Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр

на тему: Проєктування та розробка вебплатформи оцінювання ігор «Ludara» з використанням технології Node.js

Виконав(ла): студент(ка) IV курсу, групи СПс-41 спеціальності 121 Інженерія програмного забезпечення

	(підпис)	Мороз Д. І. (прізвище та ініціали)
Керівник	(підпис)	Мудрик І. Я. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М.Р. (прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавра
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Морозу Даниїлу Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Проєктування та розробка вебплатформи оцінювання ігор «Ludara» з використанням технології Node.js

Керівник роботи Мудрик Іван Ярославович, доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «___» _____ 20___ року № _____
2. Термін подання студентом завершеної роботи
3. Вихідні дані до роботи Предметна область, завдання, вимоги та специфікація, програмне рішення, методичні вказівки

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступна частина
Аналіз предметної області та теоретичних основ
Визначення методики реалізації моделі
Реалізація моделі
Визначення основних аспектів охорони праці та безпеки життєдіяльності
Висновки роботи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації та діаграми процесів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,	Мариненко С. Ю. канд. техн. наук,		
основи охорони праці	доцент		
Нормоконтроль	Стоянов Ю.М. к.т.н., доц. каф. ПІ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання та ознайомлення з темою кваліфікаційної роботи	06.04.2026	
2	Аналіз предметної області та існуючих рішень	05.05.2026	
3	Визначення вимог до програмної системи та варіантів використання	08.05.2026	
4	Проєктування архітектури системи та схеми бази даних	09.05.2026	
5	Розробка серверної частини: автентифікація, каталог ігор, оцінювання	10.05.2026	
6	Розробка серверної частини: статуси, вподобання, профіль, адмін модуль	10.05.2026	
7	Розробка клієнтської частини та інтеграція з серверною	13.05.2026	
8	Тестування програмної системи та усунення виявлених дефектів	20.05.2026	
9	Розгортання та верифікація програмної системи	21.05.2026	
10	Виконання розділу з безпеки життєдіяльності та охорони праці	22.05.2026	
11	Оформлення пояснювальної записки	05.06.2026	
12	Оформлення графічного та презентаційного матеріалу	08.06.2026	
13	Попередній захист	10.06.2026	
14	Захист кваліфікаційної роботи		

Студент _____ Мороз Даниїл
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мудрик І.Я.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Проєктування та розробка вебплатформи оцінювання ігор «Ludara» з використанням технології Node.js // Кваліфікаційна робота освітнього рівня «Бакалавр» // Мороз Даниїл Іванович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПс-41 // Тернопіль, 2026 // С. 74, рис. – 20, табл. – 12, кресл. – 0, додат. – 7, бібліогр. – 31.

Ключові слова: веб-платформа, оцінювання відеоігор, Node.js, React, REST API, PostgreSQL, Prisma ORM, JWT автентифікація, RAWG API.

Кваліфікаційна робота присвячена проєктуванню та розробці веб-платформи для оцінювання відеоігор «Ludara» з використанням технології Node.js.

У першому розділі проведено аналіз предметної області, розглянуто існуючі аналогічні рішення та їх недоліки, сформульовано мету і завдання розробки, визначено акторів системи та описано ключові варіанти використання.

У другому розділі обрано модель розробки, спроєктовано архітектуру системи та схему бази даних, обґрунтовано вибір технологічного стеку та описано реалізацію основних модулів платформи.

У третьому розділі описано тестування функціональних модулів системи, визначено вимоги до розгортання платформи та проведено верифікацію відповідності розробленого продукту встановленим вимогам.

У четвертому розділі розглянуто питання безпеки життєдіяльності та основи охорони праці.

Об'єкт дослідження: процес проєктування та розробки веб-платформ для оцінювання цифрового контенту.

Предмет дослідження: методи та технології розробки веб-платформи для оцінювання відеоігор з використанням Node.js та React

ABSTRACT

Design and Development of the Game Rating Web Platform «Ludara» Using Node.js Technology // Qualification work of the educational level «Bachelor» // Moroz Danyiil Ivanovych // Ternopil Ivan Puluj National Technical University, Computer and Information Systems and Software Engineering Faculty, Software Engineering Department, group SPs-41 // Ternopil, 2026 // P. 74, fig. – 20, tabl. – 12, draw. – 0, annexes. – 7, references. – 31.

Keywords: web platform, video game rating, Node.js, React, REST API, PostgreSQL, Prisma ORM, JWT authentication, RAWG API.

The qualification work is devoted to the design and development of the video game rating web platform «Ludara» using Node.js technology.

The first chapter analyses the subject area of video game rating web platforms, reviews existing similar solutions and their shortcomings, formulates the purpose and objectives of the development, defines the system actors and describes the key use cases.

The second chapter presents the selected iterative development model, designs the client-server system architecture, develops the database schema, justifies the choice of the technology stack and describes the implementation of the main modules of the server and client parts of the platform.

The third chapter describes the testing process of all functional system modules with the development of test scenarios, defines system requirements and the platform deployment procedure, and verifies the compliance of the developed software product with the established requirements.

The fourth chapter examines the issues of life safety and fundamentals of labor protection.

Object of research: the process of designing and developing web platforms for digital content rating.

Subject of research: methods and technologies for developing a video game rating web platform using Node.js, React and PostgreSQL within a client-server architecture.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface

JWT – JSON Web Token

ORM – Object-Relational Mapper

REST – Representational State Transfer

CORS – Cross-Origin Resource Sharing

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

URL – Uniform Resource Locator

JSON – JavaScript Object Notation

UML – Unified Modeling Language

SQL – Structured Query Language

MVC – Model-View-Controller

SPA – Single Page Application

RAWG – Rapid API for Video Games

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

npm – Node Package Manager

SDK – Software Development Kit

MIME – Multipurpose Internet Mail Extensions

HMAC – Hash-based Message Authentication Code

SHA – Secure Hash Algorithm

RFC – Request for Comments

CRUD – Create Read Update Delete

UI – User Interface

DLC – Downloadable Content

NSFW – Not Safe For Work

CI/CD – Continuous Integration/Continuous Deployment

ЗМІСТ

ВСТУП

Відеоігрова індустрія є однією з найбільш динамічно зростаючих галузей цифрових розваг у світі. За останні десятиліття кількість випущених ігор зросла в рази, щороку на різних платформах з'являються тисячі нових проєктів різних жанрів та масштабів. В умовах такого різноманіття гравці все частіше потребують зручних та надійних інструментів для орієнтування у світі відеоігор, відстеження власного ігрового досвіду та обміну думками з іншими учасниками спільноти.

Існуючі платформи для оцінювання відеоігор вирішують лише частину цих потреб. Агрегатори професійних рецензій такі як Metacritic та OpenCritic зосереджені виключно на думці критиків і не надають користувачам повноцінних інструментів для власного оцінювання. Платформи з користувацькими відгуками такі як Steam обмежені бінарною системою оцінок та прив'язані до конкретного магазину. Бази даних ігор на кшталт RAWG мають великий каталог але слабко розвинену соціальну складову. Жодне з існуючих рішень не поєднує в собі детальну числову шкалу оцінювання, систему рецензій, розмежування ролей між критиками та звичайними користувачами і зручну ігрову бібліотеку в рамках однієї платформи.

Актуальність розробки платформи Ludara обумовлена потребою у вебзастосунку який поєднує функціональність каталогу ігор, платформи для оцінювання та особистого ігрового журналу. Використання технології Node.js дозволяє побудувати масштабовану та продуктивну серверну частину з єдиною мовою програмування для обох частин застосунку, що спрощує розробку та підтримку системи [2].

Метою кваліфікаційної роботи є проєктування та розробка веб-платформи для оцінювання відеоігор «Ludara» з використанням технології Node.js.

Для досягнення поставленої мети вирішувались такі завдання.

– Реалізація системи автентифікації користувачів на основі JWT токенів з підтримкою перегляду платформи без реєстрації.

– Розробка каталогу відеоігор з інтеграцією зовнішнього RAWG API та підтримкою пошуку, фільтрації і пагінації.

– Реалізація системи оцінювання ігор за числовою шкалою від 1 до 10 з можливістю написання текстових рецензій та вподобання рецензій інших користувачів.

– Розробка системи статусів ігор для ведення власної ігрової бібліотеки з трьома станами.

– Реалізація системи ролей користувачів з розмежуванням між звичайними користувачами, критиками та адміністраторами.

– Розробка адміністративної панелі для керування користувачами та моніторингу статистики платформи.

– Реалізація публічних профілів користувачів з детальною статистикою активності.

– Розробка зручного адаптивного інтерфейсу користувача з підтримкою темної та світлої теми оформлення.

Об’єктом дослідження є процес проєктування та розробки веб-платформ для оцінювання цифрового контенту.

Предметом дослідження є методи та технології розробки веб-платформи для оцінювання відеоігор з використанням Node.js, React та PostgreSQL в рамках клієнт-серверної архітектури.

Практичне значення отриманих результатів полягає у тому що розроблена платформа Ludara є повнофункціональним веб-застосунком готовим до використання реальними користувачами. Платформа реалізує повний цикл взаємодії користувача з відеоіграми від перегляду каталогу та читання рецензій до виставлення власних оцінок та ведення ігрової бібліотеки. Розроблена архітектура системи та обрані технологічні рішення можуть бути використані, як основа для подальшого розвитку платформи, додавання нових функцій, масштабування та розгортання у виробничому середовищі.

РОЗДІЛ 1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Цей розділ присвячено аналізу вимог до платформи Ludara. Розглядається предметна область веб-платформ для оцінювання відеоігор, проводиться огляд існуючих аналогічних рішень та визначаються їх недоліки. На основі проведеного аналізу формулюються мета та завдання розробки, визначаються актори системи та описуються ключові варіанти використання платформи.

1.1 Аналіз предметної області

Відеоігрова індустрія є однією з найбільш динамічно зростаючих галузей цифрових розваг. Щороку випускаються тисячі нових ігор на різних платформах, і користувачі все частіше потребують зручних інструментів для орієнтування в цьому потоці. Платформи для оцінювання та каталогізації відеоігор набули значної популярності, вони допомагають гравцям приймати обґрунтовані рішення щодо вибору ігор, стежити за власною ігровою бібліотекою та ділитися думками з іншими учасниками спільноти.

Такі платформи можна умовно поділити на два типи. Перший це агрегатори оцінок, які збирають рецензії від професійних видань і виводять загальний бал. Другий це користувацькі платформи, де самі гравці виставляють оцінки та пишуть рецензії. Деякі рішення поєднують обидва підходи, проте кожне з них має свої обмеження.

Для кращого розуміння предметної області було проаналізовано чотири найбільш відомі платформи: Metacritic, RAWG, OpenCritic та Steam. Аналіз дозволяє виявити сильні та слабкі сторони існуючих рішень і визначити напрямки, які варто врахувати при розробці власної платформи [7].

Metacritic є одним із найстаріших і найвідоміших агрегаторів оцінок у сфері відеоігор. Платформа збирає рецензії від авторитетних ігрових видань і на їх основі формує єдиний показник Metascore. Поруч з ним відображається також User Score, який формується з оцінок звичайних користувачів. Така дворівнева система дає

змогу бачити, як думку професійної критики, так і реакцію аудиторії. Проте Metacritic має суттєві недоліки: платформа не є відкритою для широкої спільноти в плані написання рецензій, інтерфейс морально застарів, а система User Score неодноразово ставала жертвою так званих «review bombing» атак, коли користувачі масово виставляли низькі оцінки з нігілістичних або політичних міркувань, не пов'язаних з якістю гри.

OpenCritic є більш сучасною альтернативою Metacritic і також зосереджена виключно на агрегації професійних рецензій. Платформа відрізняється прозорішим підходом до підрахунку балів і чіткіше вказує, які саме видання враховуються при формуванні загальної оцінки. Однак OpenCritic повністю відмовилась від користувацьких оцінок, а це робить платформу корисним інструментом для ознайомлення з думкою критиків, але позбавляє її будь-якої соціальної складової та взаємодії між користувачами.

RAWG позиціонує себе насамперед, як найбільша база даних відеоігор. На платформі зібрано інформацію про сотні тисяч ігор з різних платформ, є зручний пошук та розгалужена система фільтрів. Користувачі можуть додавати ігри до власних списків і виставляти базові оцінки. Проте RAWG є скоріше каталогом ніж повноцінною платформою для оцінювання, система рецензій там практично відсутня, соціальна взаємодія між користувачами мінімальна, а розмежування між звичайними користувачами та критиками не передбачено [7].

Steam є найбільшою платформою для розповсюдження ігор на ПК і має вбудовану систему відгуків. Користувачі, які придбали гру, можуть залишити рецензію та позначити своє ставлення як позитивне або негативне. На основі цих даних формується загальна оцінка гри. Перевагою Steam є те, що залишати відгуки можуть лише реальні покупці, а це певною мірою захищає від зловживань. Однак платформа не є спеціалізованою для оцінювання ігор, система оцінок зведена до бінарної шкали без можливості виставити числовий бал, а критики та звичайні гравці не розрізняються жодним чином.

Проведений аналіз показує, що кожна з розглянутих платформ вирішує лише частину потреб користувачів. Агрегатори зосереджені на професійній критиці й

ігнорують спільноту, Steam обмежений бінарною системою оцінок і прив'язаний до магазину, а RAWG попри великий каталог не пропонує повноцінного досвіду оцінювання. Платформа Ludara розроблялась з урахуванням цих недоліків, вона поєднує детальну числову шкалу оцінювання, систему рецензій, розмежування ролей між звичайними користувачами та критиками, а також використовує каталог RAWG як джерело даних про ігри [7].

1.2 Постановка завдання та цілей

Метою даної кваліфікаційної роботи є проєктування та розробка веб-платформи для оцінювання відеоігор «Ludara» з використанням технології Node.js. Платформа призначена для широкої аудиторії гравців, які прагнуть зручного інструменту для оцінювання ігор, ведення власної ігрової бібліотеки та ознайомлення з думками інших користувачів і критиків.

Для досягнення поставленої мети було визначено ряд конкретних завдань, виконання яких забезпечує повноцінне функціонування платформи.

Першим завданням є реалізація системи автентифікації користувачів. Платформа має підтримувати реєстрацію нових користувачів з унікальним нікнеймом та електронною поштою, а також вхід до системи на основі токенів JWT. При цьому перегляд каталогу ігор залишається доступним без реєстрації, тоді як взаємодія з платформою: оцінювання, написання рецензій, ведення бібліотеки буде вимагати авторизації.

Другим завданням є розробка каталогу відеоігор з інтеграцією зовнішнього API. Оскільки самостійне наповнення бази даних іграми є трудомістким процесом, прийнято рішення використовувати RAWG API, як джерело даних. Каталог має підтримувати пошук за назвою, фільтрацію та сортування, а також пагінацію результатів. При першій взаємодії користувача з грою її дані автоматично зберігаються у внутрішній базі даних платформи [7].

Третім завданням є реалізація системи оцінювання та рецензування. Кожен зареєстрований користувач може виставити оцінку грі за шкалою від 1 до 10 та

написати текстову рецензію. На основі всіх оцінок формується середній бал платформи для кожної гри. Рецензії інших користувачів можна переглядати на сторінці гри, а також відзначати їх як корисні за допомогою вподобань.

Четвертим завданням є розробка системи статусів ігор. Користувач може позначити гру одним із трьох статусів: «хочу пограти», «граю» або «пройшов». Це дозволяє вести власну ігрову бібліотеку та відстежувати прогрес. Статуси відображаються на публічній сторінці профілю користувача.

П’ятим завданням є реалізація системи ролей користувачів. Платформа передбачає три ролі: звичайний користувач, критик та адміністратор. Роль критика призначається адміністратором і надає користувачу окремий блок для рецензій на сторінці гри, а також спеціальний бейдж на профілі. Адміністратор має повний доступ до керування платформою.

Шостим завданням є розробка адміністративної панелі. Через неї адміністратор може переглядати список усіх користувачів, змінювати їх ролі, накладати тимчасові або постійні бани, видаляти акаунти та рецензії, а також переглядати загальну статистику платформи.

Сьомим завданням є реалізація публічних профілів користувачів зі статистикою. Кожен користувач має власну публічну сторінку, на якій відображаються його аватар, нікнейм, роль, список ігор за статусами, виставлені оцінки та рецензії. Також підраховується статистика: середня оцінка користувача, кількість рецензій та отриманих вподобань, а також топ улюблених жанрів на основі оцінених ігор.

Восьмим завданням є забезпечення зручного та адаптивного інтерфейсу користувача. Фронтенд частина платформи розроблена з підтримкою темної та світлої теми, що дозволяє користувачам обирати комфортний для них вигляд. Інтерфейс орієнтований на простоту використання та доступність основних функцій без зайвих кроків.

Виконання усіх перелічених завдань у сукупності забезпечує розробку повноцінної веб-платформи, яка відповідає визначеній меті та задовольняє потреби цільової аудиторії.

1.3 Пошук акторів та варіантів використання

Для коректного проєктування системи необхідно визначити всіх учасників, які взаємодіють з платформою, та описати набір дій, доступних кожному з них. У контексті платформи Ludara виділено чотири основні актори: незареєстрований відвідувач, зареєстрований користувач, критик та адміністратор.

Незареєстрований відвідувач – це будь-яка особа, яка відкрила платформу без входу в систему. Такий актор має обмежений доступ до функціоналу, він може переглядати каталог ігор, здійснювати пошук та фільтрацію, переходити на сторінки окремих ігор і читати рецензії інших користувачів. Проте будь-яка активна взаємодія з платформою: оцінювання, написання рецензій, ведення бібліотеки вимагає реєстрації та входу в систему. На рисунку 1.1 зображено діаграму варіантів використання для незареєстрованого відвідувача.

