінкою, каталогом, сторінкою гри, особистим кабінетом та адміністративною панеллю.

Для реалізації роботи з даними, авторизації користувачів і зберігання основної інформації платформи було розглянуто кілька можливих підходів: використання Supabase, Firebase [9] або розробка власної backend-частини на основі Node.js. Порівняння цих засобів наведено в таблиці 2.6.

Таблиця 2.6 - Порівняння засобів реалізації backend-частини

Засіб	Переваги	Недоліки
Supabase	Використовує реляційну базу даних PostgreSQL, підтримує SQL-запити, зв'язки між таблицями, автоматичне створення API, авторизацію користувачів і відкриту модель розробки	Потребує правильного проєктування структури таблиць, налаштування політик доступу та розуміння принципів роботи реляційної бази даних
Firebase	Має готові хмарні сервіси для авторизації, зберігання даних і файлів, швидко інтегрується з frontend-застосунками	Орієнтований переважно на NoSQL-модель даних, що є менш зручним для систем із великою кількістю зв'язків між сутностями
Власний Backend на Node.js	Дає повний контроль над логікою серверної частини, API, безпекою та структурою обробки запитів	Потребує більше часу на розробку, налаштування авторизації, бази даних, API, захисту та розгортання серверної частини

На основі порівняння для реалізації backend-частини було обрано Supabase. Основною перевагою цього засобу є використання реляційної бази даних PostgreSQL, що добре відповідає структурі розроблюваної веб-платформи. У системі необхідно зберігати пов'язані між собою дані про користувачів, ігри, жанри, коментарі, оцінки та обрані ігри. Реляційна модель дозволяє чітко описати ці зв'язки та забезпечити цілісність даних.

На відміну від Firebase, який переважно орієнтований на NoSQL-підхід, Supabase дає змогу працювати з даними у форматі таблиць і використовувати SQL-логіку. Це є зручним для реалізації зв'язків між іграми, жанрами, користувачами, коментарями та оцінками. Крім того, Supabase надає готові засоби авторизації та автоматично створює API для взаємодії frontend-застосунку з базою даних, що дозволяє не розробляти окрему backend-частину з нуля.

У межах розроблюваної платформи Supabase використовується для зберігання даних про користувачів, ролі, профілі, ігри, жанри, коментарі, оцінки та обрані ігри. Також Supabase Storage застосовується для зберігання простих медіафайлів, зокрема аватарів користувачів і зображень-заставок ігор.

Окремою задачею під час розробки веб-платформи є зберігання та запуск файлів браузерних HTML5-ігор. На відміну від зображень аватарів, HTML5-гра складається з набору взаємопов'язаних файлів: `index.html`, JavaScript-файлів, стилів, зображень, аудіо та інших ресурсів. Для коректного запуску такої гри необхідно не лише зберегти файли, а й забезпечити правильну їх видачу браузеру з відповідними шляхами та типами вмісту.

Для зберігання повного набору файлів браузерних ігор було обрано Cloudflare R2. Це об'єктне сховище дозволяє розміщувати файли HTML5-ігор окремо від клієнтської частини веб-застосунку та не перевантажувати структуру React-проєкту. Такий підхід є зручним для подальшого додавання нових ігор і оновлення вже наявних, оскільки файли гри можуть зберігатися у віддаленому сховищі, а в базі даних Supabase достатньо зберігати інформацію про гру та посилання для її запуску.

Для розгортання клієнтської частини веб-платформи використовується сервіс Netlify [10]. Він призначений для розміщення сучасних веб-застосунків і добре підходить для проєктів, створених за допомогою React та Vite. Після виконання збірки застосунку формується набір статичних файлів, які можуть бути розміщені на хостингу та відкриватися користувачами через браузер.

Використання Netlify є доцільним для розроблюваної платформи, оскільки клієнтська частина системи реалізована як frontend-застосунок. Netlify дозволяє швидко розгорнути сайт, забезпечити доступ до нього через мережу Інтернет і спростити процес оновлення після внесення змін до коду. Додатковим чинником вибору є наявний практичний досвід роботи з Netlify, що спрощує налаштування розгортання, перевірку роботи застосунку та подальшу підтримку платформи.

Для створення браузерних ігор у межах роботи використовується Construct 2. Це середовище призначене для розробки 2D-ігор і дозволяє експортувати створені проєкти у формат HTML5. Такий формат є зручним для розроблюваної веб-платформи, оскільки отримані після експорту файли можуть запускатися безпосередньо у браузері без встановлення додаткового програмного забезпечення.

Використання Construct 2 є доцільним, оскільки він дозволяє створювати прості браузерні ігри без необхідності розробляти ігровий рушій з нуля. Це відповідає задачі роботи, де основна увага приділяється не розробці складної гри, а створенню веб-платформи для розміщення, запуску та адміністрування браузерних HTML5-ігор.

Після вибору основних засобів розробки варто описати алгоритми, які використовуються для реалізації основних функцій веб-платформи. До таких алгоритмів належать пошук і фільтрація ігор, запуск HTML5-гри, додавання гри адміністратором та розрахунок середнього рейтингу гри.

Алгоритм пошуку і фільтрації ігор використовується для швидкого відбору ігрового контенту в каталозі. Користувач вводить пошуковий запит або обирає жанр гри. Система отримує поточний список ігор, порівнює назву гри із введеним запитом і перевіряє належність гри до вибраного жанру. Після цього

формується оновлений список ігор, який відображається в каталозі. Якщо ігор, що відповідають заданим умовам, не знайдено, користувачу відображається відповідне повідомлення.

Найбільш комплексним серед описаних алгоритмів є алгоритм додавання гри адміністратором, оскільки він охоплює введення даних, перевірку їх коректності, роботу з файлами гри та подальше відображення нового запису в каталозі. Для наочного представлення цього процесу побудовано блок-схему, наведену на рисунку 2.3.

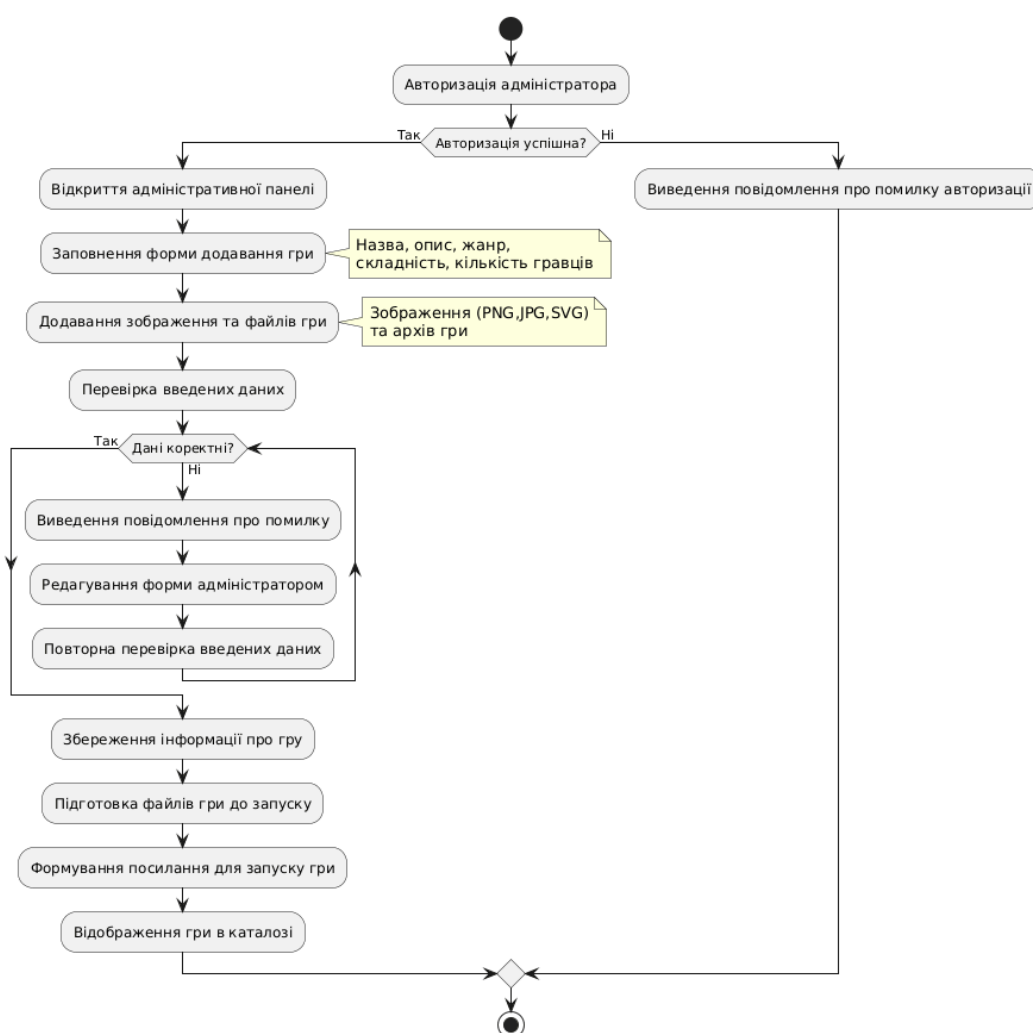


Рисунок 2.3 – Блок-схема алгоритму додавання гри адміністратором

Алгоритм додавання гри адміністратором передбачає послідовне введення інформації про гру, перевірку заповнених даних, збереження описових

характеристик гри та підготовку файлів для подальшого запуску. Адміністратор відкриває адміністративну панель, заповнює форму додавання гри, вказує назву, опис, жанр, складність, кількість гравців, зображення та файли гри. Після перевірки введених даних система зберігає інформацію про гру та робить її доступною в каталозі платформи.

Алгоритм запуску HTML5-гри починається з відкриття користувачем сторінки конкретної гри. Система отримує інформацію про гру, зокрема назву, опис, жанр, рейтинг і посилання для запуску. Якщо посилання на файл запуску доступне, HTML5-гра відкривається у вбудованому вікні на сторінці веб-платформи.

Для оцінювання ігор у веб-платформі використовується математична модель розрахунку середнього рейтингу. Користувачі можуть залишати оцінки для окремих ігор, після чого система визначає середній рейтинг гри на основі всіх отриманих оцінок. Середній рейтинг гри обчислюється за формулою (2.1).

$$R_{avg} = \frac{\sum_{i=1}^n R_i}{n} , \quad (2.1)$$

де R_{avg} – середній рейтинг гри;

R_i – оцінка, яку поставив i -й користувач;

n – загальна кількість оцінок.

Отже, середній рейтинг гри визначається як сума всіх користувацьких оцінок, поділена на їх кількість. Отримане значення використовується для відображення рейтингу гри в каталозі та на сторінці окремої гри. Якщо гра ще не має користувацьких оцінок, для неї може відображатися початкове значення рейтингу або повідомлення про відсутність оцінок.

РОЗДІЛ 3

РОЗРОБКА ВЕБ-ПЛАТФОРМИ З БРАУЗЕРНИМИ ІГРАМИ

3.1 Практична реалізація об'єкта проєктування

Практична реалізація веб-платформи GamletLand розпочалася зі створення репозиторію на сервісі GitHub. Використання репозиторію дозволило зберігати вихідний код проєкту, фіксувати окремі етапи розробки, відстежувати зміни у файлах та за потреби повертатися до попередніх версій. Це є важливим під час розробки веб-платформи, оскільки система складається з кількох функціональних частин: каталогу ігор, сторінки окремої гри, системи авторизації, особистого кабінету користувача, адміністративної панелі та механізму запуску HTML5-ігор.

Після створення репозиторію було сформовано початкову структуру frontend-застосунку за допомогою інструмента Vite. Для цього було створено React-проєкт із підтримкою TypeScript. Такий підхід дозволив одразу отримати готову основу для розробки сучасного веб-застосунку, у якій уже передбачено структуру файлів, налаштування збірки та можливість запуску проєкту в локальному середовищі.

Після запуску локального середовища розробки застосунків відкривався у браузері, що дозволяло одразу перевіряти зміни в інтерфейсі, компонентах і логіці роботи сторінок. Це спростило процес поступової реалізації функцій платформи, оскільки кожен змін можна було швидко перевірити без повної повторної збірки проєкту.