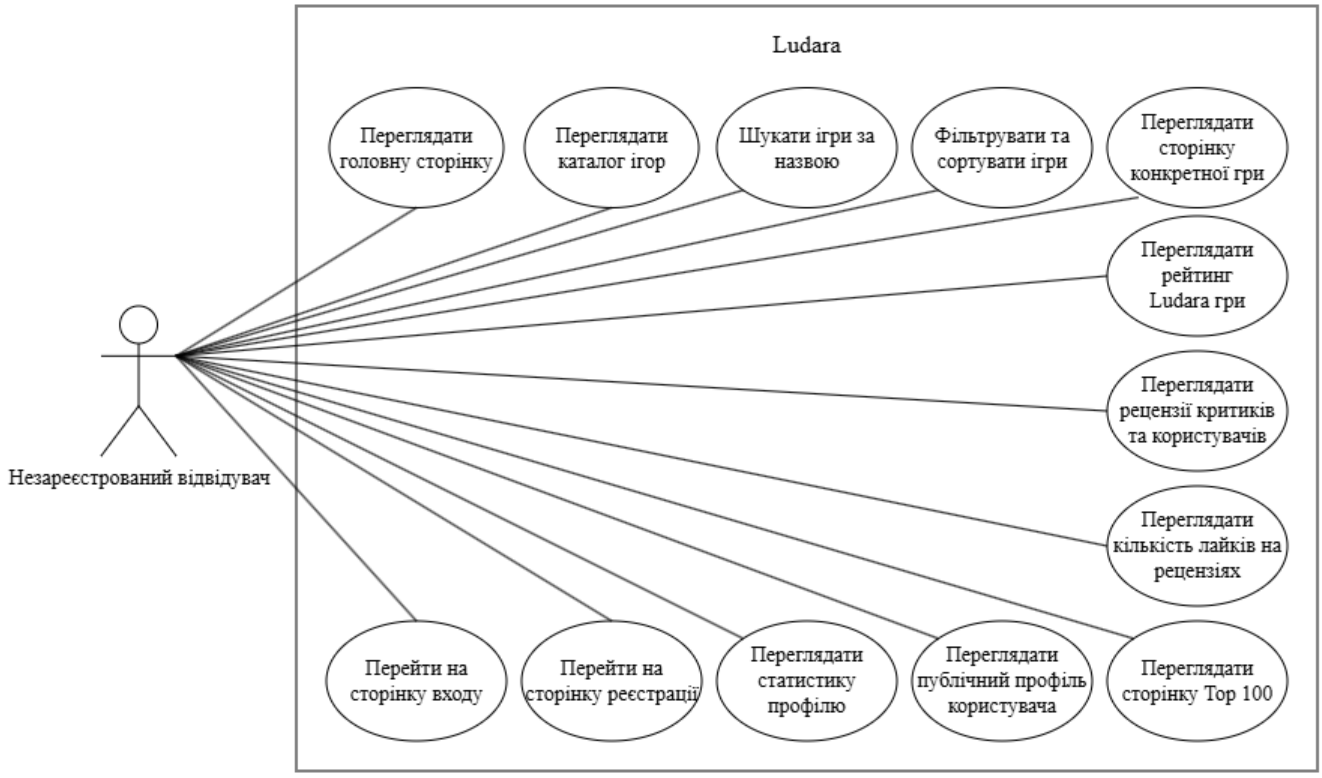


Рисунок 1.1 – Діаграма варіантів використання незареєстрованого відвідувача

Зареєстрований користувач отримує доступ до повного базового функціоналу платформи. Після входу в систему він може виставляти оцінки іграм за шкалою від 1 до 10, писати текстові рецензії, позначати ігри статусами, вподобати рецензії інших користувачів, переглядати публічні профілі та керувати власним профілем, а саме змінювати нікнейм, електронну пошту, пароль та аватар. На рисунку 1.2 зображено діаграму варіантів використання для зареєстрованого користувача.

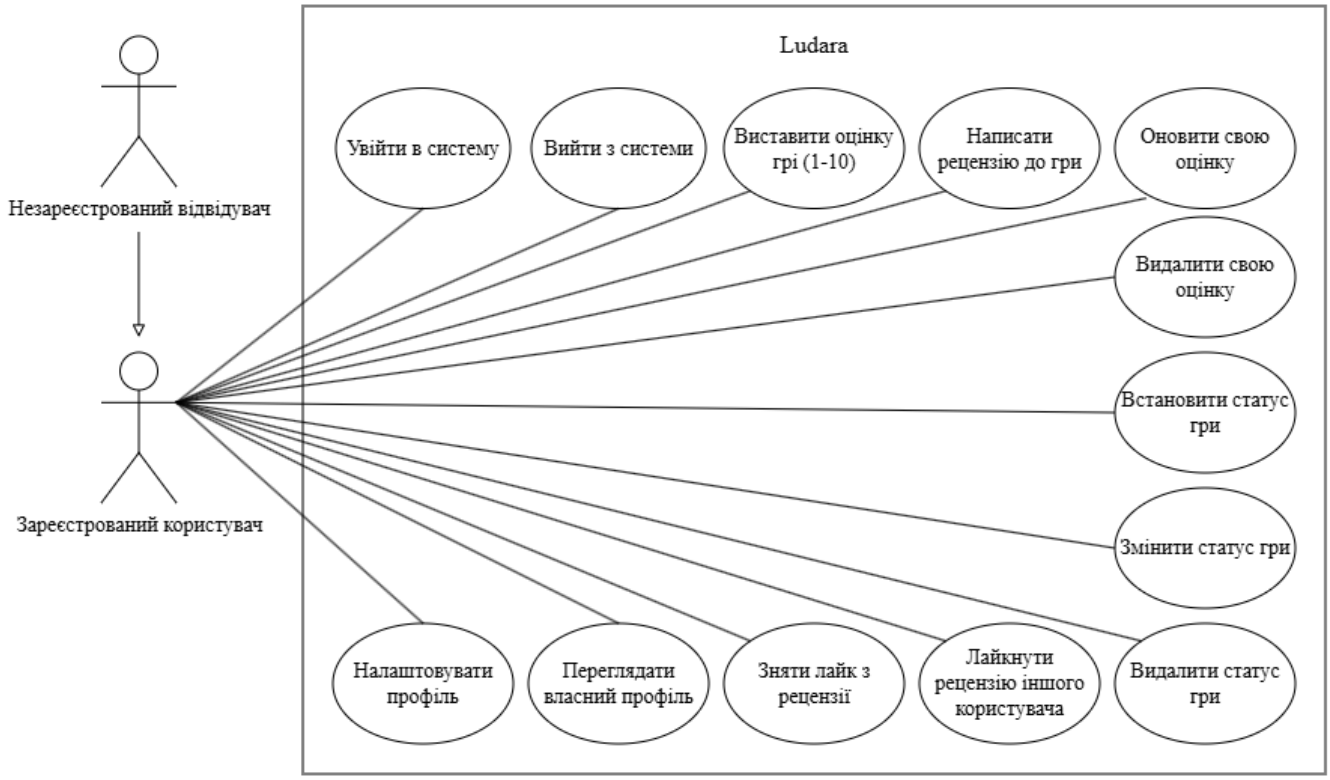


Рисунок 1.2 – Діаграма варіантів використання зареєстрованого користувача

Критик є розширеною роллю зареєстрованого користувача, яка призначається адміністратором. Критик має весь функціонал звичайного користувача, проте його рецензії відображаються в окремому блоці на сторінці гри і виділяються спеціальним бейджем на профілі. Це дозволяє чітко розмежувати думку досвідченого оглядача від звичайних користувацьких відгуків. На рисунку 1.3 зображено діаграму варіантів використання для критика.

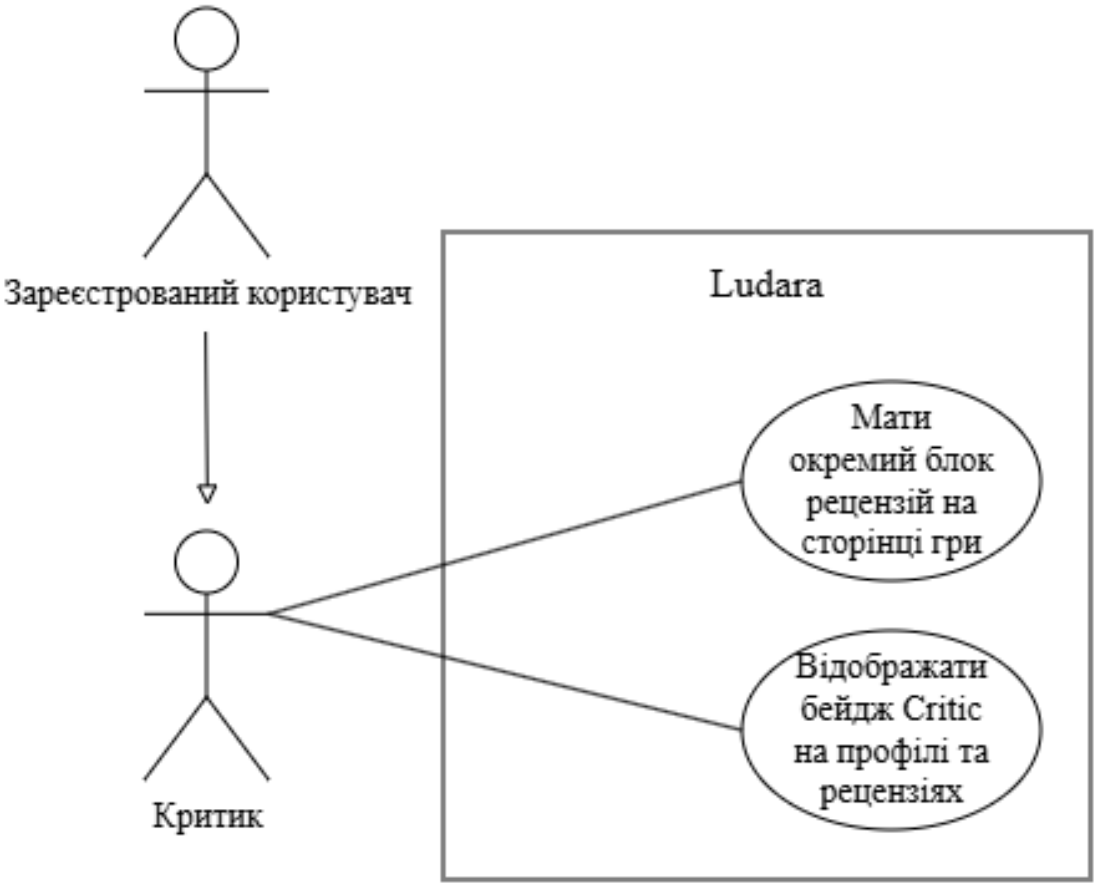


Рисунок 1.3 – Діаграма варіантів використання критика

Адміністратор має найширші повноваження на платформі. Окрім усього функціоналу звичайного користувача, адміністратор може керувати іншими користувачами через адміністративну панель, змінювати їх ролі, накладати тимчасові або постійні бани, видаляти акаунти та рецензії, а також переглядати загальну статистику платформи. На рисунку 1.4 зображено діаграму варіантів використання для адміністратора.

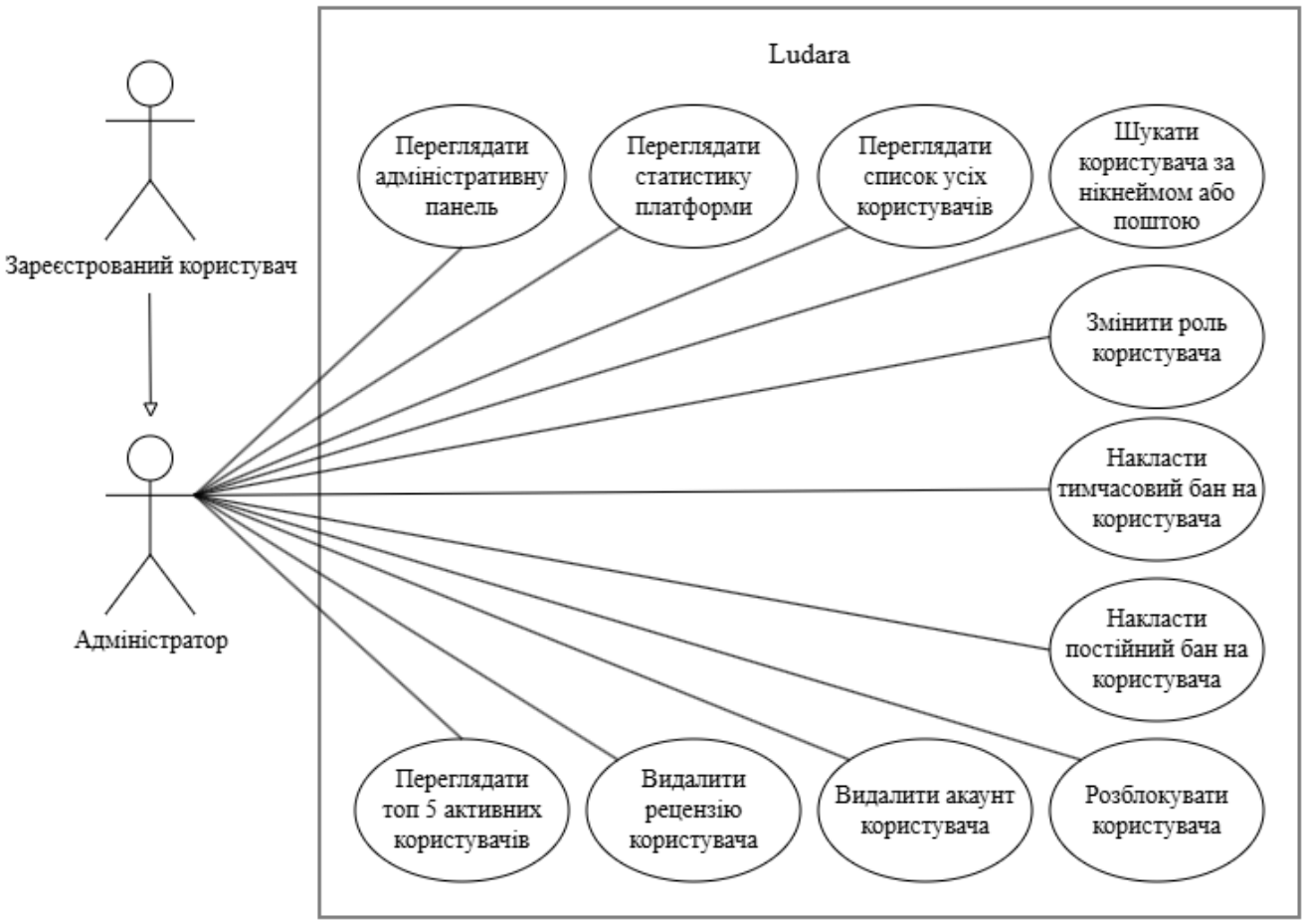


Рисунок 1.4 – Діаграма варіантів використання адміністратора

1.4 Опис ключових варіантів використання

Для розуміння взаємодії користувачів з платформою розглянуто ключові варіанти використання системи. Кожен з них описує конкретний сценарій дій актора, передумови для його виконання, основний та альтернативний потоки подій, а також очікуваний результат.

Реєстрація та автентифікація є базовим сценарієм, який відкриває користувачу доступ до повного функціоналу платформи. Саме від коректної роботи цього залежить безпека облікових записів та розмежування прав між різними ролями користувачів. У таблиця 1.1 зображено варіант використання «Реєстрація та вхід в систему».

Таблиця 1.1 – Варіант використання «Реєстрація та вхід в систему»

Поле	**Опис**
Актор	Незареєстрований відвідувач
Передумови	Користувач відкрив платформу і не має облікового запису
Основний потік	Користувач переходить на сторінку реєстрації, вводить унікальний нікнейм, електронну пошту та пароль. Система перевіряє унікальність даних, хешує пароль та створює обліковий запис. Після успішної реєстрації користувач автоматично отримує JWT-токен і отримує доступ до повного функціоналу платформи
Альтернативний потік	Якщо введений нікнейм або електронна пошта вже зареєстровані в системі, користувач отримує повідомлення про помилку і пропозицію ввести інші дані або перейти на сторінку входу
Результат	Створено новий обліковий запис, користувач авторизований у системі

Оцінювання ігор та написання рецензій є основним функціоналом платформи, навколо якого будується вся соціальна взаємодія. Цей сценарій демонструє, як користувач може висловити свою думку щодо гри та долучитися до формування загального рейтингу платформи. У таблиці 1.2 зображено варіант використання «Оцінювання гри та написання рецензії».

Таблиця 1.2 – Варіант використання «Оцінювання гри та написання рецензії»

Поле	**Опис**
Актор	Зареєстрований користувач
Передумови	Користувач авторизований і знаходиться на сторінці конкретної гри
Основний потік	Користувач обирає числову оцінку за шкалою від 1 до 10 та за бажанням додає текстову рецензію. Після підтвердження система зберігає оцінку у базі даних, оновлює середній бал гри на платформі та відображає рецензію у відповідному блоці на сторінці гри
Альтернативний потік	Якщо користувач вже оцінював цю гру раніше, система відображає його попередню оцінку і дозволяє її відредагувати. Якщо користувач не авторизований і намагається виставити оцінку, система перенаправляє його на сторінку входу
Результат	Оцінка збережена, середній бал гри оновлено, рецензія відображається на сторінці гри

Керування статусами ігор дозволяє користувачу вести власну ігрову бібліотеку та відстежувати свій прогрес. Цей сценарій є важливим з точки зору персоналізації досвіду на платформі, оскільки бібліотека відображається публічно

і дає змогу іншим користувачам бачити ігровий профіль людини. У таблиці 1.3 зображено варіант використання «Керування статусами ігор».

Таблиця 1.3 – Варіант використання «Керування статусами ігор»

Поле	**Опис**
Актор	Зареєстрований користувач
Передумови	Користувач авторизований і знаходиться на сторінці конкретної гри
Основний потік	Користувач обирає один із трьох доступних статусів — «хочу пограти», «граю» або «пройшов». Система зберігає вибраний статус і відображає гру у відповідному розділі бібліотеки на профілі користувача
Альтернативний потік	Якщо статус для цієї гри вже був встановлений раніше, система замінює його на новий. Користувач також може повністю прибрати статус якщо більше не хоче відстежувати гру
Результат	Статус гри збережено, гра відображається у бібліотеці профілю користувача у відповідній категорії

Керування користувачами через адміністративну панель забезпечує порядок і безпеку на платформі. Цей сценарій демонструє, як адміністратор може реагувати на порушення правил спільноти та підтримувати якість контенту на платформі. У таблиці 1.4 зображено варіант використання «Керування користувачем через адмін-панель».

Таблиця 1.4 – Варіант використання «Керування користувачем через адмін-панель»

Поле	**Опис**
Актор	Адміністратор
Передумови	Адміністратор авторизований і знаходиться в адміністративній панелі
Основний потік	Адміністратор здійснює пошук користувача за нікнеймом або переглядає загальний список. Після вибору конкретного користувача адміністратор може змінити його роль, накласти тимчасовий або постійний бан із зазначенням причини, видалити обліковий запис або видалити окремі рецензії користувача
Альтернативний потік	Якщо пошук не повертає жодного результату, система відображає відповідне повідомлення. У разі спроби заблокувати іншого адміністратора система відхиляє дію
Результат	Зміни застосовано до облікового запису користувача, інформація оновлена у базі даних

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

Розділ присвячено опису процесу проєктування та розробки веб-платформи Ludara. Розглядаються ключові рішення які були прийняті на етапі планування, вибір моделі розробки, архітектури системи, технологічного стеку та інструментів. Окремо описується структура бази даних, основні компоненти серверної та клієнтської частин, а також підходи до реалізації інтерфейсу користувача. При прийнятті архітектурних рішень та організації процесу розробки враховувались рекомендації та кращі практики визначені у стандарті SWEBOK [1].