У процесі розробки базова структура React/Vite-проєкту була розширена відповідно до функціональних вимог веб-платформи. До проєкту були додані окремі папки для компонентів, сторінок, сервісів, типів даних, початкових даних і SQL-файлів, які використовувалися для налаштування бази даних Supabase.

Оскільки основні файли вихідного коду frontend-застосунку розміщені в папці `src` (рис.3.1), для демонстрації структури проєкту доцільно навести саме її вміст.

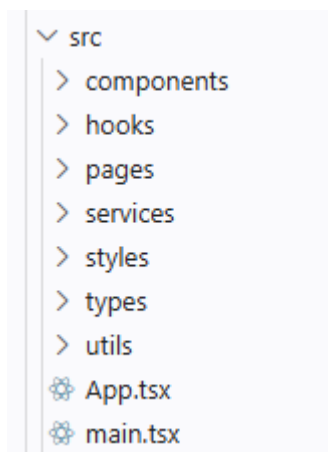


Рисунок 3.1 – Організація файлів веб-платформи

Структура проєкту відображає поділ вихідного коду за функціональним призначенням. У папці `components` розміщено повторно використовувані компоненти інтерфейсу, зокрема загальний макет застосунку та картку гри. У папці `hooks` розміщено окремі файли з логікою проєкту. Папка `pages` містить основні сторінки веб-платформи: головну сторінку, сторінку гри, сторінки авторизації та реєстрації, профіль користувача, сторінку про платформу й адміністративну панель.

Окрема папка `pages/admin` використовується для компонентів адміністративної частини. У ній розміщено файли, що відповідають за керування іграми, жанрами, коментарями, навігацією адміністративної панелі та відображенням її заголовка. Такий поділ дозволяє не змішувати логіку звичайних сторінок користувача з функціями адміністратора.

У папці `services` розміщено файли, які відповідають за взаємодію із зовнішніми сервісами. Зокрема, файл `supabaseClient.ts` використовується для підключення до Supabase, а `gameUploadWorker.ts` – для взаємодії з Cloudflare Worker під час завантаження файлів браузерної гри. Папка `types` містить описи типів даних TypeScript, які використовуються під час роботи з іграми, користувачами, коментарями, оцінками та іншими сутностями системи.

Основними файлами проєкту є `main.tsx`, `App.tsx` і `theme.ts`. Файл `main.tsx` є точкою входу React-застосунку та відповідає за підключення основного компонента до HTML-сторінки. Файл `App.tsx` координує відображення сторінок

платформи та загальну логіку роботи застосунку. Файл `theme.ts` використовується для налаштування візуального стилю інтерфейсу.

Наступним етапом опису практичної реалізації є створення візуальної частини веб-платформи. Перед реалізацією інтерфейсу було підготовлено елементи візуальної ідентичності проєкту. Для цього використовувався графічний редактор Figma [11], у якому було створено логотип платформи GamletLand та визначено основну кольорову палітру сайту. Логотип і кольорова палітра веб-платформи наведені на рисунку 3.2.

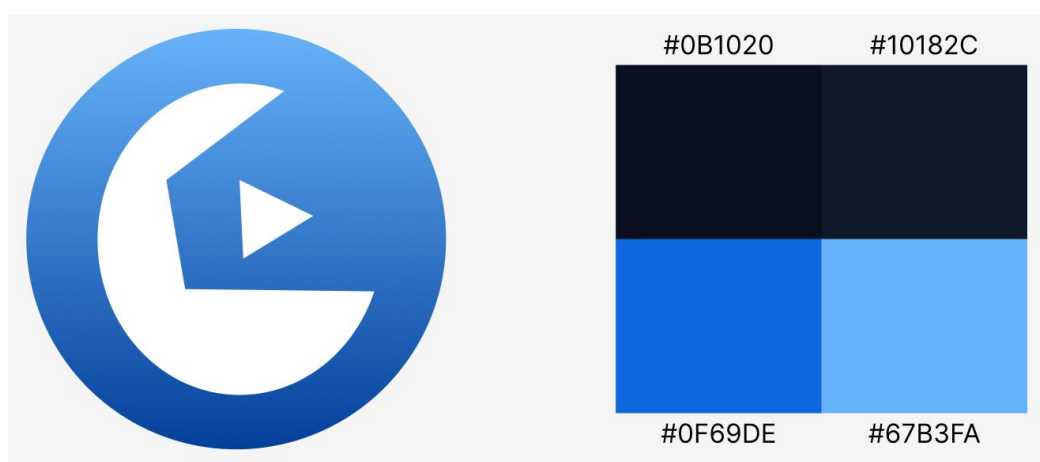


Рисунок 3.2 – Компоненти візуальної ідентичності платформи

У палітрі використано темні кольори `#0B1020` та `#10182C`, які застосовуються для основного фону сторінок і окремих блоків інтерфейсу. Колір `#0F69DE` використовується для кнопок, активних елементів і важливих акцентів. Світліший синій колір `#67B3FA` застосовується для додаткового виділення елементів і декоративного оформлення. Таке поєднання кольорів відповідає тематиці ігрової веб-платформи та підтримує сучасний вигляд інтерфейсу.

Після визначення кольорової палітри її було перенесено в код застосунку через файл `theme.ts`. У цьому файлі налаштовано основні кольори інтерфейсу, параметри типографіки, стилі кнопок, карток, полів введення та інших елементів. Використання спільної теми дозволило зберегти однакове оформлення сторінок і забезпечити візуальну цілісність веб-платформи. Крім того, централізоване

налаштування стилів спростило подальше редагування дизайну, оскільки зміни основних кольорів або параметрів інтерфейсу можна виконувати в одному файлі.

Для побудови інтерфейсу було використано бібліотеку Material UI [12]. Вона надала готові компоненти для створення кнопок, полів введення, карток, вкладок, діалогових вікон, повідомлень, аватарів та інших елементів. Компоненти Material UI працюють разом із налаштованою темою, тому кнопки, поля, картки та інші елементи інтерфейсу мають узгоджений вигляд у різних частинах веб-платформи. Завдяки цьому процес створення сторінок став швидшим, а основну увагу можна було зосередити на логіці роботи платформи: навігації, пошуку, фільтрації, авторизації, профілі користувача та адміністративному керуванні. Яскравим прикладом застосування бібліотеки є реалізація хедеру сайту (ліст. 3.1). Для його побудови було використано компоненти AppBar, Container, Toolbar і Button.

Лістинг 3.1 – Реалізація візуальних компонентів за допомогою Material UI

```
import { AppBar, Button, Container, Toolbar } from '@mui/material'

<AppBar position="sticky" elevation={0}>
  <Container maxWidth="lg">
    <Toolbar sx={{ justifyContent: 'space-between' }}>
      <Button onClick={onHomeClick}>GamletLand</Button>
      <Button variant="contained" onClick={onLoginClick}>
        Увійти
      </Button>
    </Toolbar>
  </Container>
</AppBar>
```

Кінець лістингу 3.1

У наведеному фрагменті коду компонент AppBar використовується для створення верхньої панелі сайту, Container обмежує ширину її вмісту, а Toolbar розміщує елементи шапки в один ряд. Властивість sx дає змогу задавати стилі без створення окремого CSS-файлу. Кнопка з назвою GamletLand виконує перехід на головну сторінку, а кнопка Увійти відкриває сторінку авторизації.

У результаті було створено навігаційну шапку, яка відображається у верхній частині веб-платформи та забезпечує швидкий доступ до основних розділів сайту. Приклад реалізованої навігаційної шапки наведено на рисунку 3.3.



Рисунок 3.3 – Навігаційна шапка веб-платформи GamletLand

У ній розміщено логотип і назву GamletLand, а також основні елементи навігації. Для неавторизованого користувача в шапці доступні переходи до каталогу ігор, сторінки «Про нас», форми входу та сторінки реєстрації. Завдяки цьому користувач одразу бачить основні можливості платформи та може швидко перейти до потрібного розділу.

Для переходу між основними сторінками веб-платформи було реалізовано власну hash-навігацію. Такий підхід дозволив не підключати окрему бібліотеку маршрутизації, оскільки структура застосунку є відносно простою. Поточна сторінка визначається на основі значення `window.location.hash`, а після вибору пункту меню активна сторінка оновлюється у стані React. Фрагмент реалізації маршрутизації наведено в лістингу 3.2.

Лістинг 3.2 – Реалізація hash-навігації у веб-платформі

```

type AppPage = 'home' | 'register' | 'login' | 'admin' | 'about' |
'profile' | 'game'

const pageFromHash = (hash: string): AppPage => {
  const cleanHash = hash.replace(/^#/, '')
  if (cleanHash.startsWith('game/')) return 'game'
  if (['register', 'login', 'admin', 'about',
'profile'].includes(cleanHash)) {
    return cleanHash as AppPage
  }
  return 'home'
}

const [activePage, setActivePage] = useState<AppPage>(<

```

```

    () => pageFromHash(window.location.hash)
  )

const openPage = (page: AppPage) => {
  setActivePage(page)
  window.location.hash = page === 'home' ? 'home' : page
}

```

Кінець лістингу 3.2

У цьому фрагменті тип `AppPage` визначає перелік сторінок, які можуть відображатися у веб-платформі. Функція `pageFromHash` отримує поточний `hash` з адресного рядка браузера, очищує його від символу `#` і визначає, яку сторінку потрібно показати. Якщо адреса починається з `game/`, система відкриває сторінку окремої гри. Якщо значення `hash` відповідає одній із службових сторінок, зокрема `register`, `login`, `admin`, `about` або `profile`, воно використовується як активна сторінка. В усіх інших випадках користувачу відображається головна сторінка.

Стан `activePage` використовується для збереження поточної сторінки у React-застосунку. Функція `openPage` змінює активну сторінку та оновлює `window.location.hash`, завдяки чому користувач може переходити між розділами платформи без перезавантаження сторінки. Такий спосіб маршрутизації забезпечує просту навігацію між основними частинами веб-платформи та відповідає структурі розробленого застосунку.

Після реалізації навігаційної частини було створено блок пошуку та фільтрації ігор. Він розміщується на головній сторінці після основного інформаційного заголовка платформи та призначений для швидкого відбору ігор у каталозі. Вигляд реалізації цього блоку наведено на рисунку 3.4.

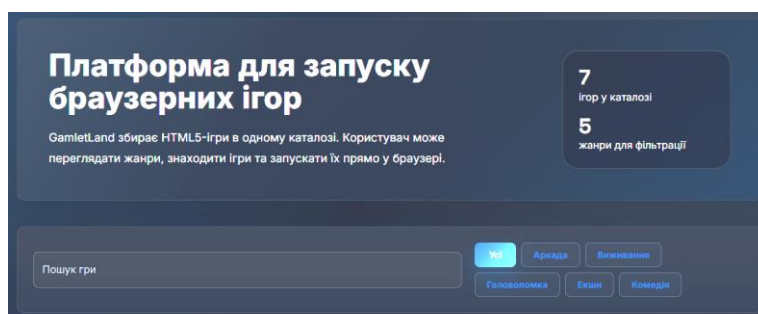


Рисунок 3.4 – Блок пошуку та фільтрації ігор

Блок містить поле введення пошукового запиту та кнопки жанрів. Поле пошуку дозволяє користувачу знайти гру за назвою, а кнопки жанрів використовуються для фільтрації ігор за категоріями. За замовчуванням активним є фільтр «Усі», тому користувачу відображається повний список доступних ігор.

Логіка пошуку та фільтрації ігор реалізована на стороні frontend-застосунку. Для цього використовуються два стани: `searchText` зберігає текст, введений користувачем у поле пошуку, а `selectedGenre` зберігає вибраний жанр. За замовчуванням активним є значення `Усі`, тому без вибору конкретної категорії користувачу відображаються всі доступні ігри.

Формування відфільтрованого списку виконується за допомогою `useMemo`. Це дозволяє перераховувати масив `filteredGames` лише тоді, коли змінюється список ігор, пошуковий запит або вибраний жанр. Під час фільтрації пошуковий текст очищується від зайвих пробілів і переводиться в нижній регістр, після чого для кожної гри перевіряється відповідність жанру та наявність введеного тексту в назві. Фрагмент реалізації пошуку та фільтрації ігор наведено в лістингу 3.3.

Лістинг 3.3 – Реалізація логіки пошуку та фільтрації ігор

```
const [searchText, setSearchText] = useState('')
const [selectedGenre, setSelectedGenre] = useState('Усі')

const filteredGames = useMemo(() => {
  const normalizedSearch = searchText.trim().toLowerCase()

  return games.filter((game) => {
    const fitsGenre = selectedGenre === 'Усі' || game.genre ===
selectedGenre
    const fitsSearch =
game.title.toLowerCase().includes(normalizedSearch)

    return fitsGenre && fitsSearch
  })
}, [games, searchText, selectedGenre])
```

Кінець лістингу 3.3

Головним компонентом сторінки є каталог ігор. Ігри подаються у вигляді окремих карток, що містять обкладинку, назву, жанр, рівень складності, рейтинг, короткий опис, кількість гравців, кнопку відкриття сторінки гри та кнопку додавання до обраного. Приклад відображення каталогу ігор наведено на рисунку 3.5.

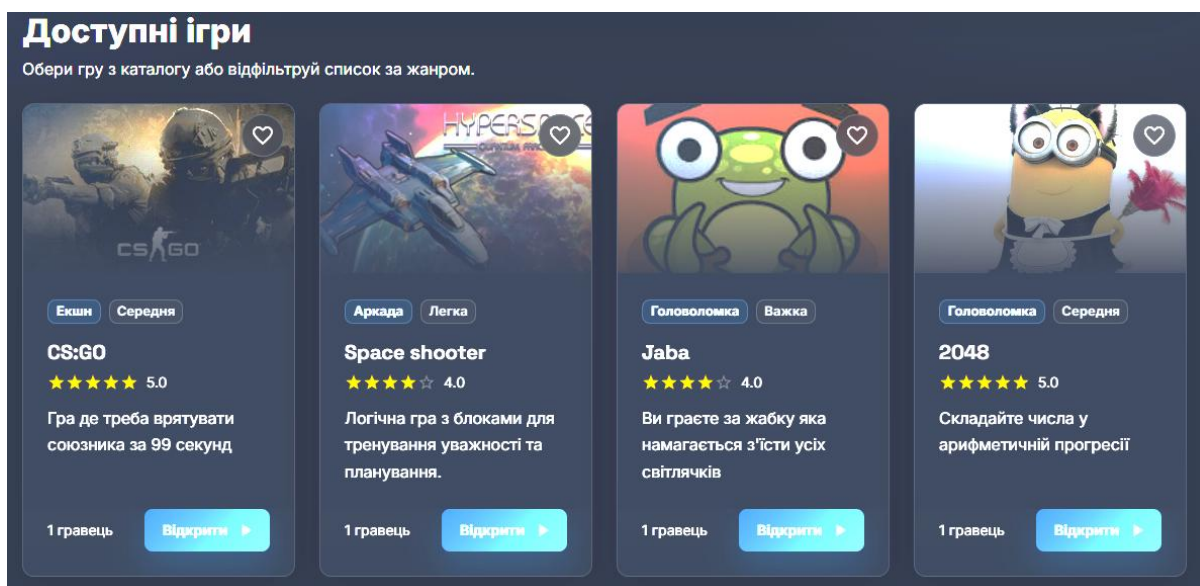


Рисунок 3.5 – Каталог ігор

Каталог ігор формується на основі масиву `filteredGames`, який містить результат роботи пошуку та фільтрації. Для кожного елемента цього масиву створюється компонент `GameCard`. У нього передаються дані про гру, інформація про те, чи додана гра до обраного, а також функції для відкриття сторінки гри та зміни стану обраного. Відображення карток реалізовано за допомогою адаптивної CSS-сітки, тому кількість карток в одному рядку автоматично змінюється залежно від ширини екрана. Фрагмент реалізації каталогу ігор наведено в лістингу 3.4.

Лістинг 3.4 – Формування каталогу ігор у вигляді адаптивної сітки

```
<Box
  sx={{
    display: 'grid',
    gridTemplateColumns: 'repeat(auto-fit, minmax(260px, 1fr))',
```

```

    gap: { xs: 2.5, md: 3 },
  }}
>
{filteredGames.map((game) => (
  <GameCard
    key={game.id}
    game={game}
    isFavorite={favoriteIds.includes(game.id)}
    onOpen={onOpenGame}
    onToggleFavorite={onToggleFavorite}
  />
))}
</Box>

```

Кінець лістингу 3.4

Після вибору гри в каталозі користувач переходить на її сторінку, де може запустити гру безпосередньо у браузері. Основним елементом цієї сторінки є область відображення HTML5-гри. Вона вбудовується у веб-платформу за допомогою елемента `iframe` (рис. 3.6).

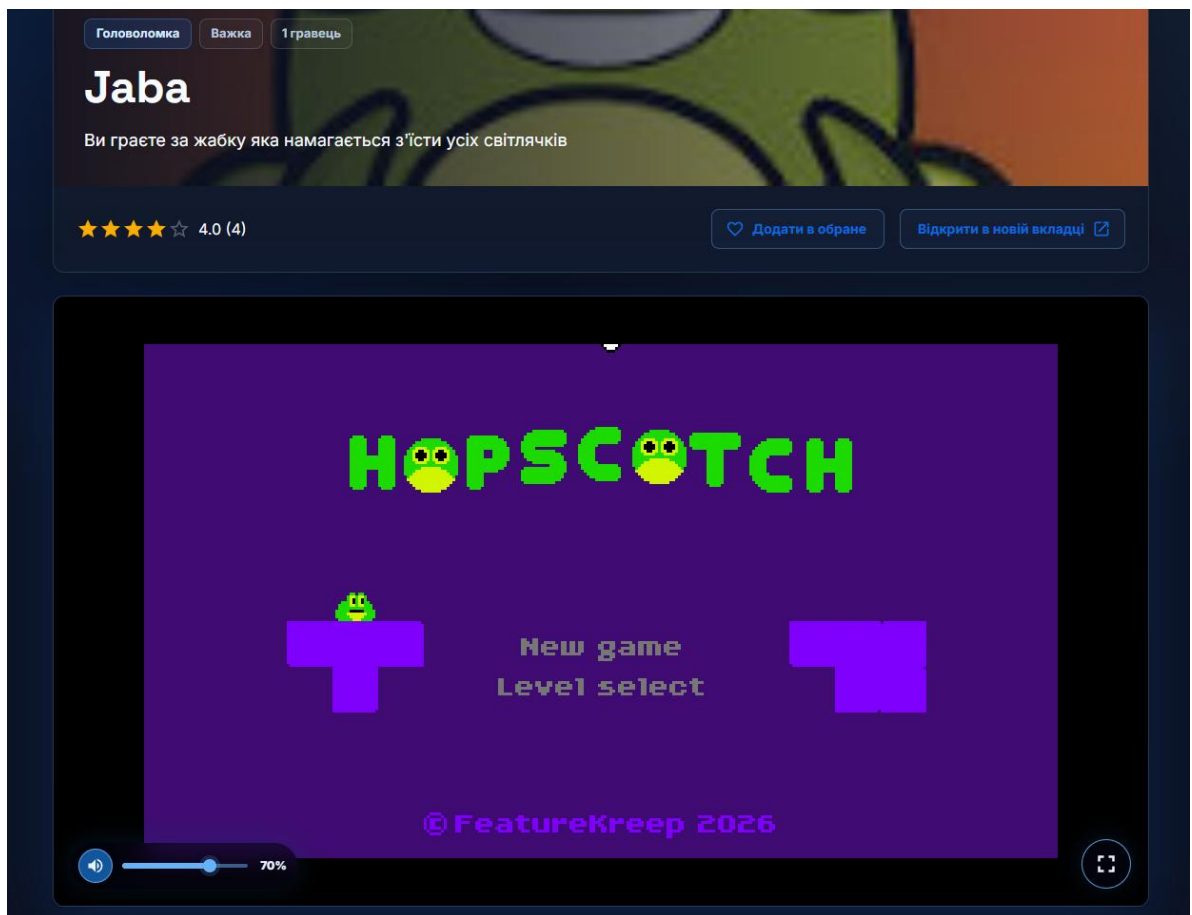


Рисунок 3.6 – Відображення гри за допомогою Iframe

На сторінці гри також передбачено елементи керування. Зокрема, користувач може розгорнути гру на більшу область екрана за допомогою кнопки повноекранного режиму. Якщо гра ще не була завантажена або для неї відсутнє посилання запуску, замість області гри відображається відповідне повідомлення.

Логіка відображення гри реалізована через перевірку наявності посилання `playUrl`. Якщо таке посилання існує, воно передається в атрибут `src` елемента `iframe`, після чого HTML5-гра завантажується всередині сторінки. Для `iframe` також задано дозволи на автозапуск, повноекранний режим, використання геймпада та захоплення вказівника миші. Фрагмент реалізації запуску гри наведено в лістингу 3.5.

Лістинг 3.5 – Відображення HTML5-гри на сторінці гри

```
<Box
  component="iframe"
  ref={gameFrameRef}
  title={title}
  src={playUrl}
  allow="autoplay; fullscreen; gamepad; pointer-lock"
  allowFullScreen
  onLoad={onFrameLoad}
  sx={{ width: '100%', height: { xs: 420, md: 640 }, border: 0 }}
/>
```

Кінець лістингу 3.5

Окремо у веб-платформі реалізовано інформаційну сторінку «Про нас». Вона містить короткий опис призначення GamletLand, пояснює основну ідею платформи та знайомить користувача з її можливостями. Ця сторінка доповнює структуру сайту та робить платформу зрозумілішою для нового користувача.

Базові можливості веб-платформи, зокрема перегляд каталогу, пошук і запуск браузерних ігор, доступні неавторизованому користувачу. Водночас функції взаємодії з платформою, такі як додавання ігор до обраного, виставлення оцінок і написання коментарів, потребують наявності облікового запису. Крім того, доступ до адміністративної панелі залежить від ролі користувача. Тому

наступним елементом практичної реалізації стала система реєстрації та авторизації.

Для реалізації реєстрації та авторизації користувачів було використано сервіс Supabase. Він забезпечує створення облікових записів, вхід за електронною поштою та паролем, збереження поточної сесії користувача й подальшу роботу з даними платформи.

Підключення frontend-застосунку до Supabase винесено в окремий файл `supabaseClient.ts`. Для цього використовуються змінні середовища, які зберігаються у файлі `.env`. У них вказано адресу Supabase-проєкту та публічний ключ доступу. Такий підхід дозволяє не прописувати службові значення безпосередньо в коді застосунку. Перед створенням клієнта виконується перевірка наявності цих змінних, що допомагає швидше виявити помилку конфігурації під час запуску проєкту. Реалізацію підключення до Supabase наведено в лістингу 3.6.