2.1 Вибір процесу розробки

Вибір моделі розробки програмного забезпечення є одним з перших рішень на етапі планування проєкту. Від обраного підходу залежить порядок виконання робіт, спосіб організації коду та загальна передбачуваність результату. Для розробки платформи Ludara було обрано ітеративну ґмодель розробки, яка найкраще відповідає характеру та масштабу проєкту.

Ітеративна модель передбачає розбиття всього обсягу роботи на окремі завершені частини ітерації. Кожна ітерація є самостійним функціональним блоком, який розробляється, тестується і вводиться в експлуатацію послідовно. Результат кожної попередньої ітерації стає основою для наступної, що дозволяє поступово нарощувати функціонал системи не порушуючи вже реалізовані частини.

Від водоспадної моделі ітеративний підхід відрізняється тим що не вимагає повного завершення одного етапу перед початком наступного в рамках всього проєкту, натомість кожен модуль проходить повний цикл розробки від проєктування до тестування окремо. Порівняно з Agile та Scrum ітеративна модель є простішою в організації і не передбачає роботи в команді зі спринтами, щоденними зустрічами та розподілом ролей, що робить її оптимальним вибором для індивідуальної розробки.

Процес розробки платформи Ludara був організований наступним чином. На початковому етапі було визначено повний перелік функціональних модулів, які має містити платформа. Після цього розробка велась послідовно, спочатку повністю реалізовувалась серверна частина, а потім клієнтська. У рамках серверної частини кожен модуль розроблявся окремо: спочатку система автентифікації, потім модуль каталогу ігор з інтеграцією RAWG API, далі модуль оцінювання та рецензування, модуль статусів ігор, система вподобань, профіль користувача зі статистикою та адміністративний модуль. Після завершення серверної частини аналогічним чином послідовно розроблявся фронтенд сторінка за сторінкою, без змішування різних частин інтерфейсу [7].

2.2 Проєктування архітектури системи

Архітектура платформи Ludara побудована за клієнт-серверною моделлю з чітким розділенням на три незалежні шари: клієнтська частина, серверна частина та шар даних [16]. Така організація забезпечує незалежність компонентів системи, кожен шар виконує свою роль і не знає про внутрішню реалізацію інших [16]. Загальну схему архітектури системи наведено на рисунку 2.1.

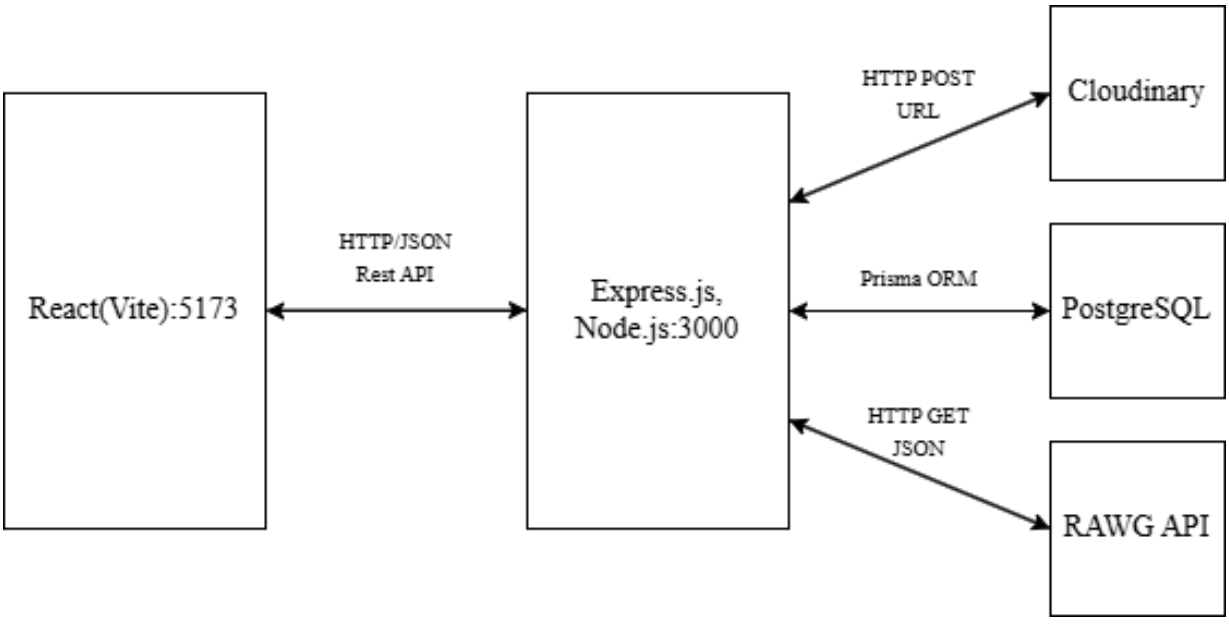


Рисунок 2.1 – Загальна схема архітектури системи

Клієнтська частина є односторінковим застосунком розробленим на React 19 з використанням інструменту збірки Vite. Вона працює у браузері користувача і взаємодіє з сервером виключно через HTTP запити у форматі JSON [10]. Фронтенд не має жодного прямого доступу до бази даних чи зовнішніх сервісів, вся комунікація з зовнішнім світом відбувається через бекенд. Маршрутизація між сторінками реалізована на стороні клієнта за допомогою React Router 7, що забезпечує плавну навігацію без повного перезавантаження сторінки [11].

Серверна частина реалізована на Node.js з використанням фреймворку Express.js 5 і працює на порту 3000. Вона відповідає за всю бізнес-логіку платформи, автентифікацію користувачів, обробку запитів, валідацію даних, взаємодію з базою даних та інтеграцію із зовнішніми сервісами. Комунікація між клієнтом і сервером відбувається за принципами REST API, де кожен ресурс має свій URL, а дії над ним визначаються HTTP методами [15]. Для запобігання несанкціонованим запитам з інших доменів на сервері налаштовано CORS, який дозволяє запити виключно з адреси клієнтської частини [2,3].

Шар даних складається з бази даних PostgreSQL та двох зовнішніх сервісів. База даних містить всі внутрішні дані платформи: користувачів, оцінки, рецензії, статуси ігор та інформацію про заблокованих користувачів. Взаємодія з базою даних відбувається виключно через Prisma ORM, який забезпечує типізований доступ до даних та автоматично генерує оптимізовані SQL запити [5,18]. RAWG API використовується, як зовнішнє джерело даних про ігри, каталог, пошук, інформація про конкретну гру [7]. Cloudinary використовується для зберігання аватарів користувачів [8].

Бекенд дотримується патерну Routes-Controller-Service, який є адаптацією архітектурного патерну MVC без явного шару представлення, оскільки відповіді сервера є JSON об’єктами а не HTML сторінками [17]. Шар маршрутів відповідає виключно за прив'язку HTTP методів і URL шляхів до відповідних контролерів та визначає які middleware необхідно виконати перед обробкою запиту. Шар контролерів є центральним місцем бізнес-логіки, тут відбувається валідація вхідних даних через бібліотеку Zod, звернення до бази даних через Prisma та

формування відповіді [5,14]. Шар сервісів інкапсулює інтеграцію із зовнішніми системами RAWG API та Cloudinary і надає контролерам простий інтерфейс для роботи з ними без необхідності знати деталі зовнішніх API [7,8].

Архітектурний патерн Routes-Controller-Service є адаптацією класичного патерну MVC для REST API застосунків, де відсутній традиційний шар представлення. У класичному MVC модель відповідає за дані, представлення за відображення а контролер за логіку взаємодії між ними. У випадку платформи Ludara шар представлення замінено форматом JSON відповідей, що повертаються клієнту, шар моделі реалізовано через Prisma ORM та схему бази даних, а контролери зосереджують всю бізнес-логіку обробки запитів. Такий підхід дозволяє розмежувати відповідальність між компонентами системи та забезпечує незалежність кожного шару від деталей реалізації інших [17].

Для управління єдиним з’єднанням з базою даних застосовано патерн Singleton. Інстанс PrismaClient створюється один раз у файлі prisma.js і експортується, як спільний об’єкт для всіх контролерів системи. Такий підхід є правильним з точки зору управління ресурсами оскільки Prisma Client внутрішньо управляє пулом з’єднань до PostgreSQL, створення окремого інстансу для кожного запиту призвело б до надмірного споживання ресурсів та деградації продуктивності [5]. Обробка вхідних HTTP запитів реалізована через патерн Middleware Chain. Ланцюжок послідовно виконуваних функцій, де кожна виконує свою роль і передає управління наступній через виклик next(). Цей патерн забезпечує модульність та повторне використання логіки автентифікації та авторизації незалежно від конкретного маршруту.

Для вирішення проблеми синхронізації між зовнішнім каталогом RAWG та внутрішньою базою даних застосовано патерн Lazy Loading через функцію ensureGameExists. Замість попереднього імпорту всього каталогу ігор запис у таблиці Game створюється лише в момент першої взаємодії користувача з конкретною грою. При кожному наступному зверненні функція знаходить вже існуючий запис без додаткових запитів до зовнішнього API. Такий підхід мінімізує обсяг даних що зберігаються локально та знижує кількість запитів до зовнішнього

сервісу [7]. Взаємодія фронтенду з RAWG API реалізована через патерн Proxy. Всі запити до зовнішнього сервісу виконуються виключно на стороні сервера через rawg.service.js. Це забезпечує приховування API ключа від клієнта, централізовану фільтрацію небажаного контенту та можливість кешування відповідей у майбутньому.

Для операцій оцінювання ігор та встановлення статусів застосовано патерн Upsert. Операція, яка автоматично створює новий запис якщо він відсутній або оновлює існуючий якщо він вже є. Це дозволяє уникнути перевірки наявності запису перед кожною операцією збереження та гарантує, що один користувач може мати лише одну оцінку та один статус для кожної гри. Система вподобань рецензій реалізована через патерн Toggle. При кожному запиті перевіряється поточний стан вподобання і залежно від результату воно або додається або видаляється. Такий підхід забезпечує інтуїтивну взаємодію де повторний запит скасовує попередню дію без необхідності мати окремі ендпоінти для додавання та видалення вподобання.

Сукупність застосованих архітектурних рішень та патернів проєктування забезпечує платформі Ludara чітку структуру коду, передбачувану поведінку системи та можливість незалежного розвитку окремих компонентів. Кожен патерн вирішує конкретну проблему, Singleton керує ресурсами, Middleware Chain забезпечує модульність обробки запитів, Lazy Loading оптимізує взаємодію з зовнішнім API, Proxy захищає конфіденційні дані, а Upsert та Toggle спрощують бізнес-логіку роботи з даними [16,17].

Окремої уваги заслуговує механізм обробки HTTP запиту, детальну схему якого наведено у додатку А. Коли клієнт надсилає запит, Axios interceptor автоматично додає до нього JWT токен з localStorage у вигляді заголовку Authorization [9,12]. На сервері запит проходить через глобальні middleware перевірку CORS та парсинг JSON тіла, після чого передається до відповідного роутера. Якщо маршрут захищений, перед контролером виконується middleware автентифікації, який перевіряє токен і заповнює об’єкт запиту даними користувача. Якщо маршрут вимагає адміністративних прав, додатково виконується перевірка

ролі. Після проходження всіх middleware запит потрапляє до контролера, де відбувається вся бізнес-логіка, і сервер повертає відповідь у форматі JSON.

Така архітектура забезпечує розмежування відповідальності між компонентами системи, спрощує підтримку коду та дозволяє незалежно розвивати клієнтську і серверну частини платформи.

2.3 Побудова схем бази даних

База даних платформи Ludara реалізована на основі PostgreSQL і містить шість таблиць, які забезпечують зберігання всіх даних системи [6]. Для роботи з базою даних використовується Prisma ORM, який на основі декларативної схеми автоматично генерує типізований клієнт та SQL міграції [5]. Перед визначенням таблиць у схемі оголошено два enum типи. Тип Role визначає можливі ролі користувачів у системі, USER для звичайних користувачів, CRITIC для критиків та ADMIN для адміністраторів. Тип Status визначає можливі стани гри в бібліотеці користувача, WANT для ігор які користувач хоче пройти, PLAYING для ігор у процесі проходження та COMPLETED для завершених ігор. Повну схему бази даних з усіма таблицями та зв’язками між ними наведено на рисунку 2.2.

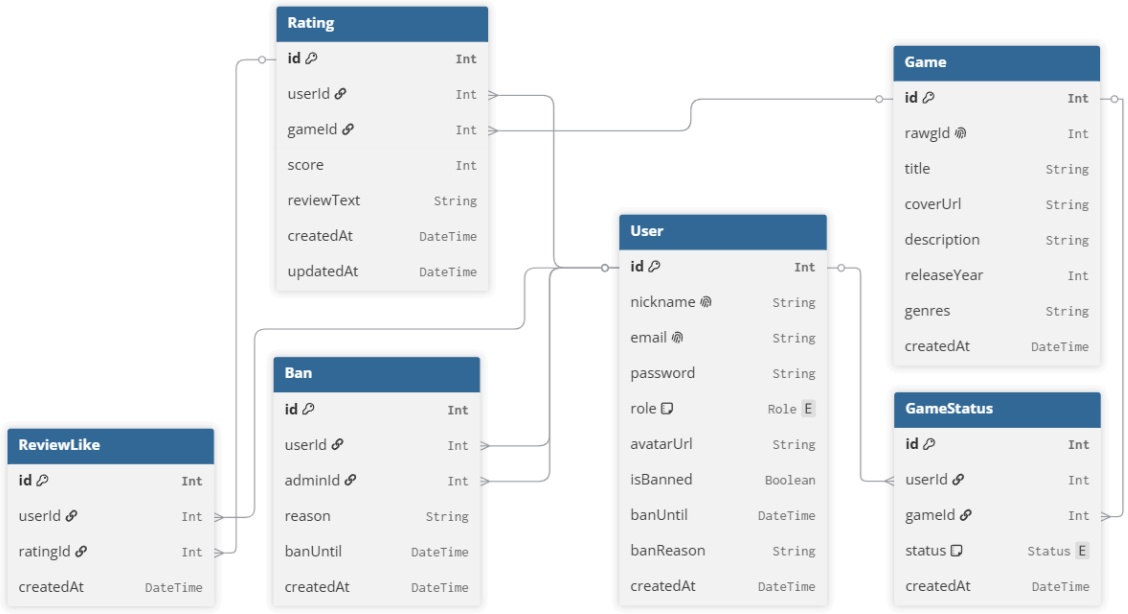


Рисунок 2.2 – Схема бази даних платформи Ludara

Таблиця «User» є центральною таблицею системи і зберігає облікові записи всіх зареєстрованих користувачів. Вона містить унікальний нікнейм та електронну пошту, bcrypt хеш пароля, роль користувача з типом Role та посилання на аватар, що зберігається на Cloudinary [20]. Окремої уваги потребують поля системи банів, isBanned визначає поточний статус блокування, banUntil зберігає дату закінчення тимчасового бану і при значенні NULL означає постійне блокування, а banReason містить причину блокування.

Таблиця «Game» є локальним кешем метаданих ігор отриманих з RAWG API. Вона не заповнюється наперед, запис створюється лише в момент першої взаємодії користувача з грою через механізм відкладеного збереження. Таблиця зберігає мінімальний набір полів необхідних для роботи платформи зовнішній ідентифікатор rawgId, який є точкою синхронізації з RAWG, назву гри, посилання на обкладинку, опис, рік виходу та жанри у вигляді текстового рядка [7]. Поле rawgId має унікальний індекс оскільки використовується при кожній взаємодії користувача з грою для перевірки її наявності у внутрішній базі даних.

Таблиця «Rating» є центральною таблицею бізнес-логіки платформи і зберігає оцінки та рецензії користувачів. Кожен запис пов’язаний з конкретним користувачем і конкретною грою через зовнішні ключі. Унікальна комбінація полів userId та gameId гарантує що один користувач може мати лише одну оцінку для однієї гри, при повторному оцінюванні виконується операція upsert яка оновлює існуючий запис замість створення нового. Числова оцінка зберігається у діапазоні від 1 до 10, а текстова рецензія є необов’язковим полем.

Таблиця «GameStatus» реалізує ігрову бібліотеку користувача і зберігає статус кожної гри. Як і в таблиці Rating, унікальна комбінація userId та gameId забезпечує наявність лише одного статусу для кожної гри в бібліотеці конкретного користувача. Зміна статусу також реалізована через upsert.

Таблиця «ReviewLike» реалізує можливість вподобати рецензії інших користувачів і є junction таблицею, що реалізує зв’язок багато до багатьох між таблицями User та Rating. Такий підхід з явною junction таблицею обраний тому що запис вподобань несе власні атрибути, зокрема дату створення і потребує окремої

перевірки унікальності. Унікальна комбінація userId та ratingId гарантує, що один користувач може вподобати одну рецензію лише один раз.

Таблиця «Ban» є аудиторським журналом усіх адміністративних рішень щодо блокування користувачів. На відміну від полів isBanned та banUntil у таблиці User, які відображають поточний стан блокування, таблиця Ban зберігає повну історію всіх накладених банів незалежно від їх поточного статусу. Кожна адміністративна дія створює новий запис у цій таблиці навіть якщо користувач блокується повторно. Таблиця має самореференційний зв’язок з таблицею User через поле userId посилається на заблокованого користувача, а поле adminId на адміністратора який прийняв рішення про блокування.

Усі зовнішні ключі в схемі налаштовані з каскадним видаленням. При видаленні користувача автоматично видаляються всі його оцінки, рецензії, статуси ігор, вподобання та записи про бани. При видаленні рецензії автоматично видаляються всі вподобання до неї. Такий підхід забезпечує цілісність даних без необхідності реалізовувати логіку каскадного видалення на рівні застосунку.

Керування змінами схеми бази даних здійснюється через механізм міграцій Prisma Migrate. При кожній зміні схеми автоматично генерується SQL файл міграції який фіксує всі зміни і зберігається у системі контролю версій разом з кодом проєкту. Це забезпечує відтворюваність структури бази даних на будь-якому середовищі та дозволяє відстежувати історію змін схеми [5].

2.4 Побудова UML-діаграм модулів

Для повного розуміння внутрішньої структури серверної частини платформи необхідно розглянути організацію основних модулів системи: middleware, контролерів та сервісів, їх відповідальність, методи та взаємодію між собою. Хоча бекенд платформи Ludara реалізований у парадигмі модульного програмування на Node.js без використання класів у класичному об'єктно-орієнтованому розумінні, структуру системи можна представити у вигляді UML діаграми модулів що

відображає залежності та відповідальність кожного компонента. Діаграму модулів серверної частини наведено на рисунку 2.3 [2].