Лістинг 3.6 – Підключення frontend-застосунку до Supabase

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL
const supabaseKey = import.meta.env.VITE_SUPABASE_PUBLISHABLE_KEY

if (!supabaseUrl || !supabaseKey) {
  throw new Error('Supabase URL or publishable key is missing in .env')
}

export const supabase = createClient(supabaseUrl, supabaseKey)
```

Кінець лістингу 3.6

Реєстрація користувача реалізована через форму, у якій користувач вводить ім'я, електронну пошту та пароль. Після надсилання форми дані передаються до Supabase Auth, де створюється новий обліковий запис. Якщо реєстрація проходить успішно, користувач отримує повідомлення про необхідність підтвердження електронної пошти. Форма реєстрації зображено на рисунку 3.7.

Рисунок 3.7 – Форма реєстрації

Реєстрація користувача реалізована через метод `supabase.auth.signUp`. Під час створення облікового запису до Supabase передаються електронна пошта та пароль, введені користувачем у формі. Додатково через параметр `options.data` зберігаються ім'я користувача та початкова роль `user`. Це дозволяє одразу після реєстрації пов'язати обліковий запис із базовою роллю звичайного користувача.

Після виконання запиту перевіряється наявність помилки. Якщо Supabase повертає помилку, її текст відображається користувачу. Якщо обліковий запис створено, але активна сесія ще не сформована, система виводить повідомлення про необхідність підтвердження електронної пошти. У разі успішної реєстрації та створення сесії викликається функція `onRegisterSuccess`, яка завершує процес реєстрації в інтерфейсі застосунку. Фрагмент реалізації реєстрації користувача наведено в лістингу 3.7.

Лістинг 3.7 – Реалізація реєстрації користувача через Supabase Auth

```
const { data, error } = await supabase.auth.signUp({
  email: form.email,
  password: form.password,
  options: {
    data: {
      full_name: form.name,
      role: 'user',
    },
  },
})
```

```

if (error) {
  setMessage(error.message)
  return
}

if (!data.session) {
  setMessage('Акаунт створено. Перевір email для підтвердження
реєстрації.')
  return
}

onRegisterSuccess()

```

Кінець лістингу 3.7

Після успішного створення облікового запису користувачу відображається повідомлення про необхідність підтвердження електронної пошти. Це дозволяє завершити реєстрацію та надалі використовувати акаунт для входу на платформу.

Авторизація користувача реалізована за схожим принципом, на сторінці входу користувач вводить електронну пошту та пароль, після чого ці дані передаються до Supabase Auth для перевірки. Якщо введені дані правильні, Supabase створює активну сесію, а frontend-застосунок отримує інформацію про поточного користувача.

Після авторизації користувачу стають доступні додаткові функції взаємодії з іграми: додавання до обраного, виставлення оцінок і написання коментарів. Дані, які створюються під час виконання цих дій, зберігаються в базі даних Supabase. Для цього використовуються таблиці favorites, ratings і comments, у яких записи пов'язуються з користувачем та грою.

Однією з функцій, доступних авторизованому користувачу, є додавання гри до обраного. Для цього на картці гри є кнопка у формі сердечка та кнопка на сторінці гри. Логіка функції працює як перемикач: якщо гра вже додана до обраного, вона вилучається зі списку, а якщо її ще немає – додається. Зміни зберігаються в Supabase та одразу відображаються в інтерфейсі без

перезавантаження сторінки. Фрагмент реалізації цієї функції наведено в лістингу 3.8.

Лістинг 3.8 – Реалізація додавання гри до обраного

```
const toggleFavorite = async (gameId: number) => {
  if (!userId) return false

  if (favoriteIds.includes(gameId)) {
    await supabase
      .from('favorites')
      .delete()
      .eq('user_id', userId)
      .eq('game_id', gameId)

    setFavoriteIds((ids) => ids.filter((id) => id !== gameId))
    return true
  }

  await supabase.from('favorites').upsert({
    user_id: userId,
    game_id: gameId,
  })

  setFavoriteIds((ids) => [...ids, gameId])
  return true
}
```

Кінець лістингу 3.8

Ще однією функцією авторизованого користувача є оцінювання гри. Для візуального відображення цієї можливості на сторінці гри використано компонент оцінювання у вигляді зірок (рис. 3.8).

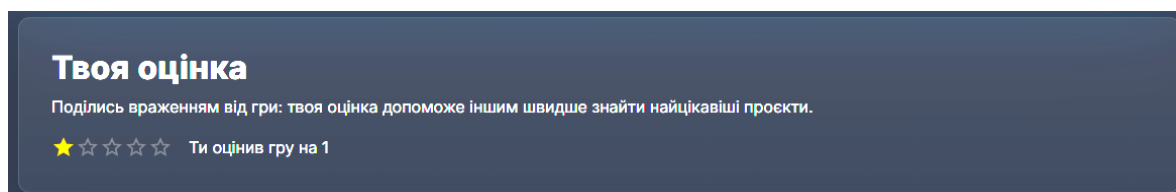


Рисунок 3.8 – Блок оцінювання гри користувачем

Логіка збереження оцінки реалізована у функції `saveRating`. Перед виконанням запиту перевіряється, чи користувач авторизований і чи було

вибрано значення оцінки. Якщо ці умови виконані, у Supabase передаються ідентифікатор гри, ідентифікатор користувача та вибране значення. Для збереження використовується метод `upsert`, тому повторне оцінювання тієї самої гри не створює дублікату, а оновлює попередню оцінку користувача. Після збереження викликається функція `reloadRatings`, яка оновлює рейтинг на сторінці гри. Фрагмент реалізації збереження оцінки наведено в лістингу 3.9.

Лістинг 3.9 – Реалізація збереження оцінки гри

```
const saveRating = async (value: number | null) => {
  if (!userId || !value) return

  await supabase.from('ratings').upsert(
    {
      game_id: game.id,
      user_id: userId,
      value,
    },
    { onConflict: 'game_id,user_id' },
  )

  await reloadRatings()
}
```

Кінець лістингу 3.9

Останнім інтерактивним елементом для користувача є можливість залишити коментар про гру, який буде відображуватись на сторінці гри під оцінкою (рис. 3.9).

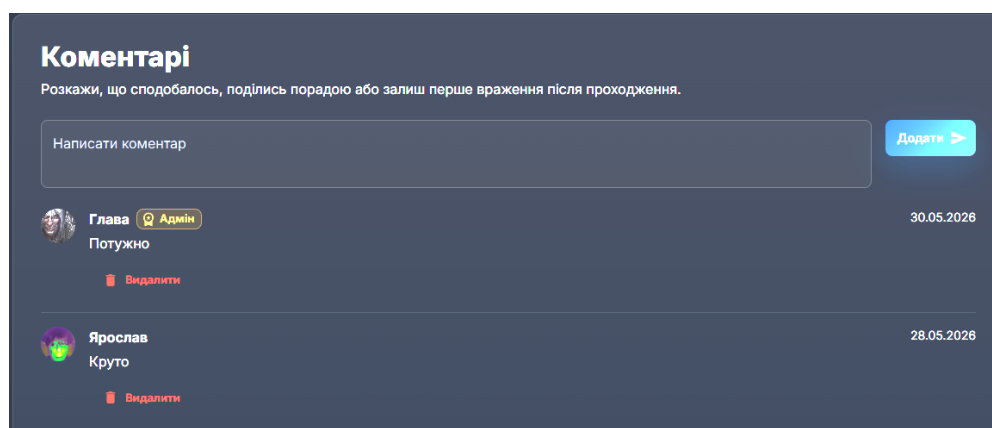


Рисунок 3.9 – Блок коментарів на сторінці гри

Логіка додавання коментаря реалізована у функції `addComment`. Перед збереженням текст коментаря очищується від зайвих пробілів за допомогою `trim()`. Якщо користувач не авторизований або поле коментаря порожнє, функція не виконує запит до бази даних. Якщо дані коректні, у Supabase передаються ідентифікатор гри, ідентифікатор користувача, ім'я автора, його роль, аватар і текст коментаря. Після успішного додавання поле введення очищується, а список коментарів оновлюється через функцію `reloadComments`. Фрагмент реалізації додавання коментаря наведено в лістингу 3.10.

Лістинг 3.10 – Реалізація додавання коментаря до гри

```
const addComment = async () => {
  const trimmedText = commentText.trim()

  if (!userId || !trimmedText) return

  await supabase.from('comments').insert({
    game_id: game.id,
    user_id: userId,
    author_name: userName,
    author_role: userRole === 'admin' ? 'admin' : 'user',
    author_avatar_url: avatarUrl,
    text: trimmedText,
  })

  setCommentText('')
  await reloadComments()
}
```

Кінець лістингу 3.10

Додавання до обраного, оцінювання та коментування формують активність користувача на платформі. Надалі ці дані використовуються в особистому кабінеті (рис. 3.10) для відображення статистики та списку обраних ігор. Крім цього, у профілі користувач може змінити своє ім'я та оновити аватар.

Користувач має зручний доступ до обраних ігор відразу у своєму кабінеті, що спрощує навігацію по сайту.

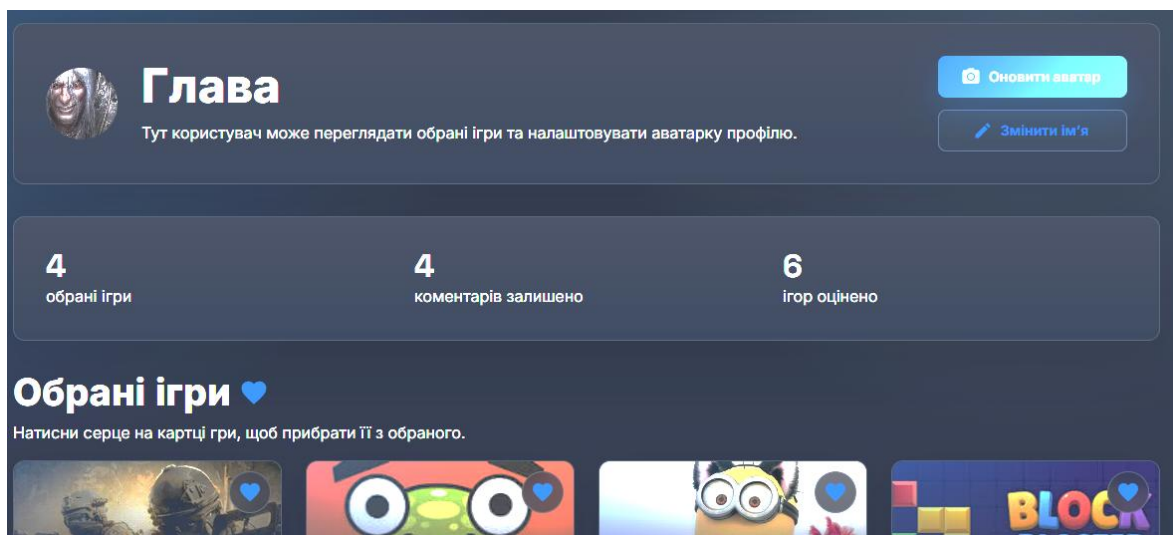


Рисунок 3.10 – Особистий кабінет користувача

Для відображення статистики в особистому кабінеті було створено окремий hook `useProfileStats`. Він отримує кількість коментарів і кількість оцінених ігор поточного користувача. Після отримання `userId` застосунок паралельно звертається до відповідних даних і повертає значення, які відображаються у профілі. Фрагмент реалізації формування статистики наведено в лістингу 3.11.

Лістинг 3.11 – Формування статистики активності користувача

```
function useProfileStats(userId: string | null) {
  const [commentCount, setCommentCount] = useState<number | null>(null)
  const [ratedGameCount, setRatedGameCount] = useState<number |
  null>(null)

  useEffect(() => {
    if (!userId) return

    const loadProfileStats = async () => {
      const [{ count: comments }, { count: ratings }] = await
      Promise.all([
        supabase
          .from('comments')
          .select('id', { count: 'exact', head: true })
          .eq('user_id', userId),
        supabase
          .from('ratings')
          .select('id', { count: 'exact', head: true })
          .eq('user_id', userId),
      ])
    }
  })
}
```