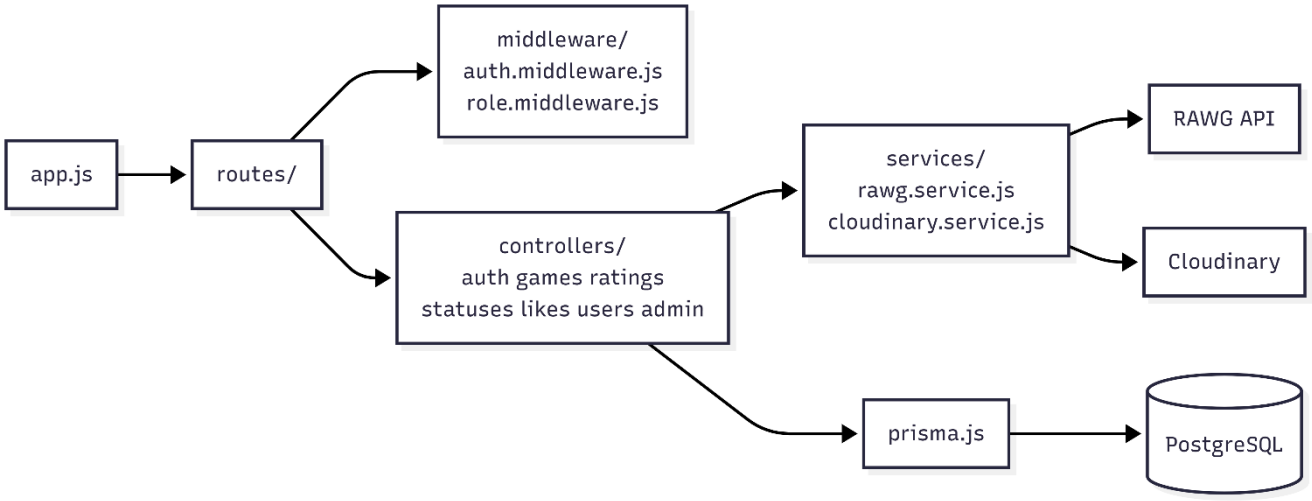


Рисунок 2.3 – Діаграма модулів серверної частини платформи Ludara**.**

Шар middleware є першою точкою обробки захищених запитів і відповідає за наскрізну логіку автентифікації та авторизації. Файл auth.middleware.js містить дві функції з різною поведінкою. Функція authenticate є обов’язковою перевіркою токена і використовується на всіх маршрутах, де наявність авторизованого користувача є обов’язковою умовою. Її алгоритм передбачає зчитування заголовку Authorization, перевірку наявності префіксу Bearer, верифікацію JWT токена за допомогою секретного ключа та запис декодованого payload у об’єкт запиту у вигляді req.user з полями id, nickname та role [9]. У разі відсутності токена або його невалідності функція завершує обробку запиту з кодом 401. Функція optionalAuthenticate працює за схожим принципом але ніколи не завершує запит з помилкою, якщо токен відсутній або невалідний вона просто продовжує обробку без заповнення req.user. Така поведінка використовується на публічних маршрутах, де відповідь може відрізнятися залежно від того авторизований користувач чи ні, наприклад при перевірці чи поточний користувач вже відзначив рецензію.

Файл role.middleware.js містить функцію requireAdmin, яка завжди використовується після authenticate оскільки потребує вже заповненого req.user.

Функція перевіряє роль користувача і у разі якщо вона не є ADMIN завершує запит з кодом 403. У адміністративному роутері ця функція застосовується одразу до всього роутера через router.use, що означає автоматичну перевірку ролі для кожного маршруту без необхідності вказувати її окремо [11].

Окремої уваги заслуговує inline middleware для завантаження аватарів що вбудований безпосередньо у визначення маршруту в users.routes.js. Таке рішення обумовлено особливістю бібліотеки multer, помилки обробки файлів виникають всередині неї і не потрапляють у стандартний обробник помилок Express [21]. Тому обробка помилок multer реалізована через callback патерн безпосередньо у визначенні маршруту перевищення розміру файлу понад 2 МБ повертає код 400 з відповідним повідомленням, як і передача файлу з невірним MIME типом [3,21].

Шар контролерів є центральним місцем бізнес-логіки платформи. Контролер auth.controller.js відповідає за три операції: реєстрацію, вхід та отримання профілю поточного користувача [17]. При реєстрації виконується валідація вхідних даних через Zod схему registerSchema, перевірка унікальності нікнейму та електронної пошти, хешування пароля з використанням bcrypt з десятьма раундами солі та створення запису в базі даних [14,19]. При успішній реєстрації повертається JWT токен з кодом 201. При вході в систему окрім стандартної перевірки credentials реалізована логіка обробки банів, якщо користувач заблокований перевіряється тип бану і його актуальність. Якщо тимчасовий бан вже минув він автоматично знімається в базі даних і користувач отримує доступ. Якщо бан активний повертається код 403 з інформацією про причину та тривалість блокування.

Контролер games.controller.js виконує роль проксі між фронтендом і RAWG API та забезпечує роботу з внутрішніми рейтингами [17]. Метод getGames передає параметри запиту безпосередньо до сервісу rawg.service.js який виконує фільтрацію та пагінацію. Метод getGameById отримує повні дані гри з RAWG і доповнює їх середньою оцінкою Ludara та кількістю оцінок якщо гра вже є у внутрішній базі даних [7]. Метод getTopGames реалізує агрегацію через Prisma, групування оцінок за ідентифікатором гри з підрахунком середнього балу та сортуванням спочатку за середньою оцінкою, а потім за кількістю оцінок як вторинний критерій. Метод

getBulkRatings використовується після завантаження сторінки каталогу і дозволяє за один запит отримати рейтинги для всіх ігор на сторінці що значно зменшує кількість запитів до бази даних [5].

Контролер ratings.controller.js керує оцінками та рецензіями користувачів. Центральним елементом є функція ensureGameExists, яка реалізує патерн відкладеного збереження, перед кожним записом оцінки або статусу перевіряється наявність гри у внутрішній базі даних і якщо вона відсутня дані отримуються з RAWG API та зберігаються локально [7]. Метод rateGame використовує операцію upsert, що дозволяє як створювати нову оцінку так і оновлювати існуючу одним запитом, це гарантує, що один користувач може мати лише одну оцінку для однієї гри. Метод getGameRatings повертає всі рецензії для конкретної гри з даними авторів та кількістю вподобань, при цьому якщо гра відсутня у внутрішній базі даних повертається порожній масив що є коректною поведінкою, а не помилкою.

Контролер statuses.controller.js реалізує ігрову бібліотеку користувача. Логіка встановлення статусу ідентична логіці оцінювання, використовується той самий патерн ensureGameExists та upsert. Метод getUserStatuses повертає бібліотеку згрупованою за трьома статусами для відображення на сторінці профілю [17].

Контролер likes.controller.js реалізує систему вподобань через патерн toggle, при кожному запиті перевіряється чи існує вподобання від поточного користувача на дану рецензію і залежно від результату він або створюється або видаляється. Після зміни стану вподобань підраховується актуальна кількість вподобань і повертається разом зі статусом дії. Метод getLikes використовує optionalAuthenticate, що дозволяє неавторизованим користувачам бачити кількість вподобань без інформації про власний стан.

Контролер users.controller.js відповідає за публічні профілі та налаштування облікових записів. Метод getStats виконує п’ять паралельних запитів до бази даних через Promise.all, що мінімізує час очікування, підраховуються кількість ігор за кожним статусом, середня оцінка, кількість рецензій, загальна кількість отриманих вподобань та визначаються топ три улюблені жанри на основі аналізу оцінених ігор. Аналіз жанрів виконується в пам’яті, рядок жанрів кожної гри розбивається за

комою і для кожного жанру рахується частота появи. Метод updateProfile реалізує крос-полеву валідацію через Zod refine, щоб запит не був порожнім обов’язково має бути передане хоча б одне поле [14]. При оновленні нікнейму або електронної пошти перевіряється їх унікальність з виключенням поточного користувача.

Контролер admin.controller.js надає адміністративні функції і є найбільш захищеним модулем системи. Метод getUsers реалізує автоматичне скасування протермінованих тимчасових банів, при отриманні списку користувачів перевіряються всі заблоковані акаунти і якщо термін бану минув їх статус оновлюється в базі даних масовою операцією updateMany. Метод banUser використовує транзакцію Prisma, що гарантує атомарність операції або одночасно оновлюється статус користувача і створюється запис в журналі банів або жодна з операцій не виконується [5]. Метод updateUserRole захищений від підвищення будь-якого користувача до ролі ADMIN через API, така роль навмисно виключена зі схеми валідації. Додатково реалізований захист адміністраторів від адміністративних дій з боку інших адміністраторів є спроба заблокувати або змінити роль користувача з роллю ADMIN повертає код 403.

Шар сервісів інкапсулює взаємодію із зовнішніми системами. Сервіс rawg.service.js є абстракцією над RAWG REST API і містить внутрішню функцію rawgFetch для виконання HTTP запитів та функцію filterGames для фільтрації небажаного контенту [15]. Фільтрація видаляє NSFW контент за назвою та тегами, моди та неофіційні матеріали за ключовими словами, DLC та доповнення за наявністю поля parent_game, а також ігри без жанрів. Метод getGames реалізує рекурсивне заповнення результатів, якщо після фільтрації сторінка виявляється неповною автоматично завантажується наступна сторінка RAWG до набору потрібної кількості результатів [7]. Сервіс cloudinary.service.js конфігурує multer для прямого завантаження зображень на Cloudinary без збереження на диску сервера [8,21]. При завантаженні аватара зображення автоматично конвертується у формат JPEG та кадрується до квадрату розміром 200 на 200 пікселів на стороні Cloudinary. Результатом завантаження є публічний HTTPS URL який зберігається безпосередньо в базі даних як значення поля avatarUrl.

Валідація вхідних даних реалізована через бібліотеку Zod у п’яти з семи контролерів. Загальний патерн використання передбачає виклик методу safeParse на початку кожного контролера, цей метод не кидає виключень, а повертає об’єкт з полем success.

У разі невдалої валідації повертається код 400 з повідомленням першої знайденої помилки [14]. Система HTTP статусів є уніфікованою, код 400 використовується для помилок валідації та невалідних параметрів, 401 для відсутнього або невалідного токена, 403 для недостатніх прав доступу, 404 для відсутніх ресурсів, 409 для конфліктів унікальності, 502 для помилок зовнішнього RAWG API та 500 для неочікуваних серверних помилок [7].

2.5 Вибір мови та середовища розробки

Вибір технологічного стеку є одним з ключових рішень при розробці будь-якого програмного продукту. Від обраних технологій залежить продуктивність системи, зручність розробки, масштабованість та простота підтримки коду в майбутньому. При розробці платформи Ludara перевага надавалась широко підтримуваним технологіям з активною спільнотою та детальною документацією. Технологічний стек поділяється на три групи: інструменти серверної частини, інструменти клієнтської частини та засоби організації процесу розробки.

Node.js є середовищем виконання JavaScript на стороні сервера і став основою серверної частини платформи. Головною перевагою Node.js є асинхронна неблокуюча модель виконання коду, що робить його особливо ефективним для веб-застосунків з великою кількістю одночасних з’єднань та операцій введення-виведення. Важливою перевагою є також використання JavaScript як єдиної мови програмування для обох частин застосунку, це спрощує розробку і дозволяє використовувати спільні підходи та патерни на бекенді і фронтенді. Node.js має одну з найбільших кількість пакетів у світі через менеджер пакетів npm що забезпечує доступ до готових рішень для будь-яких задач [2].

Express.js версії 5.2.1 є фреймворком для побудови веб-застосунків і REST API на Node.js. Він надає зручний інтерфейс для визначення маршрутів, підключення middleware та обробки HTTP запитів і відповідей [15]. На відміну від більш опінійованих фреймворків Express не нав'язує конкретну структуру проєкту що дозволило організувати код відповідно до обраного патерну Routes-Controller-Service. Express є одним з найпопулярніших фреймворків для Node.js з великою кількістю документації та прикладів, що робить його достатнім вибором для побудови серверної частини платформи [3,17].

PostgreSQL це об’єктно-реляційною системою керування базами даних з відкритим вихідним кодом [6]. Вона обрана, як основне сховище даних платформи завдяки надійності, підтримці складних запитів та агрегацій, строгій типізації даних та підтримці транзакцій, що забезпечують атомарність критичних операцій. PostgreSQL чудово справляється з реляційними даними, де між таблицями існують складні зв’язки, саме така структура притаманна платформі Ludara, де користувачі, ігри, оцінки, статуси та вподобання пов’язані між собою через зовнішні ключі. Підтримка enum типів дозволила типізувати ролі користувачів та статуси ігор на рівні бази даних [6].

Prisma версії 5.22.0 є ORM інструментом для Node.js, що складається з трьох компонентів: Prisma Schema для декларативного опису структури бази даних, Prisma Client для типізованого доступу до даних та Prisma Migrate для управління міграціями [18]. Головною перевагою Prisma є автоматична генерація типізованого клієнта на основі схеми що унеможливлює цілий клас помилок пов’язаних з невірними назвами полів або типами даних. Prisma надає зручний API для виконання складних запитів, включно з агрегаціями, групуванням, вкладеними запитами та операцією upsert без необхідності писати SQL вручну. Механізм міграцій дозволяє відстежувати зміни схеми бази даних у системі контролю версій і відтворювати структуру бази даних на будь-якому середовищі [5].

JSON Web Token реалізований через бібліотеку jsonwebtoken версії 9.0.3 і використовується для автентифікації користувачів. JWT є відкритим стандартом для передачі інформації між сторонами у вигляді підписаного JSON об’єкта [9].

Перевагою JWT є stateless підхід, сервер не зберігає сесії і не потребує звернення до бази даних при кожному запиті для перевірки автентифікації [9]. Токен містить закодований payload з ідентифікатором користувача, його нікнеймом та роллю, підписаний секретним ключем за алгоритмом HMAC-SHA256. Термін дії токена складає сім діб після чого користувач має повторно пройти автентифікацію [9].

Бібліотека bcrypt версії 6.0.0 використовується для безпечного хешування паролів користувачів [20]. bcrypt є адаптивною функцією хешування що навмисно спроєктована як обчислювально затратна, це унеможливлює атаки перебором навіть у випадку витоку бази даних [19]. При реєстрації пароль хешується з десятьма раундами солі перед збереженням у базі даних. При вході в систему введений пароль порівнюється з хешем через функцію compare, яка внутрішньо відтворює хеш з тією самою сіллю і порівнює результати. Оригінальний пароль ніколи не зберігається і не передається у відповідях сервера.

Zod версії 4.4.3 є бібліотекою для декларативної валідації даних з підтримкою TypeScript. Вона використовується для валідації всіх вхідних даних, що надходять від клієнта: реєстрація, вхід, виставлення оцінки, встановлення статусу, оновлення профілю та адміністративні дії. Кожна схема валідації визначає очікуваний тип, обмеження та повідомлення про помилки для кожного поля. Метод safeParse не кидає виключень, а повертає об’єкт з результатом, що дозволяє коректно обробляти помилки валідації і повертати зрозумілі повідомлення клієнту. Zod також підтримує крос-полеву валідацію через метод refine, що використовується при оновленні профілю для перевірки що хоча б одне поле передане у запиті [14].

Multer версії 2.1.1 є middleware для обробки multipart/form-data запитів, що використовується при завантаженні аватарів користувачів [21]. Він інтегрований з Cloudinary через пакет multer-storage-cloudinary версії 4.0.0, що дозволяє завантажувати файли напряму на Cloudinary без проміжного збереження на диску сервера [8]. При завантаженні перевіряється MIME тип файлу та його розмір, максимально допустимий розмір складає два мегабайти.

Cloudinary є хмарним сервісом для зберігання та обробки медіафайлів. SDK версії 1.41.3 використовується для налаштування сховища та конфігурації

трансформацій зображень. При завантаженні аватара Cloudinary автоматично конвертує зображення у формат JPEG та кадрує його до квадрату розміром 200 на 200 пікселів незалежно від оригінального розміру та пропорцій файлу. Усі аватари зберігаються в окремій папці ludara/avatars на Cloudinary а результатом завантаження є публічний HTTPS URL який зберігається в базі даних [8].

Бібліотека express-rate-limit версії 8.5.1 використовується для обмеження кількості запитів до сервера з однієї IP адреси за певний проміжок часу. Це захищає платформу від атак перебором та зловживання API. Бібліотека dotenv версії 17.4.2 забезпечує завантаження змінних середовища з файлу .env що дозволяє зберігати секретні ключі та конфігурацію окремо від коду. Nodemon версії 3.1.14 використовується під час розробки для автоматичного перезапуску сервера при зміні файлів що значно прискорює процес розробки [3].

RAWG є великою базою даних відеоігор у світі, яка надає безкоштовний REST API для доступу до інформації про ігри. API містить дані про понад п’ятсот тисяч ігор включно з назвами, обкладинками, описами, жанрами, датами виходу та платформами [15]. Для використання API необхідно зареєструватися на сайті rawg.io та отримати унікальний API ключ який передається як параметр кожного запиту. Безкоштовний план надає достатній ліміт запитів для навчального проєкту. На платформі Ludara RAWG API використовується для отримання каталогу ігор з підтримкою пошуку фільтрації та пагінації, а також для отримання детальної інформації про конкретну гру. Усі запити до RAWG виконуються виключно на стороні сервера через сервіс rawg.service.js що забезпечує приховування API ключа від клієнта та централізовану фільтрацію небажаного контенту [7].

Для використання RAWG API необхідно зареєструватись на сайті rawg.io та отримати унікальний API ключ. Безкоштовний план надає достатню кількість запитів для роботи навчального проєкту, ліміт оновлюється щомісяця. API ключ є унікальним ідентифікатором, що передається, як параметр кожного запиту до RAWG API і зберігається у файлі змінних середовища .env серверної частини платформи щоб унеможливити його потрапляння у відкритий код репозиторію.

Інформацію про налаштування облікового запису RAWG API наведено у таблиці 2.1.

Таблиця 2.1 – Інформаційна панель RAWG API

Параметр	**Значення**
План	Free
Кількість доступних запитів на місяць	18 958
Дата оновлення	09.06.2026
Ім’я користувача	Danyil
Пошта користувача	d**********@gmail.com
Site/App URL	http://localhost:3000
Опис	Платформа для оцінювання ігор
API Key	b3c9fb7**********ccbd

React версії 19.2.5 є бібліотекою для побудови користувацьких інтерфейсів на основі компонентного підходу. Кожен елемент інтерфейсу є окремим компонентом з власним станом та логікою, що забезпечує повторне використання коду та зручність підтримки. React використовує віртуальний DOM для ефективного оновлення інтерфейсу, при зміні стану оновлюються лише ті частини сторінки які дійсно змінились. Фронтенд платформи Ludara є односторінковим застосунком де весь інтерфейс будується на React компонентах [4].