```

    })

    setCommentCount(comments ?? 0)
    setRatedGameCount(ratings ?? 0)
  }

  void loadProfileStats()
  }, [userId])

  return { commentCount, ratedGameCount }
}

```

Кінець лістингу 3.11

Окрім функцій звичайного користувача, у веб-платформі передбачено окрему роль адміністратора. Користувач із цією роллю отримує доступ до адміністративної панелі (рис. 3.11), у якій реалізовано керування основними даними платформи.

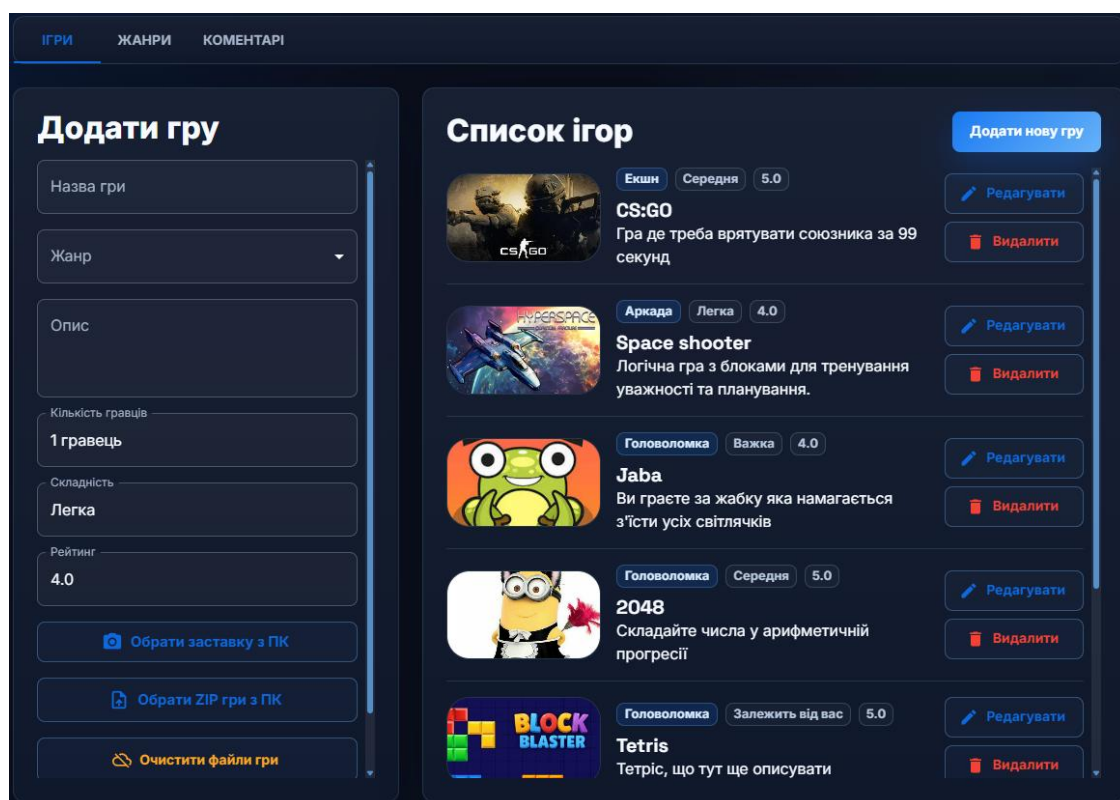


Рисунок 3.11 – Вкладка «Ігри» адміністративної панелі

У вкладці «Ігри» адміністратор може додавати нові ігри до каталогу, переглядати список уже створених записів, редагувати їхні дані та видаляти ігри

з платформи. Форма додавання гри містить поля для введення назви, жанру, опису, кількості гравців, складності та рейтингу. Також передбачено вибір зображення-заставки й ZIP-архіву з файлами HTML5-гри.

На початковому етапі для зберігання файлів ігор розглядався Supabase Storage, однак під час перевірки запуску HTML5-ігор було виявлено проблеми з коректним відображенням і взаємодією гри в браузері. Тому для зберігання ігрових файлів було обрано Cloudflare R2 у поєднанні з Cloudflare Worker, що дозволило коректно віддавати файли гри та використовувати їх у межах веб-платформи. Підключення R2-бакета до Cloudflare Worker виконується у файлі `wrangler.toml`. У цьому файлі задається назва Worker, шлях до основного файлу, змінна з публічною адресою для доступу до ігор та прив'язка R2-бакета через `binding`. Саме ця прив'язка дозволяє Worker звертатися до сховища через об'єкт `env.GAME_BUCKET`. Фрагмент налаштування Cloudflare Worker наведено в лістингу 3.12.

Лістинг 3.12 – Підключення Cloudflare R2 до Worker

```
name = "gamletland-game-upload"
main = "cloudflare-worker/src/index.ts"
compatibility_date = "2026-05-26"

[vars]
PUBLIC_GAMES_BASE_URL = "https://example.r2.dev"
SUPABASE_URL = "https://example.supabase.co"
SUPABASE_ANON_KEY = "example_publishable_key"

[[r2_buckets]]
binding = "GAME_BUCKET"
bucket_name = "gamletland-games"
```

Кінець лістингу 3.12

Після налаштування прив'язки Worker може завантажувати файли гри до R2. Під час обробки ZIP-архіву кожен файл гри записується у відповідну папку сховища. Для кожного файлу також задається тип вмісту та параметри кешування. Після завершення завантаження формується посилання `playUrl`, яке вказує на файл `index.html` гри та надалі використовується на сторінці гри для

запуску через `iframe`. Фрагмент запису файлів до R2 і формування посилання запуску наведено в лістингу 3.13.

Лістинг 3.13 – Завантаження файлів гри до Cloudflare R2

```
await env.GAME_BUCKET.put(`${uploadRoot}/${relativePath}`, uploadBytes, {
  httpMetadata: {
    contentType: getContentType(relativePath),
    cacheControl: 'public, max-age=31536000, immutable',
  },
})

const baseUrl = env.PUBLIC_GAMES_BASE_URL.replace(/\/+$/, '')
const playUrl = `${baseUrl}/${uploadRoot}/index.html`
```

Кінець лістингу 3.13

Після завантаження файлів HTML5-гри до Cloudflare R2 формується посилання `playUrl`, яке вказує на файл `index.html` гри. Разом із назвою, жанром, описом, рейтингом, рівнем складності, кількістю гравців і зображенням-заставкою це посилання зберігається в даних гри. На основі цих даних у каталозі формується картка гри, а кнопка відкриття використовує `playUrl` для переходу на сторінку запуску гри.

Наступною вкладкою адміністративної панелі є керування жанрами. Жанри використовуються під час додавання гри до каталогу, а також у фільтрації ігор на головній сторінці. У цій вкладці адміністратор може додати новий жанр або видалити вже наявний. Вкладку керування жанрами зображено на рисунку 3.12.

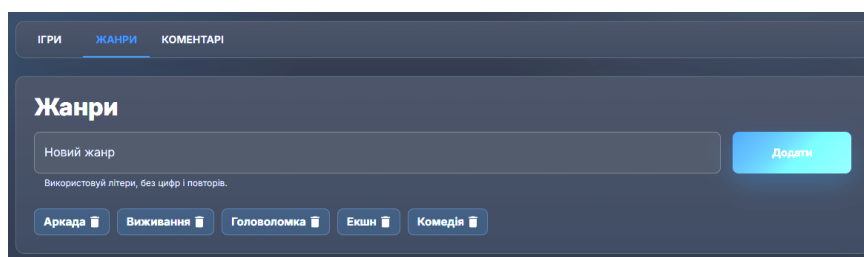


Рисунок 3.12 – Вкладка «Жанри» в адміністративній панелі

Логіка додавання жанру реалізована у функції `addGenre`. Перед збереженням назва жанру очищується від зайвих пробілів. Якщо поле порожнє, запит до Supabase не виконується. У разі коректного введення новий жанр додається до таблиці `genres`, після чого поле введення очищується, оновлюються дані адміністративної панелі та каталог ігор. Якщо під час додавання виникає помилка, користувачу відображається відповідне повідомлення. Відповідний код реалізації додавання жанру наведено в лістингу 3.14.

Лістинг 3.14 – Реалізація додавання жанру

```
const addGenre = async () => {
  const name = newGenreName.trim()

  if (!name) return

  const { error } = await supabase
    .from('genres')
    .insert({ name })

  if (error) {
    setMessage(`Не вдалося додати жанр: ${error.message}`)
    return
  }

  setNewGenreName('')
  await loadAdminData(false)
  await onCatalogChange()
}
```

Кінець лістингу 3.14

Для коректності даних назви жанрів мають бути унікальними та не повинні містити зайвих або некоректних значень.

Останньою вкладкою адміністративної панелі є модерація коментарів. У ній адміністратор має централізований доступ до коментарів, залишених користувачами до різних ігор. Для кожного запису відображається назва гри, ім'я автора, дата публікації та текст коментаря. Це дозволяє швидко переглядати вміст коментарів без переходу на сторінки окремих ігор. Вкладку модерація коментарів зображена на рисунку 3.13.

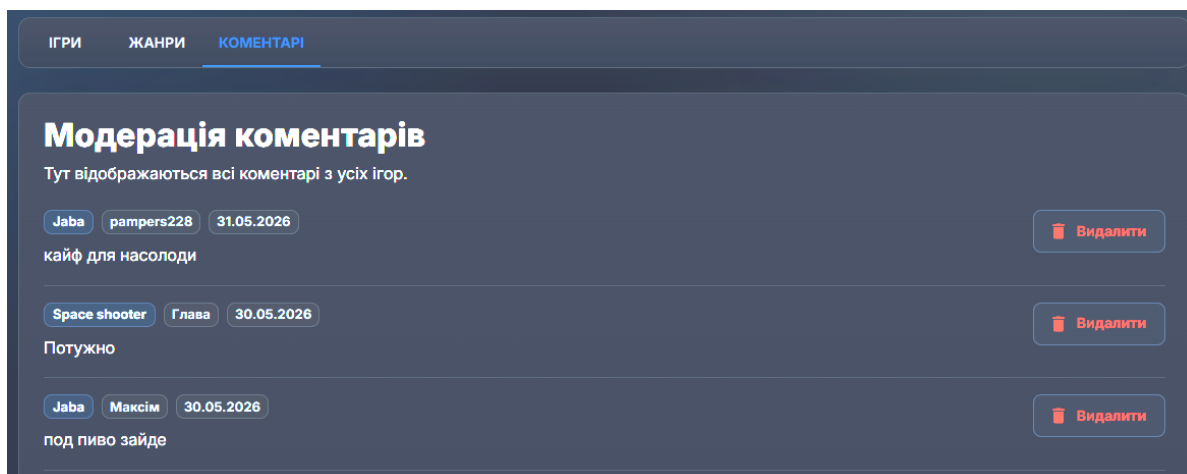


Рисунок 3.13 – Вкладка «Модерація коментарів» адміністративної панелі

Отже, у межах практичної реалізації було створено основні сторінки веб-платформи, реалізовано каталог ігор, запуск HTML5-ігор у браузері, авторизацію користувачів, функції обраного, оцінювання, коментування, особистий кабінет та адміністративну панель. Описані функції працюють із даними, що зберігаються в Supabase, тому наступним етапом є розгляд структури бази даних веб-платформи.

3.2 Розробка бази даних

Для забезпечення роботи веб-платформи GamletLand необхідно було організувати централізоване зберігання даних. До таких даних належать записи про ігри, жанри, профілі користувачів, ролі, коментарі, оцінки та обрані ігри.

Для реалізації збереження даних було використано Supabase. Цей сервіс надає базу даних PostgreSQL, механізм авторизації користувачів і програмний інтерфейс для взаємодії з даними безпосередньо з frontend-застосунку. У проєкті взаємодія з базою даних відбувається через клієнт Supabase, створений у файлі `supabaseClient.ts`. Саме через нього виконуються запити на отримання, додавання, оновлення та видалення записів у таблицях.

Структура бази даних була побудована відповідно до основних сутностей веб-платформи. Для цього було створено окремі таблиці, які відповідають за

зберігання ігор, жанрів, профілів користувачів, коментарів, оцінок та обраних ігор. Для наочного відображення структури бази даних було сформовано ER-діаграму. На ній показано основні таблиці веб-платформи, їхні поля та зв'язки між сутностями. ER-діаграма бази даних наведена на рисунку 3.14.

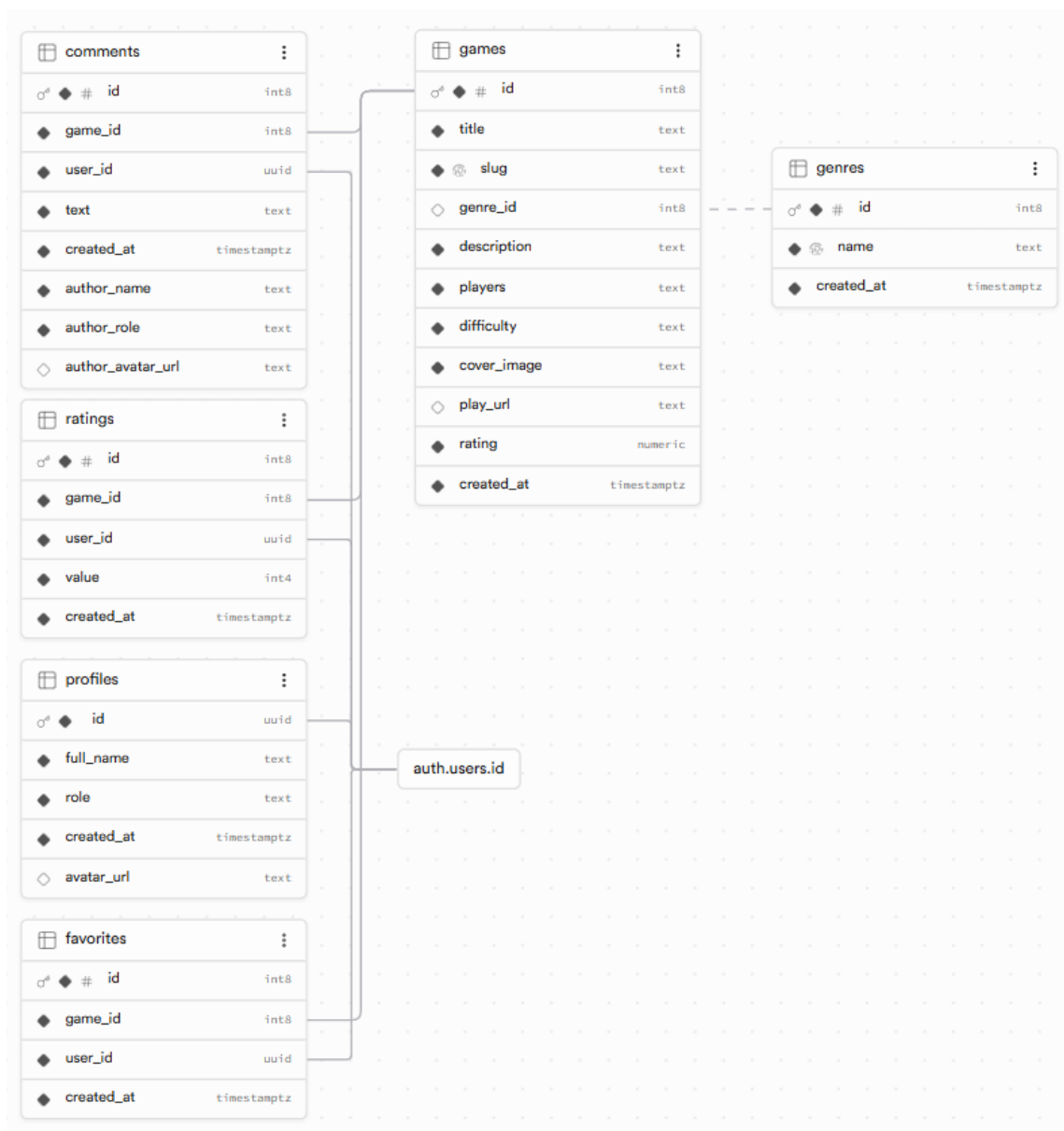


Рисунок 3.14 – ER-діаграма бази даних веб-платформи GamletLand

Центральною таблицею є Games, у якій зберігається основна інформація про ігри. Таблиця Genres використовується для зберігання жанрів, а зв'язок між грою та жанром реалізовано через поле genre_id. Таблиці Comments, Ratings і Favorites пов'язані з іграми через поле game_id, а з користувачами – через поле

user_id. Дані профілю користувача зберігаються в таблиці Profiles, яка пов’язана з auth.users за ідентифікатором користувача. Обліковий запис користувача створюється засобами Supabase Auth, а таблиця profiles використовується для збереження імені користувача, його ролі та аватара. Поле role має значення user або admin, що дозволяє розмежовувати доступ до звичайних функцій платформи та адміністративної панелі. SQL-запит створення таблиці profiles наведено в лістингу 3.15.

Лістинг 3.15 – Таблиця профілів (Profiles)

```
create table if not exists public.profiles (
  id uuid primary key references auth.users(id) on delete cascade,
  full_name text not null,
  role text not null default 'user' check (role in ('user', 'admin')),
  avatar_url text,
  created_at timestamptz not null default now()
);
```

Кінець лістингу 3.15

Якщо необхідно надати користувачу права адміністратора, значення цього поля змінюється на admin. Оскільки профіль користувача пов’язаний із записом у auth.users, визначення потрібного користувача може виконуватися за його електронною поштою. SQL-запит для призначення ролі адміністратора наведено в лістингу 3.16.

Лістинг 3.16 – Призначення ролі адміністратора користувачу

```
UPDATE public.profiles
SET role = 'admin'
WHERE id = (
  SELECT id
  FROM auth.users
  WHERE email = 'your-admin-email@example.com'
);
```

Кінець лістингу 3.16

Після виконання цього запиту користувач отримує роль адміністратора, і після авторизації йому стає доступна адміністративна панель веб-платформи.

Окрім поля `role`, для захисту даних у Supabase використовуються політики доступу. Вони визначають, які дії може виконувати користувач із певною роллю. Більшість політик мають схожу структуру, тому в роботі наведено один приклад політики (ліст. 3.17), яка дозволяє створювати записи про ігри лише адміністраторам.

Лістинг 3.17 – Політика доступу для створення ігор адміністратором

```
create policy "Admins can create games"
on public.games for insert
to authenticated
with check (public.is_admin());
```

Кінець лістингу 3.17

У цій політиці використовується функція `public.is_admin()`, яка перевіряє, чи має поточний користувач роль адміністратора. За таким самим принципом можуть обмежуватися інші дії, зокрема редагування і видалення ігор або модерація коментарів.

Таблиця `Profiles` також містить поле `avatar_url`, яке використовується для збереження посилання на аватар користувача. Для зберігання зображень у веб-платформі використовується Supabase Storage. У ньому зберігаються аватари користувачів і зображення-заставки ігор, а в таблицях бази даних зберігаються лише посилання на відповідні файли. Зокрема, у таблиці `profiles` для аватара використовується поле `avatar_url`, а в таблиці `games` для зображення гри – поле `cover_image`. Фрагмент налаштування сховища Supabase Storage наведено в лістингу 3.18.

Лістинг 3.18 – Збереження аватарів у Supabase Storage

```
alter table public.profiles
add column if not exists avatar_url text;

insert into storage.buckets (id, name, public, file_size_limit,
allowed_mime_types)
```

```

values (
  'avatars',
  'avatars',
  true,
  2097152,
  array['image/jpeg', 'image/png', 'image/webp', 'image/gif']
);
create policy "Users can upload own avatars"
on storage.objects for insert
to authenticated
with check (
  bucket_id = 'avatars'
  and (storage.foldername(name))[1] = auth.uid()::text
);

```

Кінець лістингу 3.18

Основною таблицею бази даних є games, оскільки саме вона зберігає дані, необхідні для відображення гри в каталозі та її запуску на сторінці гри. SQL-запит створення таблиці Games наведено в лістингу 3.19.

Лістинг 3.19 – Таблиця ігор (Games)