Vite версії 8.0.10 є інструментом збірки для фронтенд проєктів, що прийшов на заміну застарілим рішенням, як Create React App. Головною перевагою Vite є надзвичайно швидкий запуск сервера розробки завдяки використанню нативних ES модулів браузера замість попередньої збірки всього коду. Час гарячого оновлення при зміні файлів є практично миттєвим що значно підвищує комфорт розробки. Для продакшн збірки Vite використовує Rollup, що генерує оптимізовані та мінімізовані файли [10].

React Router версії 7.15.0 забезпечує маршрутизацію на стороні клієнта в односторінковому застосунку. Він дозволяє визначати маршрути та відповідні компоненти сторінок, реалізовувати захищені маршрути через компонент ProtectedRoute та програмно керувати навігацією. Завдяки клієнтській

маршрутизації переходи між сторінками відбуваються без повного перезавантаження що забезпечує плавний користувацький досвід [11].

Axios версії 1.16.0 є HTTP клієнтом для виконання запитів до серверного API. Порівняно з вбудованим fetch API Axios надає зручніший інтерфейс та підтримує interceptors функції, що виконуються автоматично перед кожним запитом або після отримання відповіді [12]. Request interceptor автоматично додає JWT токен до заголовку кожного запиту, а response interceptor перехоплює відповіді зі статусом 401 і виконує автоматичне розлогінення користувача [9].

TanStack Query версії 5.100.9 є бібліотекою для управління серверним станом у React застосунках. Вона забезпечує кешування відповідей від сервера, автоматичне повторне отримання даних при фокусі вікна та зручний інтерфейс для відстеження стану завантаження [13]. Це дозволяє уникнути зайвих запитів до сервера при повторних відвідуваннях тих самих сторінок [4,13].

Zustand версії 5.0.13 є бібліотекою для управління глобальним станом у React застосунках [23]. Вона використовується поряд з React Context для управління станом автентифікації та надає простий API без зайвого шаблонного коду характерного для більш важких рішень [4].

React Hot Toast версії 2.6.0 використовується для відображення сповіщень користувачу підтвердження успішних дій, повідомлення про помилки та інша інформація, що потребує негайного відображення [22]. CSS Modules використовуються для стилізації компонентів з ізоляцією стилів, де кожен компонент має власний файл стилів з локальними назвами класів що унеможливлює конфлікти між стилями різних компонентів [4].

Серед інструментів розробки використовувались Git, як система контролю версій та GitHub, як платформа для зберігання репозиторію. Git дозволяв відстежувати всі зміни в коді та повертатись до попередніх версій у разі необхідності.

Visual Studio Code використовувався, як основне середовище розробки завдяки широкій підтримці JavaScript та TypeScript, великій кількості розширень та

вбудованій підтримці Git. ESLint версії 10.2.1 використовувався для статичного аналізу коду фронтенду та виявлення потенційних помилок на етапі розробки.

2.6 Реалізація основних класів та методів

Реалізація серверної частини платформи Ludara складається з кількох ключових функцій та механізмів, що забезпечують основну бізнес-логіку системи. У цьому підрозділі розглядаються найбільш важливі з них: механізм автентифікації, патерн відкладеного збереження ігор, система транзакцій при блокуванні користувачів, алгоритм рекурсивного отримання даних з RAWG API та схеми валідації вхідних даних [7].

Функція authenticate є елементом системи безпеки платформи і виконується перед кожним захищеним запитом. Її реалізація передбачає зчитування заголовку Authorization з об’єкту запиту та перевірку наявності префіксу Bearer. Якщо заголовок відсутній або має неправильний формат функція негайно завершує обробку запиту з кодом 401 та повідомленням про відсутність токена. У разі коректного формату заголовку з нього вилучається чистий рядок токена шляхом відрізання перших семи символів префіксу. Токен верифікується через функцію jwt.verify з використанням секретного ключа зі змінних середовища, ця операція одночасно перевіряє підпис токена та термін його дії [9]. При успішній верифікації декодований payload записується в об’єкт запиту як req.user і передається до контролера. При будь-якій помилці вериIфікації невалідний підпис або протермінований токен повертається код 401. Повний код функції authenticate наведено у додатку Б.

Функція ensureGameExists реалізує патерн відкладеного збереження і є ключовим елементом інтеграції з RAWG API. Вона викликається перед кожною операцією оцінювання або встановлення статусу гри і вирішує проблему синхронізації між зовнішнім каталогом RAWG та внутрішньою базою даних платформи. Алгоритм функції є простим але ефективним, спочатку виконується пошук гри у внутрішній базі даних за її RAWG ідентифікатором. Якщо гра вже

існує у базі даних функція повертає існуючий запис без будь-яких додаткових запитів до зовнішнього API. Якщо ж гра відсутня у внутрішній базі виконується запит до RAWG API для отримання її метаданих: назви, обкладинки, опису, року виходу та жанрів. Отримані дані одразу зберігаються у таблиці Game і функція повертає новостворений запис. Такий підхід гарантує що внутрішня база даних містить лише ті ігри з якими реально взаємодіяли користувачі і не потребує попереднього імпорту всього каталогу RAWG [7]. Повний код функції ensureGameExists наведено у додатку В.

Метод banUser демонструє використання транзакцій Prisma для забезпечення атомарності критичних операцій. Блокування користувача є операцією, яка складається з двох незалежних дій: оновлення статусу блокування в таблиці User та створення запису в журналі банів у таблиці Ban. Виконання цих двох операцій окремо створювало б ризик неузгодженості даних, наприклад якщо перша операція успішно виконається, а друга зазнає невдачі система опинилась би у стані, де користувач заблокований але запис про це в журналі відсутній. Використання prisma.$transaction гарантує, що або обидві операції виконуються успішно або жодна з них, у разі будь-якої помилки база даних автоматично повертається до попереднього стану [5]. Окрім транзакційної логіки метод також реалізує захист адміністраторів від блокування, перед виконанням операції перевіряється роль цільового користувача і спроба заблокувати адміністратора завершується з кодом 403. Повний код методу banUser наведено у додатку Д.

Функція getGames у сервісі rawg.service.js реалізує складний алгоритм рекурсивного отримання та фільтрації ігор з RAWG API. Необхідність такого алгоритму обумовлена тим що RAWG API повертає певну кількість результатів на сторінку але частина з них відфільтровується на стороні сервера через функцію filterGames, видаляється NSFW контент, моди, вікі, гайди та ігри без жанрів [7]. Після фільтрації кількість результатів на сторінці може виявитись меншою за очікувану, що призведе до неповної сторінки каталогу. Для вирішення цієї проблеми реалізований цикл, що продовжує завантажувати наступні сторінки RAWG доки не буде зібрано потрібну кількість відфільтрованих результатів або

RAWG не поверне порожню відповідь. Функція filterGames перевіряє кожну гру за кількома критеріями: назва не повинна містити слів зі списку заблокованих слів для NSFW контенту, теги гри не повинні містити заблокованих slug значень, назва не повинна містити слів характерних для неофіційного контенту таких як wiki, guide, mod, tutorial, гра не повинна мати посилання на батьківську гру що означає що це DLC або доповнення, та гра повинна мати хоча б один жанр. Фіксовані параметри запиту до RAWG включають виключення доповнень через параметр exclude_additions та обмеження платформ до актуальних PC, PlayStation 5, Xbox One і Series та Nintendo Switch [7]. Повний код функцій filterGames наведено у додатку Е, код фунції getGames зображено у додатку Ж.

Валідація вхідних даних реалізована через бібліотеку Zod і визначена безпосередньо у файлах контролерів.

Схема registerSchema перевіряє що нікнейм містить від трьох до п’ятдесяти символів, електронна пошта має коректний формат та пароль містить щонайменше вісім символів [14].

Схема loginSchema перевіряє формат електронної пошти та наявність пароля.

Схема rateSchema валідує що ідентифікатор гри є позитивним цілим числом, оцінка знаходиться в діапазоні від одного до десяти та текст рецензії не перевищує п’яти тисяч символів [14].

Схема setStatusSchema перевіряє що статус є одним з трьох допустимих значень: WANT, PLAYING або COMPLETED.

Схема updateRoleSchema навмисно обмежує допустимі ролі значеннями USER та CRITIC виключаючи можливість призначення ролі ADMIN через API.

Схема banSchema вимагає обов’язкової причини блокування та опціональної дати закінчення у форматі ISO datetime.

Схема updateProfileSchema використовує метод refine для крос-полевої валідації, всі три поля є опціональними але хоча б одне з них обов’язково має бути присутнє у запиті. Повний код всіх схем валідації наведено у додатку З.

2.7 Розробка інтерфейсу користувача

Клієнтська частина платформи Ludara є односторінковим застосунком побудованим на React 19 з використанням клієнтської маршрутизації через React Router 7. Інтерфейс розроблений з акцентом на простоту використання та зручний доступ до основних функцій платформи [4]. Загальну структуру застосунку наведено на рисунку 2.5.

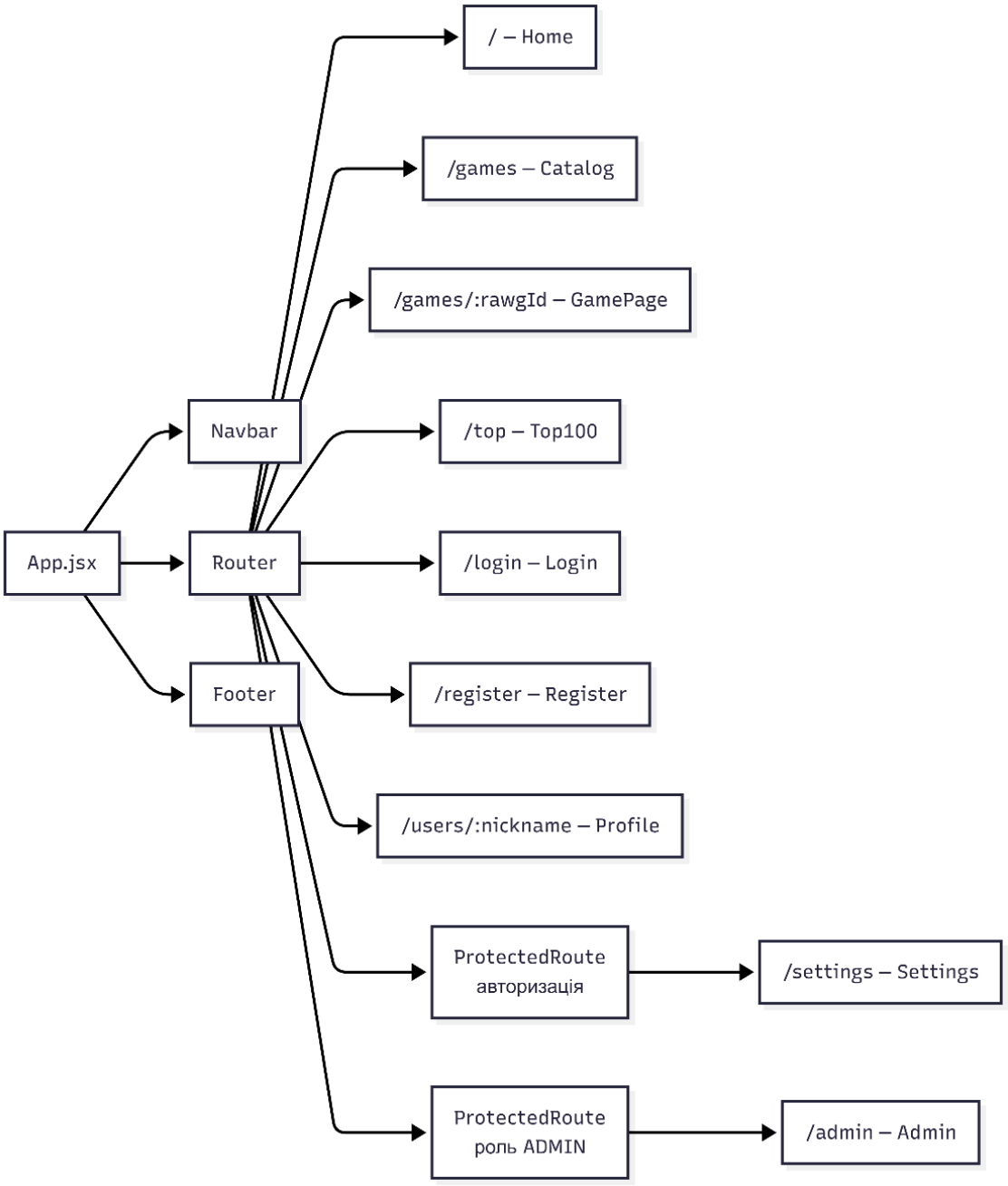


Рисунок 2.5 – Загальна структура застосунку

Навігаційна панель є фіксованим елементом інтерфейсу висотою 64 пікселі, що відображається на всіх сторінках платформи. Зліва розміщено логотип Ludara з градієнтним пурпурним кольором що є одночасно посиланням на головну сторінку. Поряд з логотипом розміщені навігаційні посилання на каталог ігор та сторінку Top 100, активне посилання виділяється завдяки компоненту NavLink. У правій частині навігаційної панелі розміщено кнопку перемикання теми оформлення та блок авторизації. Для неавторизованих користувачів відображаються кнопки входу та реєстрації, для авторизованих: аватар і нікнейм що відкривають випадаюче меню з посиланнями на профіль, налаштування та кнопкою виходу. Для адміністраторів у випадаючому меню додатково відображається посилання на адміністративну панель. Вибрана тема оформлення зберігається у localStorage під ключем ludara_theme і відновлюється при наступному відвідуванні платформи.

Захист маршрутів реалізований через компонент ProtectedRoute, який перевіряє стан авторизації перед відображенням сторінки. Поки контекст авторизації перевіряє збережений токен через запит до сервера компонент відображає порожній екран щоб уникнути миготіння редиректу. Після перевірки неавторизований користувач перенаправляється на сторінку входу, а користувач з недостатньою роллю – на головну сторінку. Маршрут налаштувань вимагає лише авторизації, маршрут адміністративної панелі вимагає ролі ADMIN.

Головна сторінка є першою точкою контакту користувача з платформою і містить великий hero блок із закликом до дії та двома кнопками: переходу до каталогу ігор та сторінки Top 100. Нижче розміщена секція з найвище оціненими іграми на платформі. Інтерфейс головної сторінки зображено на рисунку 2.6.

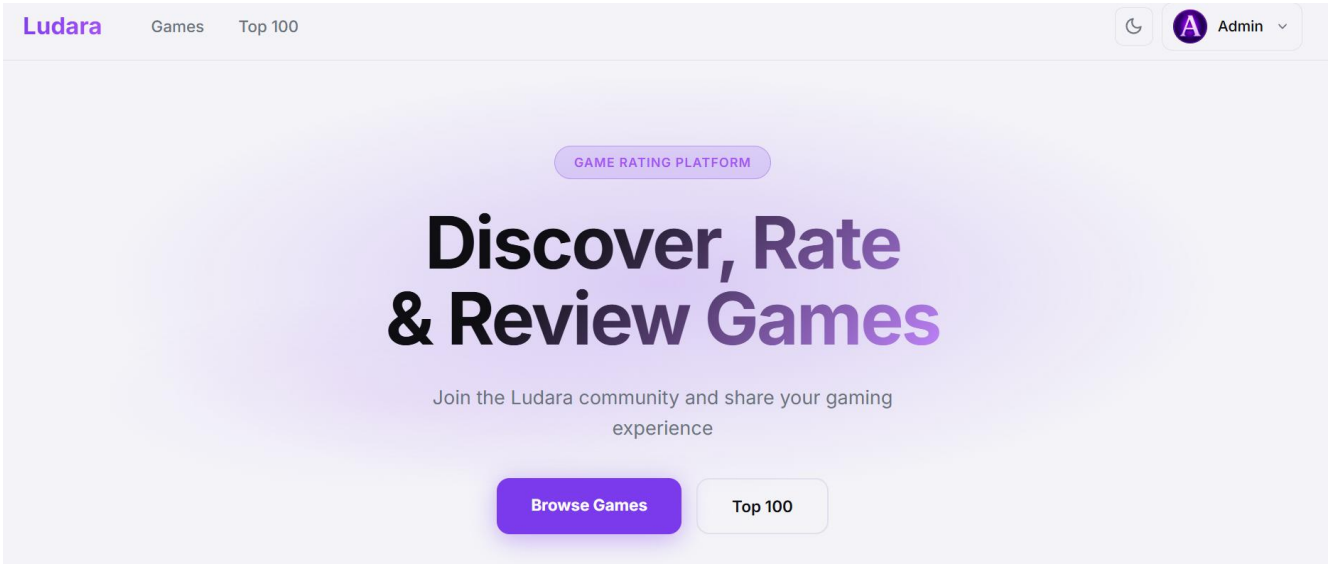


Рисунок 2.6 – Інтерфейс головної сторінки

Сторінка каталогу ігор є основним інструментом для пошуку та перегляду ігор на платформі, інтерфейс зображено на рисунку 2.7.