```

create table if not exists public.games (
  id bigint generated always as identity primary key,
  title text not null,
  slug text not null unique,
  genre_id bigint references public.genres(id) on delete set null,
  description text not null,
  players text not null default '1 гравець',
  difficulty text not null default 'Легка',
  cover_image text not null,
  play_url text,
  rating numeric(2, 1) not null default 0,
  created_at timestamptz not null default now()
);

```

Кінець лістингу 3.19

У таблиці games поле cover_image використовується для збереження посилання на зображення гри, а поле play_url – для посилання на головний файл HTML5-гри index.html. Файли браузерних ігор розміщуються в онлайн-сховищі Cloudflare R2. Оскільки HTML5-гра складається з кількох пов’язаних ресурсів, для кожної гри в R2 створюється окрема папка, у якій зберігаються файли,

необхідні для її запуску в браузері. Приклад зберігання файлів ігор у Cloudflare R2 наведено на рисунку 3.15.

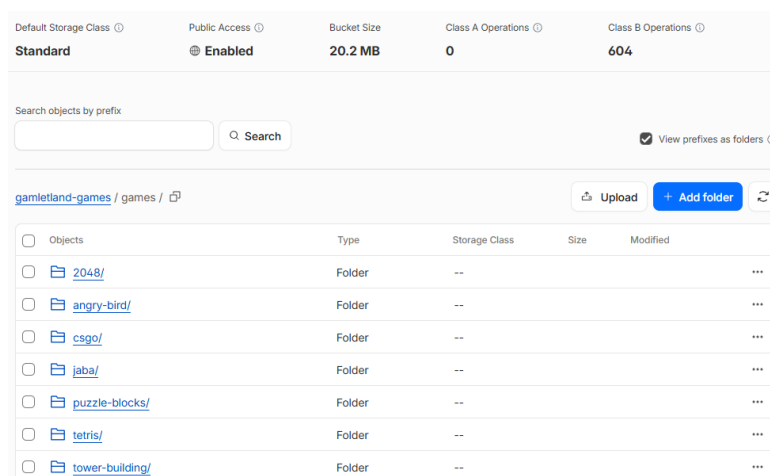


Рисунок 3.15 – Зберігання файлів HTML5-ігор у Cloudflare R2

Для завантаження файлів ігор використовується Cloudflare Worker. Він виконує роль проміжного обробника між адміністративною панеллю та Cloudflare R2: приймає ZIP-архів із файлами гри, розпаковує його, завантажує отримані файли до відповідної папки в R2 та формує посилання на файл index.html. Сформоване посилання зберігається в таблиці games у полі play_url і надалі використовується frontend-застосунком для запуску гри через iframe.

3.3 Розробка та експорт браузерних ігор у Construct 2

Важливою частиною реалізації веб-платформи є наповнення її ігровим контентом. Для створення браузерних ігор у межах проєкту використовується Construct 2. Це середовище розробки дає змогу створювати ігри за допомогою візуального редактора, системи об'єктів, властивостей і подій.

Першим етапом створення гри є підготовка графічних ресурсів. До них належать спрайти персонажів, об'єктів, перешкод, фону, елементів інтерфейсу та інших візуальних складових. Такі ресурси можна створювати самостійно в

графічних редакторах або використовувати готові набори з відкритих джерел, якщо вони відповідають стилю гри та умовам використання.

Після підготовки графічні ресурси додаються до проекту Construct 2 як об'єкти. Їм можна задати розмір, початкове положення на сцені та інші властивості. Після додавання спрайт можна розміщувати на ігровій сцені, копіювати, масштабувати та використовувати як окремий елемент майбутнього рівня. Розміщення об'єктів виконується у візуальному редакторі сцени (рис. 3.16).

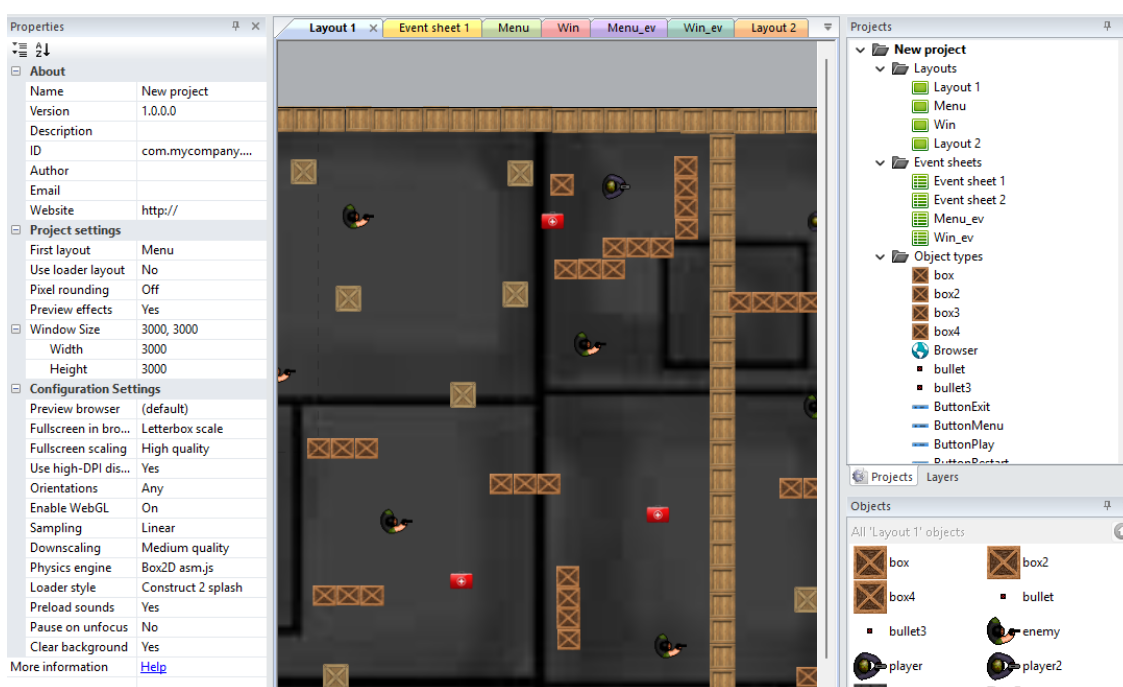
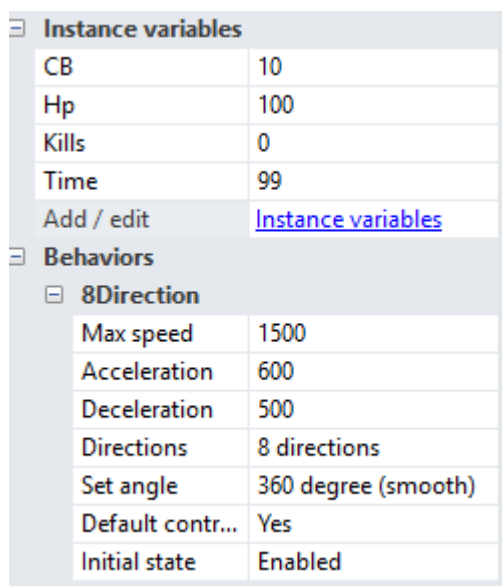


Рисунок 3.16 – Середовище розробки гри в Construct 2

На цьому етапі формується початковий вигляд гри: додається фон, розташовуються основні об'єкти, визначається межа ігрового поля та приблизна структура рівня.

Окрім основної ігрової сцени, у Construct 2 можуть використовуватися додаткові Layout, які відповідають за окремі екрани гри. Наприклад, один Layout може використовуватися для головного меню, інший – для основного рівня, а ще один – для екрана завершення гри. Такий підхід дозволяє розділити різні частини гри та не перевантажувати одну сцену всіма елементами одразу.

Наступним етапом є додавання головного героя та налаштування його властивостей (рис. 3.17), щоб він міг взаємодіяти з навколишнім світом.



Instance variables	
CB	10
Hp	100
Kills	0
Time	99
Add / edit	Instance variables

Behaviors	
8Direction	
Max speed	1500
Acceleration	600
Deceleration	500
Directions	8 directions
Set angle	360 degree (smooth)
Default contr...	Yes
Initial state	Enabled

Рисунок 3.17 – Налаштування змінних і поведінки персонажа

Для персонажа використовуються змінні екземпляра, які зберігають його основні ігрові показники. Наприклад, змінна Hp відповідає за кількість здоров'я, Kills – за кількість знищених ворогів, а Time – за час проходження або тривалість ігрової сесії. Такі змінні дозволяють зберігати поточний стан об'єкта та використовувати його в подіях гри.

Крім змінних, до об'єктів додаються поведінки. Для керованого персонажа використовується поведінка 8Direction, яка дозволяє рухатися у восьми напрямках. У її параметрах можна налаштувати максимальну швидкість, прискорення, уповільнення, кількість напрямків руху та початковий стан. Завдяки цьому персонаж отримує готову механіку переміщення, яку надалі можна поєднувати з подіями стрільби, зіткненнями з ворогами та оновленням ігрових показників.

Після налаштування об'єктів і їхніх властивостей формується логіка гри. У Construct 2 вона реалізується за допомогою Event Sheet, де кожна подія складається з умови та відповідної дії. Такий підхід дозволяє описувати

поведінку гри у візуальному форматі: якщо виконується певна умова, система виконує задану дію.

У Shooter-грі з виглядом зверху одним із базових елементів є керування персонажем. Для цього використовуються події клавіатури: натискання клавіш W, S, A і D імітує рух об'єкта з поведінкою 8Direction у відповідному напрямку (рис. 3.18). Завдяки цьому персонаж може рухатися вгору, вниз, ліворуч і праворуч, а сама логіка керування залишається зрозумілою та легко змінюваною.

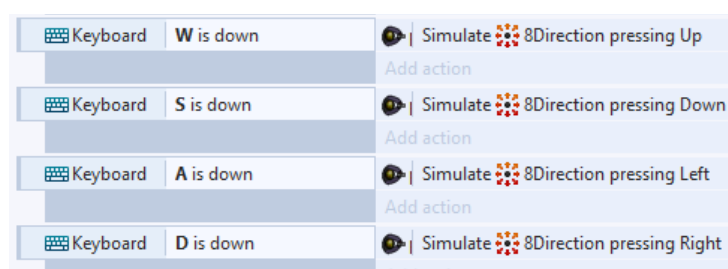


Рисунок 3.18 – Реалізація керування персонажем

Окремо в Event Sheet реалізовано оновлення ігрової статистики. Подія Every tick виконується постійно під час роботи гри, тому вона використовується для оновлення показників на екрані. У цьому випадку текстові об'єкти відображають поточне здоров'я персонажа, кількість куль, кількість знищених ворогів і час гри (рис. 3.19). Значення беруться зі змінних об'єкта player, зокрема Hp, CB, Kills і Time.

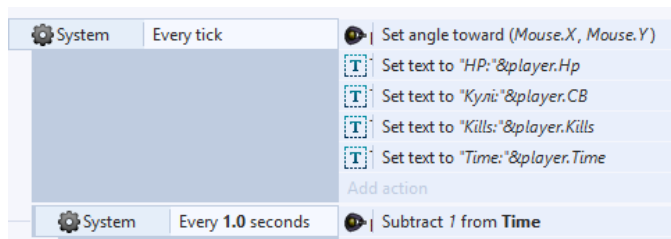


Рисунок 3.19 – Відображення ігрової статистики

Також використовується подія Every 1.0 seconds, яка щосекунди зменшує значення змінної Time. Завдяки цьому в грі реалізовано простий таймер, який

може використовуватися для обмеження часу проходження або відстеження тривалості ігрової сесії.

Наступним важливим елементом логіки Shooter-гри є реакція персонажа на влучання снарядів (рис. 3.20).

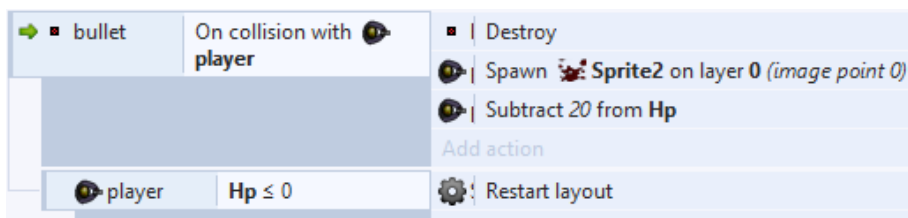


Рисунок 3.20 – Обробка зіткнення снаряда з персонажем

Влучання реалізовано через подію зіткнення між об'єктами bullet і player. Коли таке зіткнення відбувається, снаряд знищується, у місці влучання створюється візуальний ефект, а значення здоров'я персонажа зменшується.

Після завершення налаштування ігрових сцен, об'єктів і подій гра готується до експорту. У Construct 2 для цього використовується функція експорту проєкту, яка дозволяє обрати платформу, для якої буде сформовано готову гру. Оскільки веб-платформа GamletLand запускає ігри безпосередньо у браузері, для експорту обирається варіант HTML5 website (рис. 3.21).

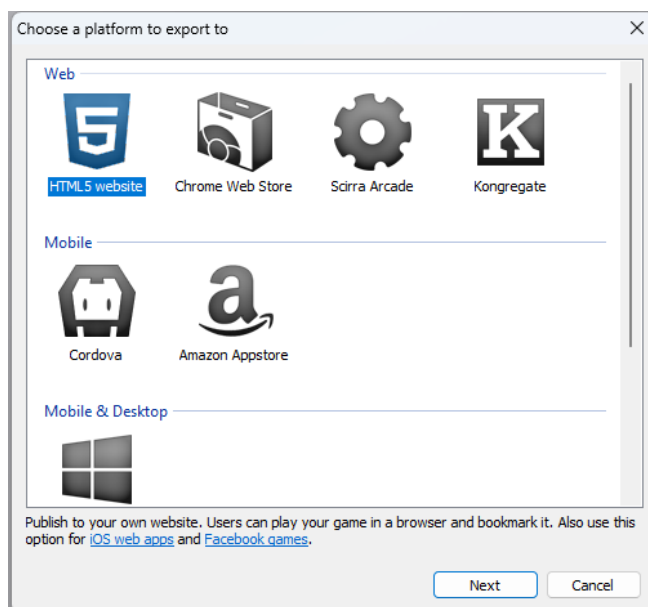


Рисунок 3.21 – Вибір платформи для експорту гри в Construct 2

Після вибору цього варіанта Construct 2 формує набір файлів, необхідних для запуску гри у браузері. До нього входить головний файл index.html, а також JavaScript-файли, зображення, звуки та інші ресурси гри. Саме файл index.html є точкою запуску, оскільки через нього браузер відкриває гру та підвантажує всі необхідні ресурси.

3.4 Тестування та налагодження вебплатформи

У межах даної роботи тестування вебплатформи з браузерними іграми проводилося на різних етапах розробки з метою перевірки її стабільності, коректності роботи основних функцій та взаємодії між окремими частинами системи.

Процес тестування охоплював перевірку функціональних можливостей платформи, правильності обробки дій користувача, збереження даних у базі даних, завантаження файлів, запуску HTML5-ігор у браузері та роботи користувацького інтерфейсу. Особлива увага приділялася сценаріям, які пов'язані з одночасною роботою кількох компонентів системи. До таких сценаріїв належать авторизація користувачів, відображення каталогу ігор, пошук і фільтрація, додавання гри до обраного, оцінювання, коментування, робота адміністративної панелі та запуск гри на сторінці платформи.

У процесі розробки було застосовано функціональне тестування, яке дозволило перевірити відповідність реалізованих можливостей поставленим вимогам. Перевірка здійснювалася шляхом виконання тестових сценаріїв, що відображають типові дії користувачів вебплатформи. Для кожного сценарію визначалися вхідні дані, очікуваний результат та фактичний результат виконання. Це дозволило виявити можливі помилки в роботі системи та усунути їх під час розробки.

Окрему увагу було приділено перевірці взаємодії frontend-застосунку з зовнішніми сервісами. Зокрема, перевірялася коректність роботи з Supabase, Supabase Storage, Cloudflare R2 та Cloudflare Worker. Це було важливо, оскільки

частина даних зберігається в базі даних, зображення розміщуються у файловому сховищі, а файли HTML5-ігор завантажуються та відкриваються через Cloudflare R2. Така перевірка дозволила переконатися, що всі частини системи працюють узгоджено.

Важливим етапом стало тестування запуску браузерних ігор. Під час перевірки аналізувалося, чи правильно формується посилання на файл запуску гри, чи відкривається HTML5-гра на сторінці платформи, а також чи завантажуються всі необхідні ресурси гри. Оскільки ігри експортуються у форматі HTML5 та складаються з кількох пов'язаних файлів, цей сценарій був одним із ключових для всієї веб-платформи.

Також проводилося тестування користувацького інтерфейсу. Перевірялася зручність навігації, коректність переходів між сторінками, відображення карток ігор, робота кнопок, форм і повідомлень системи. Це дозволило оцінити, наскільки зручно користувач може взаємодіяти з платформою та виконувати основні дії без додаткових пояснень.

Додатково перевірялася коректність роботи ролей користувачів. Звичайний користувач мав доступ лише до основних функцій платформи, тоді як адміністративні можливості були доступні лише користувачу з роллю admin. Це дало змогу перевірити правильність розмежування доступу та роботу політик безпеки.

Під час виконання тестових сценаріїв перевірявся не лише візуальний результат на сторінці, а й коректність змін у системі після виконання дії. Наприклад, після додавання гри до обраного, оцінювання або створення коментаря перевірялося, чи зберігаються відповідні дані та чи правильно вони відображаються після оновлення сторінки.

Таким чином, тестування дозволило перевірити роботу вебплатформи з позиції різних користувацьких сценаріїв, оцінити стабільність основних модулів та переконатися в коректній взаємодії інтерфейсу, бази даних, файлових сховищ і механізму запуску ігор. Для фіксації результатів було сформовано набір

тестових сценаріїв, які охоплюють основні функції веб-платформи. Результати функціонального тестування наведено в таблиці 3.1.

Таблиця 3.1 – Результати функціонального тестування веб-платформи

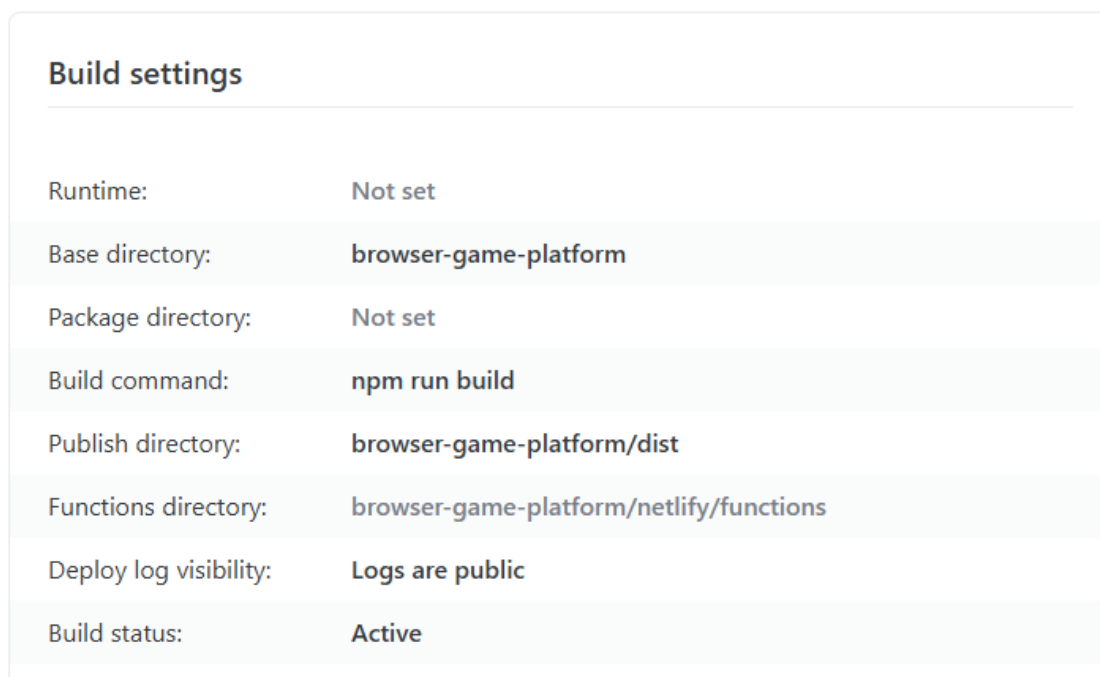
№	Тестовий сценарій	Вхідні дані	Очікуваний результат	Фактичний результат
1	Реєстрація користувача	Ім'я, email, пароль	Створення нового облікового запису	Успішно
2	Авторизація користувача	Email і пароль	Вхід у систему	Успішно
3	Перегляд каталогу ігор	Відкриття головної сторінки	Відображення списку ігор	Успішно
4	Пошук гри	Назва гри	Відображення гри	Успішно
5	Фільтрація за жанром	Обраний жанр	Відображення ігор, які відповідають жанру	Успішно
6	Запуск гри	Натискання кнопки запуску	Відкриття гри	Успішно
7	Додавання до обраного	Картка гри	Запис гри до списку обраного	Успішно
8	Оцінювання гри	Обрана оцінка	Оновлення оцінки гри	Успішно
9	Додавання коментаря	Текст коментаря	Збереження коментаря та його відображення	Успішно
10	Додавання гри	Назва, опис, жанр, заставка, архів гри	Створення картки гри у каталозі	Успішно

За результатами тестування основні функції веб-платформи працюють коректно. Перевірка охопила сценарії для гостя, авторизованого користувача та адміністратора. Окремо було протестовано додавання гри через адміністративну панель, оскільки цей процес поєднує роботу бази даних, Supabase Storage, Cloudflare R2 та Cloudflare Worker. Отримані результати підтвердили працездатність основних модулів системи.

Після завершення локального тестування, вебплатформу було розгорнуто на сервісі Netlify. Оскільки клієнтська частина проєкту реалізована як React-застосунок на основі Vite, перед публікацією виконується збірка проєкту. Для цього в налаштуваннях Netlify було вказано команду `npm run build`, яка створює оптимізовану версію застосунку для розгортання.

Після виконання збірки сформовані файли розміщуються в директорії `dist`. Саме ця директорія була вказана як `publish directory`, тобто папка, з якої Netlify

публікує готовий сайт. Налаштування збірки та директорії публікації веб-платформи наведено на рисунку 3.23.

The image shows a screenshot of the 'Build settings' interface in Netlify. It features a title 'Build settings' at the top left, followed by a horizontal line. Below this, there is a table with two columns: the left column contains labels for various settings, and the right column contains their current values. The settings include Runtime, Base directory, Package directory, Build command, Publish directory, Functions directory, Deploy log visibility, and Build status. The values are: Not set, browser-game-platform, Not set, npm run build, browser-game-platform/dist, browser-game-platform/netlify/functions, Logs are public, and Active. The table has alternating light gray and white rows.

Build settings	
Runtime:	Not set
Base directory:	browser-game-platform
Package directory:	Not set
Build command:	npm run build
Publish directory:	browser-game-platform/dist
Functions directory:	browser-game-platform/netlify/functions
Deploy log visibility:	Logs are public
Build status:	Active

Рисунок 3.23 – Налаштування збірки веб-платформи на Netlify

Після налаштування параметрів збірки проєкт було успішно зібрано та розгорнуто на Netlify. Статус Build status: Active підтверджує, що Netlify коректно обробляє проєкт і використовує директорію dist для розгортання веб-платформи.

Після розгортання було повторно перевірено доступність основних сторінок платформи, роботу маршрутизації та коректне завантаження даних у розгорнутому середовищі.

ВИСНОВКИ

У кваліфікаційній роботі було досягнуто поставлену мету, яка полягала у створенні веб-платформи з браузерними іграми з використанням React і Construct 2. У результаті розроблено веб-застосунок, який забезпечує перегляд каталогу ігор, пошук і фільтрацію, запуск HTML5-ігор безпосередньо у браузері, взаємодію користувачів із контентом та адміністрування ігрових матеріалів.

Було здійснено аналіз сучасного стану розвитку браузерних ігор та веб-платформ для їх розміщення. Розглянуто існуючі аналоги, визначено їхні переваги та недоліки. На основі проведеного аналізу обґрунтовано доцільність створення власної веб-платформи, яка поєднує каталог ігор, користувацькі функції, адміністративне керування та можливість запуску HTML5-ігор у браузері.

Визначено функціональні та нефункціональні вимоги до розроблюваної вебплатформи. Було сформовано основні ролі користувачів: гість, авторизований користувач та адміністратор. Для кожної ролі визначено доступні можливості та сценарії взаємодії із системою. Це дозволило сформулювати чітке бачення структури майбутнього вебзастосунку та його основних функцій.

Було спроектовано структуру вебплатформи, визначено основні сторінки, компоненти та логіку взаємодії між ними. У межах проектування передбачено головну сторінку з каталогом ігор, сторінку окремої гри, особистий кабінет користувача, сторінки авторизації та реєстрації, а також адміністративну панель для керування вмістом платформи.

Розроблено клієнтську частину веб-платформи з використанням React, TypeScript та сучасних засобів розробки. Реалізовано інтерфейс користувача, маршрутизацію між сторінками, відображення каталогу ігор, пошук, фільтрацію, сторінку запуску гри, особистий кабінет та адміністративні розділи. Використання React дозволило створити компонентну структуру застосунку, зручну для подальшого розширення та підтримки.

Реалізовано засоби зберігання даних веб-платформи. Для роботи з даними використано Supabase, у якому створено структуру бази даних для зберігання інформації про ігри, жанри, профілі користувачів, коментарі, оцінки та обрані ігри. Для зберігання зображень використано Supabase Storage, а для розміщення файлів HTML5-ігор інтегровано Cloudflare R2 та Cloudflare Worker. Такий підхід дозволив розділити структуровані дані платформи та файлові ресурси ігор.

Було реалізовано каталог браузерних ігор із можливістю перегляду, пошуку, фільтрації та запуску ігор безпосередньо у браузері. Окремо забезпечено інтеграцію HTML5-ігор, створених за допомогою Construct 2. Після експорту гри у формат HTML5 її файли можуть бути завантажені до сховища, а сформоване посилання на файл запуску використовується для відкриття гри на сторінці платформи.

Реалізовано взаємодію користувачів із платформою. Авторизовані користувачі можуть додавати ігри до обраного, оцінювати їх та залишати коментарі. Також було розроблено особистий кабінет, у якому користувач може переглядати власну інформацію, статистику активності та обрані ігри.

Розроблено адміністративну частину веб-платформи. Адміністратор має можливість керувати іграми, жанрами та користувацьким контентом. Для розмежування доступу реалізовано рольову модель, яка дозволяє обмежити адміністративні функції лише для користувачів із відповідними правами.

Проведено тестування та налагодження веб-платформи. Було перевірено основні сценарії роботи гостя, авторизованого користувача та адміністратора. Результати тестування підтвердили коректну роботу основних функцій системи, зокрема авторизації, каталогу ігор, пошуку, фільтрації, запуску HTML5-ігор, оцінювання, коментування, додавання до обраного та адміністративного керування. Після завершення локального тестування веб-застосунок було розгорнуто на платформі Netlify.

Практичне значення одержаних результатів полягає у створенні працездатної веб-платформи, яка може використовуватися для розміщення, перегляду та запуску браузерних ігор. Розроблене рішення може бути

застосоване як навчальний або демонстраційний проєкт для дослідження інтеграції React-застосунку, бази даних, хмарного сховища та HTML5-ігор.

Подальше вдосконалення веб-платформи може передбачати розширення каталогу ігор, додавання системи рекомендацій, покращення статистики користувачів, реалізацію рейтингових списків, удосконалення модерації контенту та адаптацію інтерфейсу для більшої кількості пристроїв. Також перспективним напрямом є розширення можливостей адміністративної панелі та автоматизація процесу перевірки завантажених ігрових файлів.

Таким чином, усі поставлені завдання кваліфікаційної роботи було виконано в повному обсязі. Розроблена вебплатформа відповідає визначеним вимогам, забезпечує основні функції для роботи з браузерними іграми та підтверджує можливість практичного використання сучасних вебтехнологій для створення інтерактивних ігрових платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Construct 2 Online Manual & Documentation Construct. URL: <https://www.construct.net/en/tutorials?flang=48> (дата звернення: 04.03.2026).
2. Quick Start – React. React. URL: <https://react.dev/learn> (дата звернення: 25.04.2026).
3. Handbook – The TypeScript Handbook. TypeScript: JavaScript With Syntax For Types. URL: <https://www.typescriptlang.org/docs/handbook/intro.html> (дата звернення: 07.03.2026).
4. Supabase Docs. Supabase. The Postgres Development Platform. URL: <https://supabase.com/docs> (дата звернення: 12.03.2026).
5. Cloudflare R2. Cloudflare Docs. URL: <https://developers.cloudflare.com/r2/> (дата звернення: 18.03.2026).
6. UML Class Diagram Tutorial. Visual Paradigm – AI-Powered Visual Modeling Platform. URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/> (дата звернення: 25.03.2026).
7. Get started with GitHub documentation – GitHub Docs. GitHub Docs. URL: <https://docs.github.com/en/get-started> (дата звернення: 01.04.2026).
8. Getting Started. vitejs. URL: <https://vite.dev/guide/> (дата звернення: 05.04.2026).
9. Firebase. Google's Mobile and Web App Development Platform. Firebase. URL: <https://firebase.google.com/> (дата звернення: 11.04.2026).
10. Build configuration overview. Netlify Docs. URL: <https://docs.netlify.com/build/configure-builds/overview/> (дата звернення: 17.04.2026).
11. Free Design Tool for Websites, Product Design & More Figma. Figma. URL: <https://www.figma.com/design/> (дата звернення: 27.04.2026).
12. Overview – Material UI. MUI: The React component library you always wanted. URL: <https://mui.com/material-ui/getting-started/> (дата звернення: 01.05.2026).