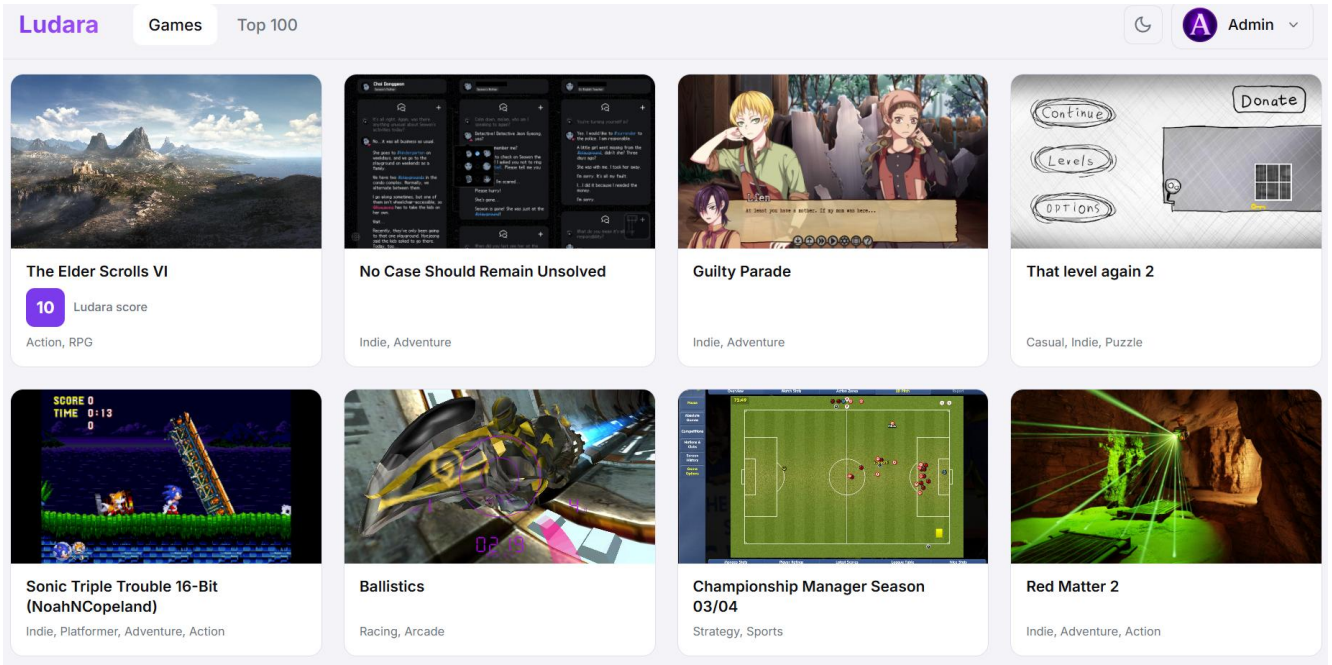


Рисунок 2.7 – Інтерфейс каталогу ігор

У верхній частині сторінки розміщена панель фільтрів, що складається з чотирьох елементів в одному рядку: пошукового поля з іконкою лупи та дебаунсом у 500 мілісекунд, випадаючого списку жанрів з дев’ятьма варіантами, випадаючого

списку року релізу від 1990 до поточного року та випадаючого списку сортування з шістьма варіантами. Всі параметри фільтрів синхронізовані з URL через useSearchParams, що дозволяє зберігати та передавати посилання на конкретний стан каталогу. При зміні будь-якого фільтру сторінка автоматично скидається до першої. Ігри відображаються у вигляді п’ятиколонкової сітки карток де кожна картка містить обкладинку гри, її назву, рейтинг Ludara якщо він є та жанри. Пагінація реалізована у вигляді кнопок переходу між сторінками з вікном номерів що завжди показує першу, останню та дві сторінки навколо поточної з розділювачами між розривами.

Сторінка конкретної гри є центральним елементом платформи, де користувач може ознайомитись з грою та взаємодіяти з нею, інтерфейс зображено на рисунку 2.8.

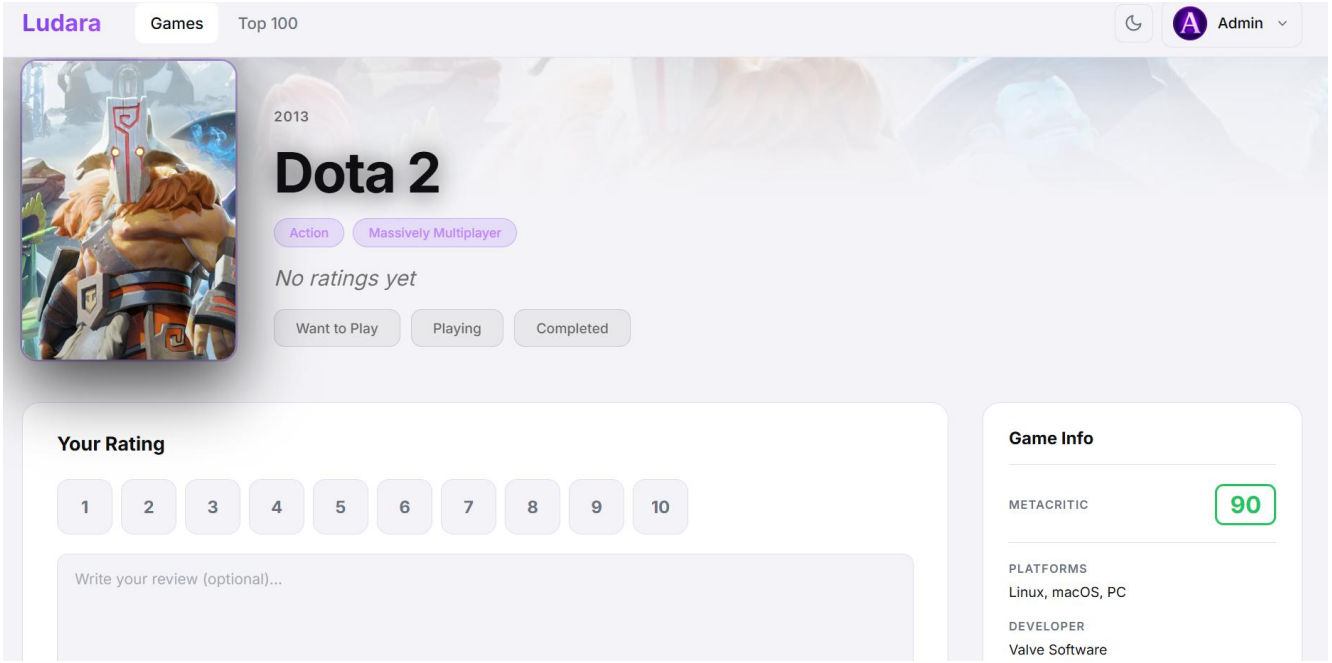


Рисунок 2.8 – Інтерфейс сторінки гри

У верхній частині розміщений hero банер з фоновим зображенням гри на всю ширину та градієнтною накладкою. Поверх банера відображається блок з обкладинкою гри, роком виходу, назвою, тегами жанрів, рейтингом Ludara у вигляді великого числа пурпурного кольору та кнопками встановлення статусу гри.

Основний контент сторінки організований у двоколонковому форматі. Ліва колонка містить блок оцінювання з десятьма кнопками числових оцінок, де колір активної кнопки залежить від значення, зелений для оцінок від восьми і вище, жовтий для оцінок від п’яти і вище та червоний для нижчих оцінок. Нижче розміщене текстове поле для рецензії та кнопки збереження або оновлення оцінки. Окремими блоками відображаються рецензії критиків та звичайних користувачів, де кожна картка рецензії містить аватар автора, нікнейм з посиланням на профіль, бейдж критика якщо потрібно, дату, кольоровий бейдж з оцінкою та кнопку вподобань. Права колонка містить картку з технічною інформацією про гру – платформи, розробник, видавець та кількість рецензій на платформі.

Сторінка профілю користувача є публічною і доступна для перегляду будь-яким відвідувачем платформи, інтерфейс зображено на рисунку 2.9.

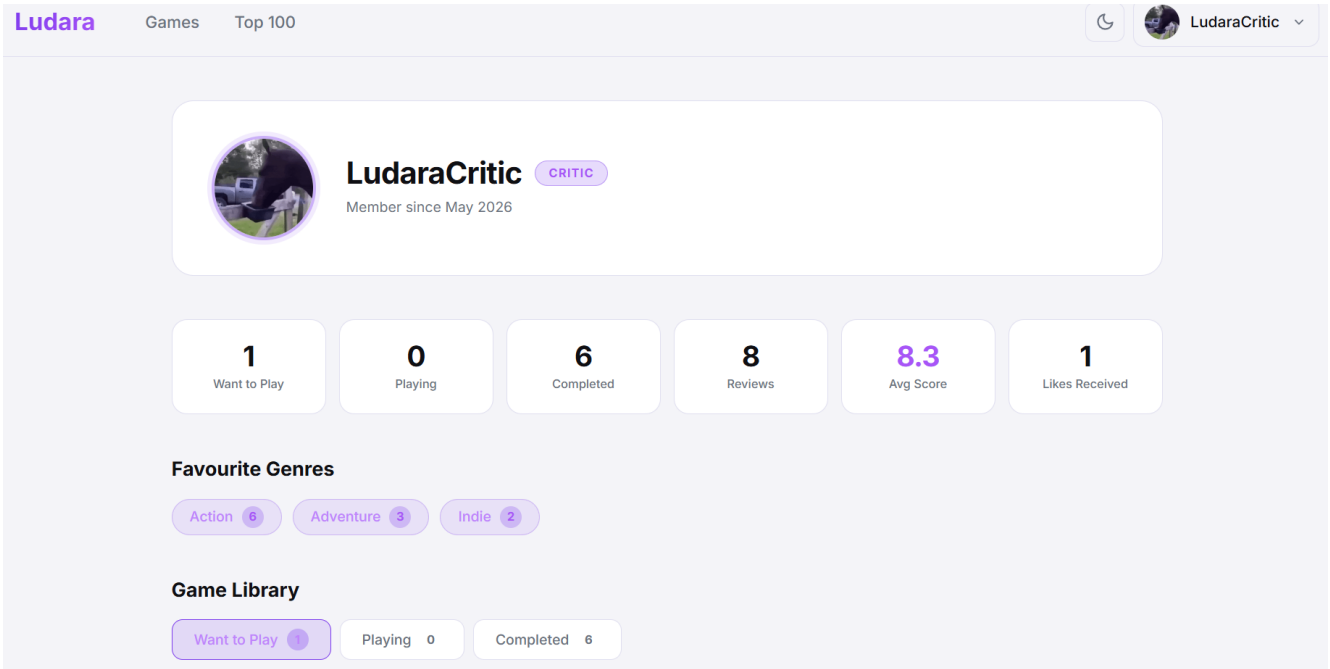


Рисунок 2.9 – Інтерфейс профілю користувача

У верхній частині відображається шапка профілю з аватаром користувача або його ініціалом на пурпурному фоні, нікнеймом, бейджем ролі для критиків та адміністраторів і датою реєстрації. Нижче розміщений блок статистики у вигляді шести карток: кількість ігор у кожному з трьох статусів, загальна кількість

рецензій, середня оцінка та кількість отриманих вподобань. Блок улюблених жанрів відображає теги найчастіших жанрів ігор, що їх оцінив користувач з числовим лічильником. Ігрова бібліотека організована у вигляді трьох вкладок з лічильниками, де при перемиканні відображається відповідна сітка ігрових карток. Блок рецензій містить список всіх рецензій користувача у вигляді карток з обкладинкою гри, назвою, кольоровим бейджем оцінки та текстом рецензії.

Сторінка Top 100 відображає рейтинг найкращих ігор платформи за оцінками спільноти Ludara, інтерфейс зображено на рисунку 2.10.

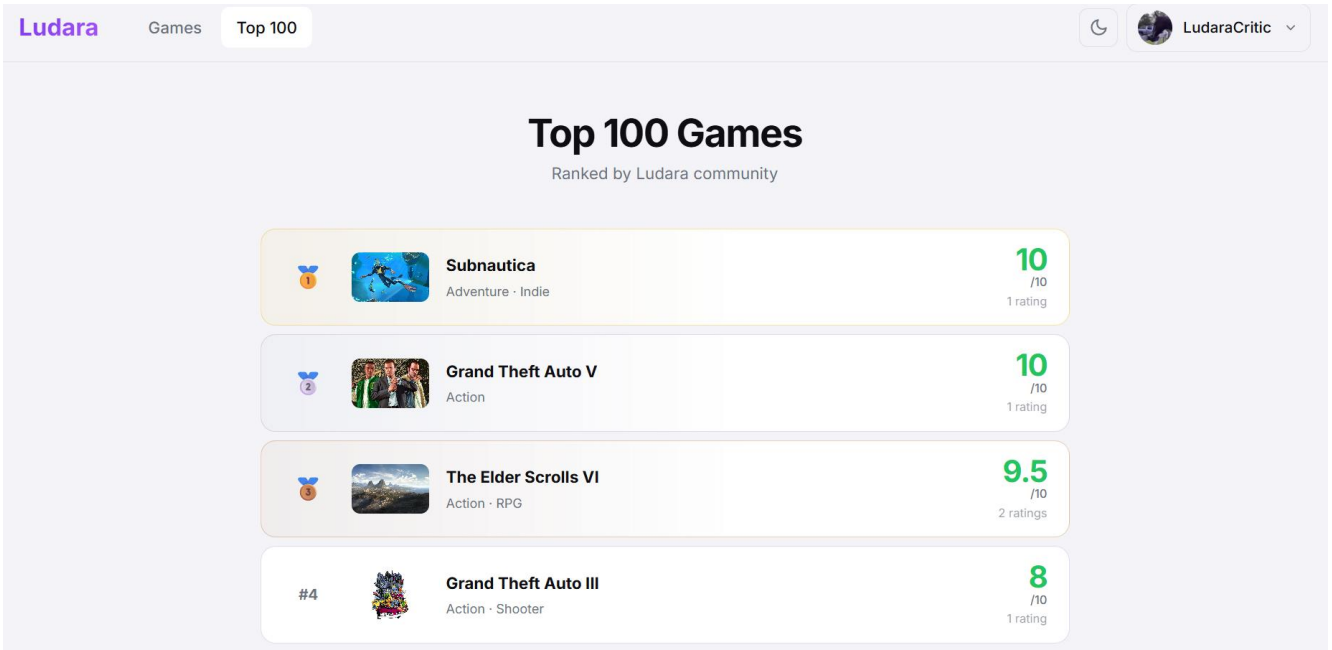


Рисунок 2.10 – Інтерфейс сторінки Топ-100 ігор

Ігри відображаються у вигляді пронумерованого списку, де для трьох перших місць використовуються відповідні медалі замість числового номера. Рядки трьох перших місць додатково виділені кольоровим підсвіченням фону. Кожен рядок містить мініатюру обкладинки гри розміром 60 на 60 пікселів, назву та жанри у лівій частині та кольоровий рейтинг з кількістю оцінок у правій. Кожен рядок є посиланням на відповідну сторінку гри.

Адміністративна панель доступна лише для користувачів з роллю ADMIN і організована у вигляді двох вкладок що перемикаються без зміни маршруту.

Вкладка статистики відображає три числові картки із загальною кількістю користувачів, оцінок та ігор що мають хоча б одну оцінку, а також список п’яти найактивніших користувачів з аватаром, нікнеймом, бейджем ролі та кількістю рецензій. Інтерфейс вкладки статистики зображено на рисунку 2.11.

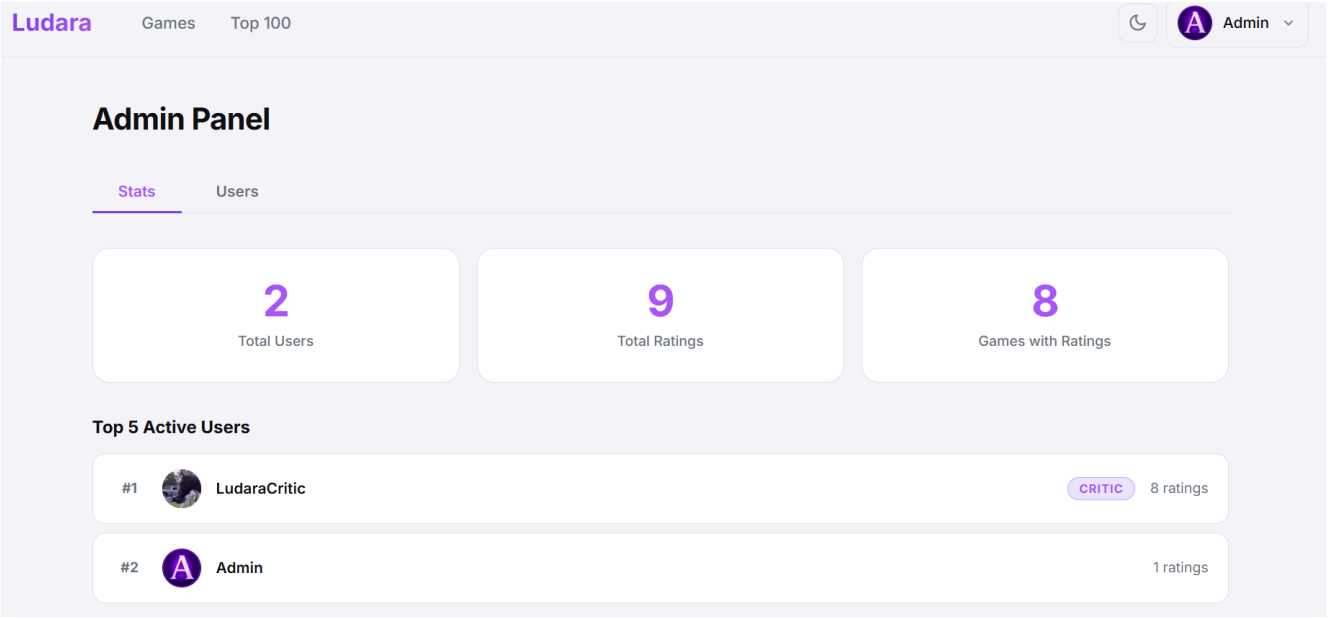


Рисунок 2.11 – Інтерфейс адміністратора з статистикою

Вкладка керування користувачами містить рядок пошуку для фільтрації за нікнеймом або електронною поштою та таблицю з колонками номера, нікнейму, електронної пошти, ролі, статусу та дій. Заголовки колонок є сортовними, клік перемикає порядок сортування. Для кожного користувача доступні дії зміни ролі через випадаючий список, блокування через модальне вікно, розблокування та видалення з підтвердженням. Власний акаунт адміністратора та акаунти інших адміністраторів захищені від адміністративних дій. Модальне вікно блокування містить поле для введення причини та поле вибору дати закінчення блокування, якщо дату не вказати блокування буде постійним. Інтерфейс вкладки керування користувачами зображено на рисунку 2.12.

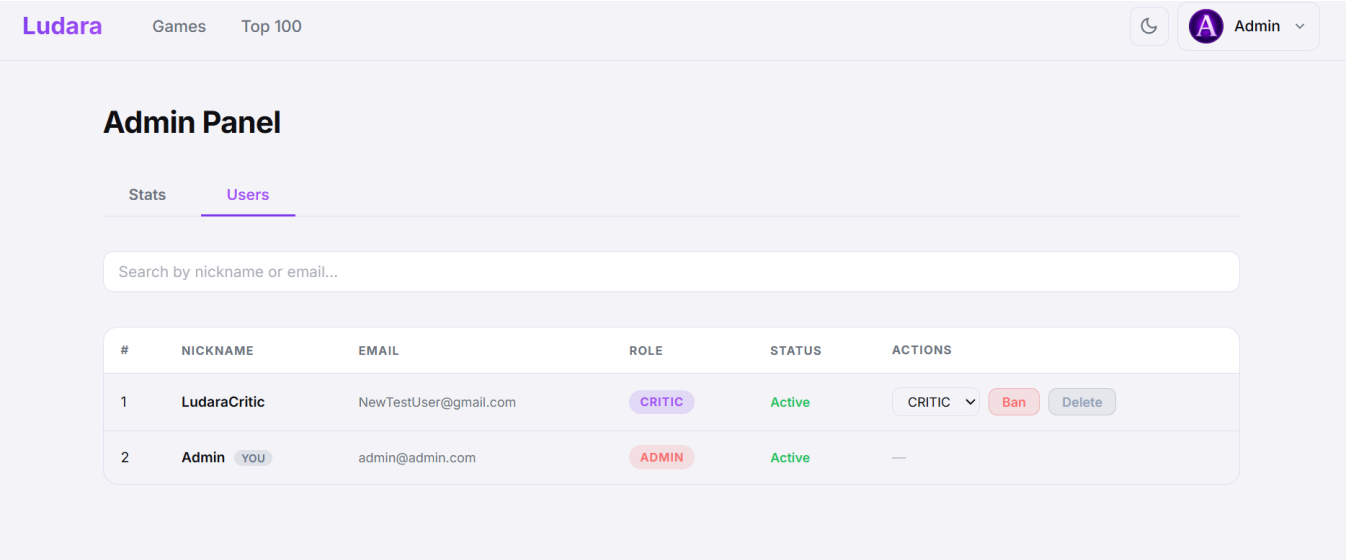


Рисунок 2.12 – Інтерфейс адміністратора з керуванням користувачів

Сторінка налаштувань доступна лише для авторизованих користувачів і організована у вигляді трьох незалежних секцій, інтерфейс зображено на рисунку 2.13.

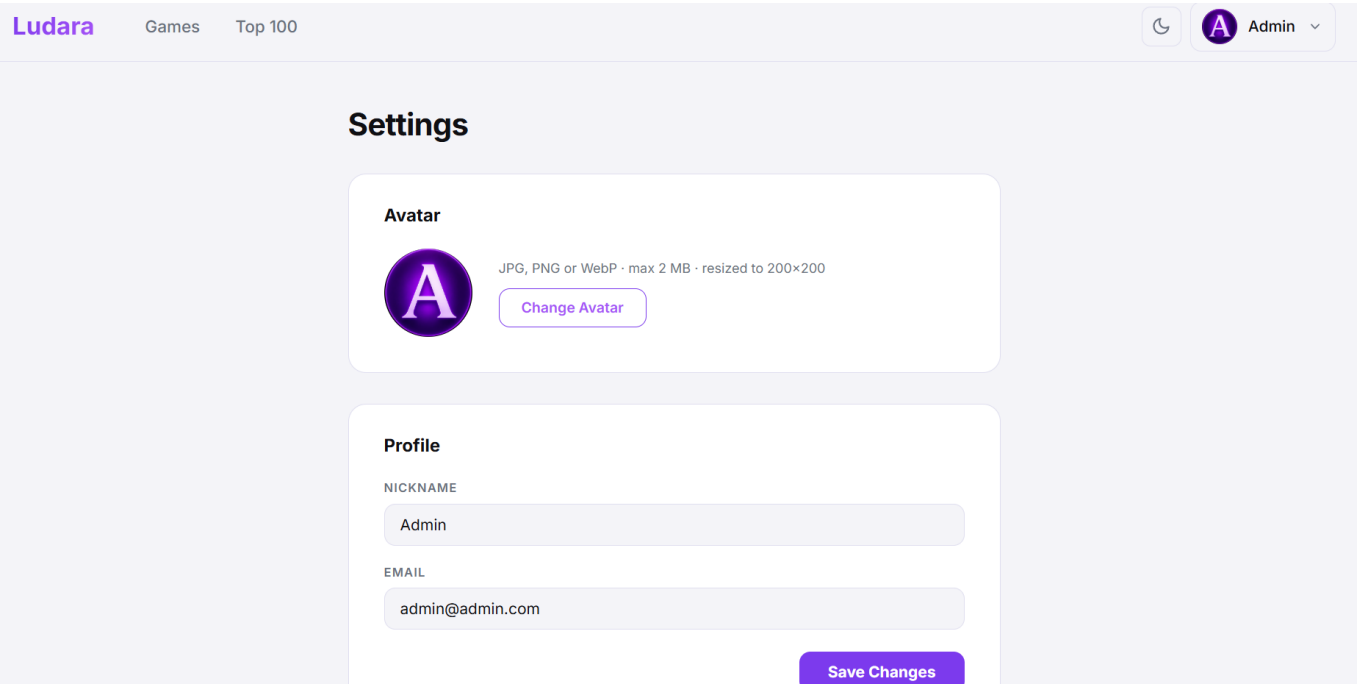


Рисунок 2.13 – Інтерфейс налаштування профілю

Секція аватара відображає поточне фото профілю та кнопку його зміни, що відкриває системний діалог вибору файлу. Підтримуються формати JPG, PNG та WebP з максимальним розміром два мегабайти, під час завантаження кнопка блокується і поверх аватара відображається індикатор завантаження. Секція профілю містить поля для зміни нікнейму та електронної пошти з початковими значеннями поточного користувача, при збереженні відправляються лише змінені поля.

2.8 Використання системи контролю версій та розгортання проєкту

Частиною процесу розробки програмного забезпечення є використання системи контролю версій. Вона дозволяє відстежувати історію змін у коді, повертатись до попередніх версій у разі виникнення помилок та документувати процес розробки через описові повідомлення до кожної зміни. Для розробки платформи Ludara використовувалась система контролю версій Git у поєднанні з платформою GitHub для зберігання віддаленого репозиторію та інструментом GitHub Desktop для зручної роботи з комітами через графічний інтерфейс.

Репозиторій проєкту розміщено на платформі GitHub у приватному режимі доступу. Структура репозиторію відповідає загальній структурі проєкту, окремі директорії для серверної та клієнтської частин, файл .gitignore для виключення конфіденційних даних та залежностей з репозиторію, а також файли документації PROJECT_OVERVIEW.md та README.md. Загальну структуру репозиторію наведено на рисунку 2.14.

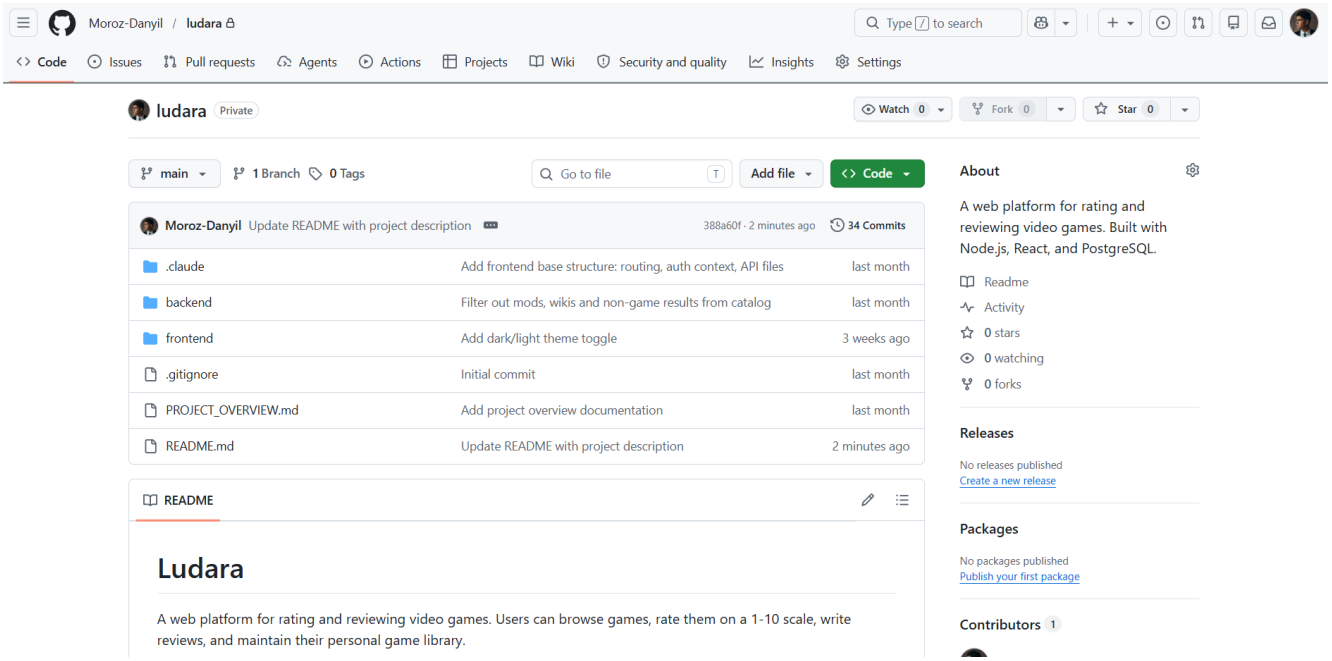


Рисунок 2.14 – Головна сторінка репозиторію

Підхід до роботи з системою контролю версій базувався на принципі атомарних комітів, де кожен коміт відповідає одному завершеному функціональному модулю або виправленню. Такий підхід забезпечує чітку та зрозумілу історію розробки, де з назви кожного коміту одразу зрозуміло, які зміни були внесені. Повідомлення до комітів формулювались англійською мовою у форматі дієслово плюс опис змін, наприклад «Add authentication module: register, login, me» або «Add games module with RAWG API integration».

Загалом репозиторій містить 33 коміти, що охоплюють повний цикл розробки платформи від початкової ініціалізації до фінальних правок інтерфейсу. Розробка велась послідовно відповідно до ітеративної моделі описаної у підрозділі 2.1, спочатку повністю реалізовувались модулі серверної частини а потім клієнтської. Початкові коміти серверної частини наведено на рисунку 2.15.

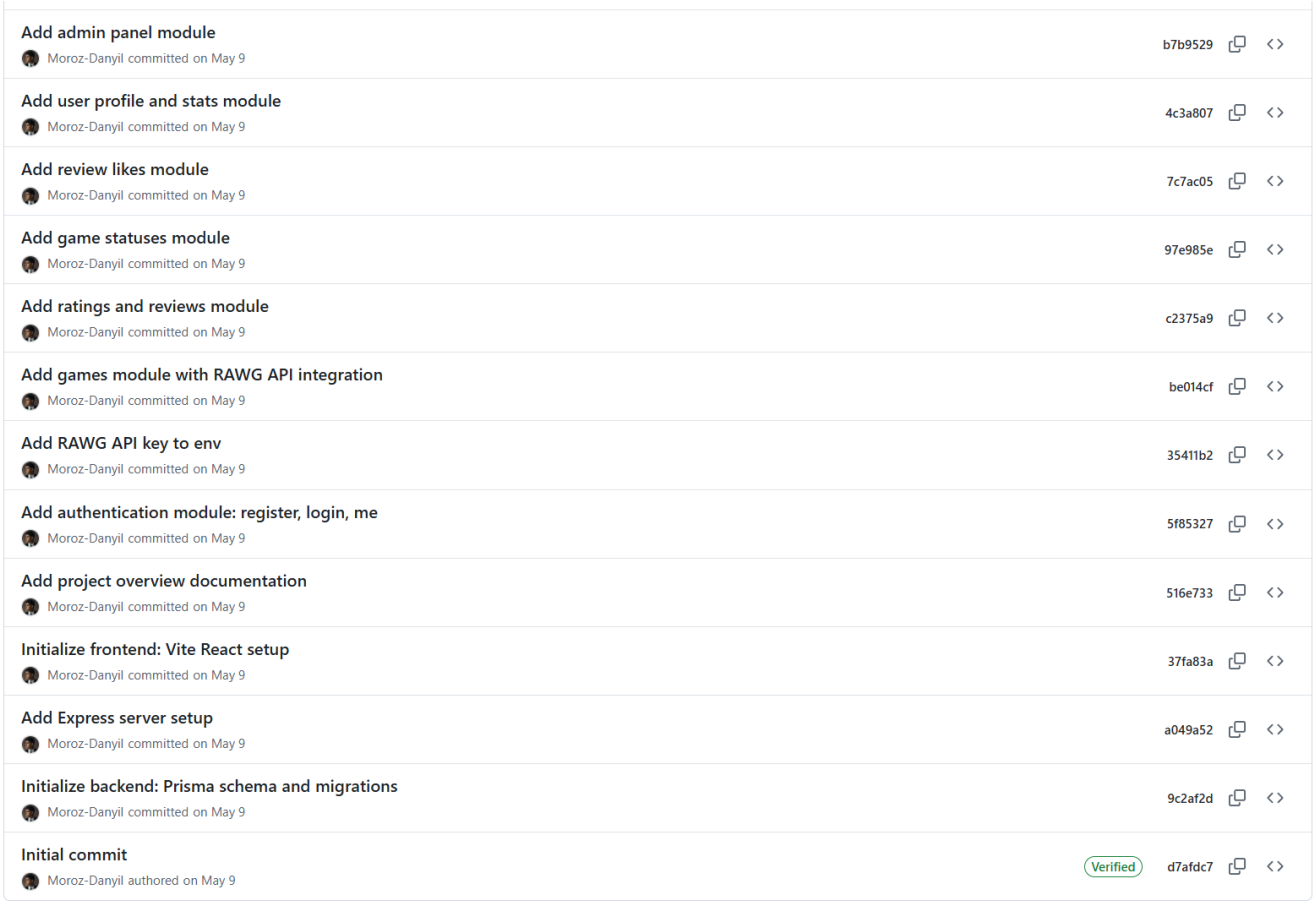


Рисунок 2.15 – Коміти серверної частини

Розробка серверної частини охопила дев’ять послідовних комітів виконаних 9 травня 2026 року - від ініціалізації бази даних через Prisma Schema та першої міграції до повної реалізації всіх функціональних модулів бекенду. Після завершення серверної частини розпочалась розробка клієнтської, базова структура фронтенду, сторінки каталогу та гри, профіль користувача та адміністративна панель. Подальші коміти охоплювали виправлення помилок, покращення функціоналу та додавання нових можливостей, наприклад фільтрацію NSFW контенту, завантаження аватарів через Cloudinary, налаштування профілю та підтримку темної і світлої теми. Коміти фронтенду та фінальних правок наведено на рисунку 2.16.

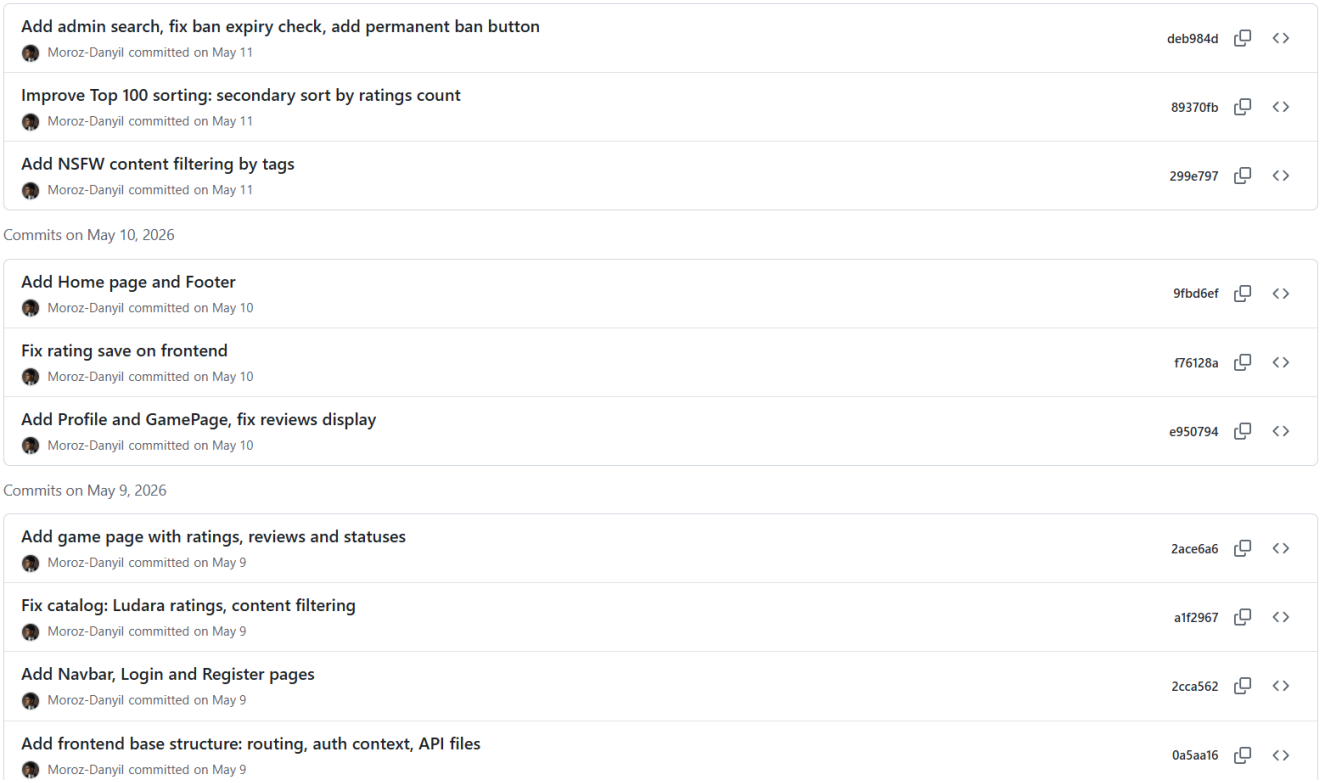


Рисунок 2.16 – Коміти фронтенду та фінальних правок

Для розгортання платформи у середовищі розробки необхідно виконати кілька послідовних кроків. Після клонування репозиторію встановлюються залежності серверної та клієнтської частин через менеджер пакетів npm. У директорії серверної частини створюється файл змінних середовища .env з параметрами підключення до бази даних PostgreSQL, секретним ключем JWT, ключем RAWG API та обліковими даними Cloudinary. Після налаштування змінних середовища виконуються міграції бази даних через команду Prisma Migrate, що автоматично створює всі необхідні таблиці та індекси. Запуск платформи відбувається одночасним стартом серверної частини на порту 3000 та клієнтської на порту 5173 після чого платформа стає доступною у браузері.

У другому розділі описано процес проєктування та розробки платформи Ludara. Обрано ітеративну модель розробки, як найбільш відповідну для індивідуального проєкту такого масштабу. Спроєктовано клієнт-серверну архітектуру з чітким розмежуванням відповідальності між компонентами системи. Розроблено схему бази даних, що містить шість таблиць з відповідними зв’язками

та обмеженнями. Визначено та обґрунтовано технологічний стек Node.js, Express.js, PostgreSQL, Prisma та супутні бібліотеки. Реалізовано всі заплановані функціональні модулі серверної та клієнтської частин платформи. Для організації процесу розробки використовувалась система контролю версій Git з розміщенням репозиторію на платформі GitHub, загалом було здійснено 33 коміти, що охоплюють повний цикл розробки від ініціалізації проєкту до фінальних правок інтерфейсу.

РОЗДІЛ 3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Розділ присвячено опису процесу тестування, розгортання та підтримки платформи Ludara. Тестування є невід'ємною частиною життєвого циклу програмного забезпечення і дозволяє переконатись у коректності роботи всіх функціональних модулів системи. Розглядаються підходи до тестування, що застосовувались під час розробки, конкретні тестові сценарії для ключових функцій платформи, а також вимоги до середовища розгортання та процес верифікації готового програмного продукту.

3.1 Види та план тестування

Тестування програмного забезпечення є процесом перевірки відповідності розробленої системи встановленим вимогам та виявлення дефектів до введення продукту в експлуатацію. Для платформи Ludara основним підходом до тестування обрано ручне функціональне тестування, що є доцільним рішенням для індивідуального проєкту з чітко визначеним функціоналом.

Функціональне тестування спрямоване на перевірку того що кожна функція системи працює відповідно до визначених вимог. У рамках цього підходу перевірялась коректність роботи кожного модуля платформи, від реєстрації та авторизації до адміністративних функцій. Кожна нова функція перевірялась одразу після реалізації, що дозволяло виявляти та усувати дефекти на ранньому етапі без накопичення технічного боргу.

Тестування зручності використання проводилось паралельно з функціональним і було спрямоване на перевірку того, що інтерфейс платформи є зрозумілим та зручним. У рамках цього виду тестування перевірялась логічність навігації між сторінками, зрозумілість повідомлень про помилки що відображаються користувачу, коректність відображення інтерфейсу при різних станах даних: порожня бібліотека, відсутність рецензій, заблокований акаунт тощо.

Тестування безпеки проводилось для перевірки коректності роботи механізмів автентифікації та авторизації. Перевірялась недоступність захищених маршрутів для неавторизованих користувачів, неможливість виконання адміністративних дій користувачами без відповідної ролі, коректна обробка протермінованих та невалідних токенів, а також захист адміністраторів від блокування або зміни ролі з боку інших адміністраторів.

Тестування граничних значень проводилось для перевірки поведінки системи при введенні даних на межі допустимих значень. Перевірялась поведінка форм при введенні мінімальної та максимальної допустимої кількості символів, реакція системи на спробу виставити оцінку менше одиниці або більше десяти, поведінка при завантаженні файлу що перевищує допустимий розмір у два мегабайти та при передачі файлу невірного формату.

Регресійне тестування проводилось після кожного виправлення виявлених дефектів для підтвердження, що внесені зміни не порушили роботу вже перевіреного функціоналу. Особлива увага приділялась модулям, що мають між собою тісні залежності, наприклад зміни в модулі автентифікації могли вплинути на роботу захищених маршрутів або адміністративних функцій.

План тестування передбачав послідовну перевірку модулів у порядку їх розробки, спочатку перевірявся модуль автентифікації, потім каталог ігор та інтеграція з RAWG API, далі система оцінювання та рецензування, модуль статусів ігор, система вподобань, профіль користувача зі статистикою та адміністративний модуль. Після завершення розробки фронтенду проводилось наскрізне тестування всієї платформи, де перевірялась коректна взаємодія між усіма модулями системи в рамках реальних користувацьких сценаріїв.

3.2 Розробка тестових сценаріїв

Тестові сценарії розроблялись для кожного функціонального модуля платформи і охоплювали, як основні так і альтернативні шляхи виконання. Кожен сценарій визначає початкові умови, послідовність дій та очікуваний результат.

Модуль автентифікації є найкритичнішим з точки зору безпеки і тому потребував особливо ретельної перевірки. Тестувались як успішні сценарії реєстрації та входу так і граничні випадки, спроба реєстрації з вже існуючими даними, вхід з невірним паролем, вхід заблокованого користувача та спроба доступу до захищених маршрутів без авторизації. Тестові сценарії модуля автентифікації наведено в таблиці 3.1.

Таблиця 3.1 – Тестові сценарії модуля автентифікації

Сценарій	**Початкові умови**	**Дії**	**Очікуваний результат**
Успішна реєстрація	Користувач не зареєстрований	Ввести унікальний нікнейм, електронну пошту та пароль не менше 8 символів	Обліковий запис створено, користувач авторизований, отримано JWT токен
Реєстрація з існуючим нікнеймом	Нікнейм вже зайнятий	Ввести існуючий нікнейм	Відображається повідомлення про конфлікт, реєстрація не виконана
Успішний вхід	Обліковий запис існує	Ввести коректну пошту та пароль	Користувач авторизований, отримано JWT токен
Вхід з невірним паролем	Обліковий запис існує	Ввести невірний пароль	Відображається повідомлення про невірні дані
Вхід заблокованого користувача	Акаунт заблоковано	Спробувати увійти	Відображається повідомлення про блокування з причиною та терміном
Доступ до захищеного маршруту без токена	Користувач не авторизований	Перейти на сторінку налаштувань	Перенаправлення на сторінку входу

Каталог ігор є основним інструментом навігації платформою і потребував перевірки коректності роботи всіх фільтрів, пошуку та пагінації. Окремо перевірялась синхронізація параметрів фільтрів з URL та автоматичне скидання сторінки при зміні фільтру. Тестові сценарії модуля каталогу наведено в таблиці 3.2.

Таблиця 3.2 – Тестові сценарії модуля каталогу ігор

Сценарій	**Початкові умови**	**Дії**	**Очікуваний результат**
Завантаження каталогу	Сторінка каталогу відкрита	Перейти на /games	Відображається сітка ігор з обкладинками та назвами
Пошук за назвою	Каталог завантажений	Ввести назву гри у пошукове поле	Після 500мс відображаються ігри що відповідають запиту
Фільтрація за жанром	Каталог завантажений	Обрати жанр з випадаючого списку	Відображаються лише ігри обраного жанру
Фільтрація за роком	Каталог завантажений	Обрати рік релізу	Відображаються лише ігри обраного року
Перехід між сторінками	Каталог завантажений	Натиснути кнопку наступної сторінки	Завантажується наступна сторінка ігор
Скидання фільтрів при зміні	Активні фільтри встановлені	Змінити будь-який фільтр	Сторінка скидається до першої

Модуль оцінювання та рецензування є центральним функціоналом платформи і потребував перевірки як базових операцій так і складніших сценаріїв, оновлення існуючої оцінки, видалення оцінки та системи вподобань на рецензіях. Також перевірялась коректна поведінка для неавторизованих користувачів. Тестові сценарії цього модуля наведено в таблиці 3.3.

Таблиця 3.3 – Тестові сценарії модуля оцінювання та рецензування

Сценарій	**Початкові умови**	**Дії**	**Очікуваний результат**
Виставлення оцінки	Користувач авторизований, на сторінці гри	Натиснути на числову оцінку та зберегти	Оцінка збережена, середній рейтинг гри оновлено
Оновлення оцінки	Користувач вже оцінив гру	Обрати нову оцінку та натиснути оновити	Попередня оцінка замінена новою
Видалення оцінки	Користувач вже оцінив гру	Натиснути кнопку видалення оцінки	Оцінка видалена, рейтинг гри перераховано
Написання рецензії	Користувач авторизований	Ввести текст рецензії разом з оцінкою	Рецензія відображається у блоці рецензій
Спроба оцінити без авторизації	Користувач не авторизований	Спробувати натиснути оцінку	Відображається посилання на сторінку входу
Вподобання рецензії	Користувач авторизований	Натиснути кнопку вподобань на рецензії	Лічильник вподобань збільшується на одиницю
Повторне вподобання	Користувач вже відзначив рецензію	Натиснути кнопку вподобань повторно	Вподобання знімається, лічильник зменшується

Модуль статусів ігор відповідає за ведення ігрової бібліотеки користувача і тестувався до коректного відображення статусів на сторінці профілю після їх встановлення або зміни. Тестові сценарії модуля статусів наведено в таблиці 3.4.

Таблиця 3.4 — Тестові сценарії модуля статусів ігор

Сценарій	**Початкові умови**	**Дії**	**Очікуваний результат**
Встановлення статусу	Користувач авторизований, на сторінці гри	Натиснути кнопку статусу	Статус збережено, гра відображається у відповідному розділі бібліотеки профілю
Зміна статусу	Статус вже встановлений	Натиснути іншу кнопку статусу	Попередній статус замінено новим
Перевірка бібліотеки профілю	Статуси встановлені	Перейти на сторінку профілю	Ігри відображаються у відповідних вкладках бібліотеки

Адміністративний модуль потребував ретельного тестування з точки зору безпеки, перевірялись захисні механізми, що унеможливлюють зловживання адміністративними повноваженнями. Окрім базових функцій керування користувачами тестувались граничні випадки, спроба заблокувати іншого адміністратора та спроба видалити власний акаунт. Тестові сценарії адміністративного модуля наведено в таблиці 3.5.

Таблиця 3.5 – Тестові сценарії адміністративного модуля

Сценарій	**Початкові умови**	**Дії**	**Очікуваний результат**
Зміна ролі користувача	Адміністратор у панелі керування	Змінити роль з USER на CRITIC	Роль оновлено, бейдж критика відображається на профілі
Тимчасове блокування	Адміністратор у панелі керування	Заблокувати користувача з датою закінчення	Користувач не може увійти до закінчення терміну
Постійне блокування	Адміністратор у панелі керування	Заблокувати користувача без дати закінчення	Користувач заблокований без можливості входу
Розблокування	Користувач заблокований	Натиснути кнопку розблокування	Користувач може знову увійти в систему
Спроба заблокувати адміністратора	Адміністратор у панелі керування	Спробувати заблокувати іншого адміністратора	Дія відхилена, відображається повідомлення про помилку
Видалення користувача	Адміністратор у панелі керування	Видалити акаунт користувача	Акаунт та всі пов’язані дані видалено

Продовження таблиці 3.5

Сценарій	Початкові умови	Дії	Очікуваний результат
Спроба видалити себе	Адміністратор у панелі керування	Спробувати видалити власний акаунт	Дія відхилена системою

Сторінка налаштувань профілю поєднує три незалежні функціональні блоки, що потребували окремого тестування: завантаження аватара, оновлення профільних даних та зміна пароля. Додаткова увага приділялась клієнтській валідації форм та коректній поведінці при граничних значеннях. Тестові сценарії модуля налаштувань наведено в таблиці 3.6.

Таблиця 3.6 – Тестові сценарії модуля налаштувань профілю

Сценарій	Початкові умови	Дії	Очікуваний результат
Зміна нікнейму	Користувач авторизований	Ввести новий унікальний нікнейм та зберегти	Нікнейм оновлено у навігаційній панелі та профілі
Зміна на зайнятий нікнейм	Нікнейм вже зайнятий	Ввести існуючий нікнейм	Відображається повідомлення про конфлікт
Зміна пароля	Користувач авторизований	Ввести поточний пароль та новий пароль двічі	Пароль оновлено
Невідповідність паролів	Поля паролів заповнені	Ввести різні значення у поля нового пароля	Поля підсвічуються червоним, збереження заблоковано
Завантаження аватара	Користувач авторизований	Вибрати файл зображення до 2МБ	Аватар оновлено у профілі та навігаційній панелі
Завантаження великого файлу	Файл перевищує 2МБ	Спробувати завантажити файл	Відображається повідомлення про перевищення розміру

3.3 Розгортання програмної системи та системні вимоги

Розгортання програмної системи є процесом підготовки та запуску застосунку у робочому середовищі. У даному розділі описуються системні вимоги до середовища розробки та запуску платформи Ludara, а також послідовність кроків необхідних для розгортання серверної та клієнтської частин.

Для коректної роботи платформи Ludara необхідна наявність таких компонентів програмного середовища: Node.js версії 18 або вище є обов’язковим компонентом для запуску серверної частини та інструментів збірки фронтенду. PostgreSQL версії 14 або вище використовується як основна система управління базами даних, необхідно мати доступ до запущеного екземпляру з правами на створення бази даних та виконання міграцій. Менеджер пакетів npm що постачається разом з Node.js використовується для встановлення залежностей обох частин застосунку [6,21].

Мінімальні апаратні вимоги для розгортання платформи у середовищі розробки є наступними. Процесор з тактовою частотою не менше 1.5 ГГц забезпечує комфортну роботу серверної частини та інструментів розробки. Оперативна пам’ять обсягом не менше 4 гігабайт є достатньою для одночасної роботи серверної частини, бази даних та інструментів розробки. Вільний простір на диску обсягом не менше 2 гігабайт необхідний для зберігання коду проєкту, залежностей та даних бази даних.

Процес розгортання платформи розпочинається з клонування репозиторію та налаштування середовища. Після отримання коду проєкту необхідно встановити залежності серверної частини виконавши команду встановлення пакетів у директорії backend. Аналогічно встановлюються залежності клієнтської частини у директорії frontend. Далі необхідно створити файл змінних середовища .env у директорії backend та заповнити його необхідними значеннями: рядком підключення до бази даних PostgreSQL, секретним ключем для підпису JWT токенів, ключем RAWG API та обліковими даними Cloudinary для зберігання аватарів [6].

Після налаштування змінних середовища виконується застосування міграцій бази даних через команду Prisma Migrate, що автоматично створює всі необхідні таблиці, індекси та обмеження відповідно до визначеної схеми [5]. Після успішного виконання міграцій Prisma автоматично генерує типізований клієнт що використовується серверною частиною для взаємодії з базою даних.

Запуск платформи у режимі розробки передбачає одночасний запуск серверної та клієнтської частин. Серверна частина запускається командою запуску з nodemon, що забезпечує автоматичний перезапуск при зміні файлів, сервер починає прослуховувати порт 3000. Клієнтська частина запускається командою запуску Vite, нструмент збірки запускає сервер розробки на порту 5173 з підтримкою гарячого оновлення модулів [10]. Після запуску обох частин платформа доступна у браузері за адресою http://localhost:5173.

Для зовнішніх сервісів, що використовуються платформою необхідно виконати попередню реєстрацію та отримання облікових даних. Для RAWG API необхідно зареєструватись на сайті rawg.io та отримати безкоштовний API ключ у розділі налаштувань облікового запису [7].

Для Cloudinary необхідно зареєструватись на сайті cloudinary.com та отримати назву хмари, API ключ та API секрет у розділі Dashboard, ці дані використовуються для автентифікації при завантаженні аватарів користувачів [8].

3.4 Верифікація програмної системи

Верифікація програмної системи є процесом підтвердження того що розроблений програмний продукт відповідає визначеним на початку розробки вимогам та коректно виконує всі заявлені функції. На відміну від тестування окремих модулів верифікація розглядає систему, як єдине ціле і перевіряє відповідність кінцевого продукту початковим вимогам сформульованим у першому розділі.

Верифікація функціональних вимог проводилась шляхом перевірки кожного з восьми завдань визначених у підрозділі 1.2. Система автентифікації реалізована у повному обсязі: реєстрація з унікальним нікнеймом та електронною поштою, вхід через JWT токен та перегляд каталогу без реєстрації функціонують коректно. Каталог ігор з інтеграцією RAWG API забезпечує пошук за назвою, фільтрацію за жанром та роком релізу, сортування та пагінацію [7]. Система оцінювання та рецензування дозволяє виставляти оцінки за шкалою від 1 до 10, писати текстові

рецензії та відзначити рецензії інших користувачів. Система статусів ігор коректно зберігає та відображає ігрову бібліотеку користувача з розподілом за трьома статусами. Система ролей функціонує відповідно до вимог, критики мають окремий блок рецензій та бейдж на профілі, адміністратори мають повний доступ до панелі керування. Адміністративна панель забезпечує повний набір функцій керування користувачами включно зі зміною ролей, блокуванням та видаленням. Публічні профілі користувачів відображають повну статистику активності. Інтерфейс підтримує перемикання між темною та світлою темою оформлення.

Верифікація нефункціональних вимог охоплювала перевірку безпеки, цілісності даних та коректності обробки помилок. Механізми захисту маршрутів функціонують коректно, неавторизовані користувачі не мають доступу до захищених функцій, а спроби виконати адміністративні дії без відповідної ролі відхиляються з кодом 403. Цілісність даних забезпечується на рівні бази даних через унікальні обмеження та каскадне видалення, видалення користувача автоматично видаляє всі пов’язані записи без порушення цілісності. Обробка помилок реалізована на всіх рівнях системи, клієнт отримує зрозумілі повідомлення про помилки в усіх граничних випадках. Валідація вхідних даних через Zod запобігає збереженню некоректних даних у базі [14].

Верифікація інтеграції із зовнішніми сервісами підтвердила коректну роботу всіх трьох інтеграцій. RAWG API коректно повертає дані каталогу з фільтрацією та пагінацією, дані конкретних ігор отримуються без помилок, механізм відкладеного збереження ігор у внутрішній базі даних працює відповідно до визначеної логіки. Cloudinary коректно приймає завантажені зображення, виконує автоматичне кадрування до розміру 200 на 200 пікселів та повертає публічний URL, що успішно зберігається у базі даних та відображається в інтерфейсі [8]. JWT автентифікація забезпечує коректну ідентифікацію користувачів на всіх захищених маршрутах.

За результатами верифікації платформа Ludara відповідає всім функціональним вимогам визначеним на етапі аналізу та проєктування. Всі вісім завдань поставлених у підрозділі 1.2 реалізовані у повному обсязі та підтверджені в ході тестування. Система коректно обробляє як стандартні так і граничні випадки

використання що свідчить про достатній рівень якості розробленого програмного продукту.

У підсумку в третьому розділі описано процес тестування, розгортання та верифікації платформи Ludara. Проведено ручне функціональне тестування всіх модулів системи з розробкою детальних тестових сценаріїв що охоплюють як основні так і граничні випадки використання. Визначено системні вимоги та описано послідовність кроків для розгортання платформи у середовищі розробки. За результатами верифікації підтверджено що платформа відповідає всім восьми функціональним вимогам визначеним на етапі аналізу – всі заплановані модулі реалізовані у повному обсязі та коректно взаємодіють між собою.

3.5 Автоматизоване тестування програмної системи

Автоматизоване тестування є складовою забезпечення якості програмного забезпечення, що дозволяє систематично перевіряти коректність роботи системи без участі людини. На відміну від ручного тестування автоматизовані тести виконуються швидко і можуть запускатись при кожній зміні коду, що значно знижує ризик появи регресійних помилок у вже працюючому функціоналі.

Для автоматизованого тестування серверної частини Node.js застосунків найбільш поширеними інструментами є Jest та Supertest. Jest є повнофункціональним фреймворком для тестування JavaScript застосунків, що підтримує модульне тестування окремих функцій та інтеграційне тестування взаємодії між модулями. Supertest є бібліотекою, що дозволяє тестувати HTTP ендпоінти Express.js застосунків шляхом надсилання реальних HTTP запитів до сервера та перевірки отриманих відповідей. Комбінація Jest та Supertest є стандартним підходом для тестування REST API на платформі Node.js і дозволяє автоматично перевіряти коректність роботи кожного ендпоінту включно з перевіркою статусних кодів відповідей, структури даних та поведінки системи в граничних випадках.

Для тестування клієнтської частини React застосунків широко використовується бібліотека React Testing Library у поєднанні з Jest. Вона дозволяє тестувати React компоненти з точки зору кінцевого користувача, перевіряти, що компоненти коректно відображають дані, реагують на взаємодію користувача та правильно оновлюють стан. Такий підхід до тестування фронтенду забезпечує впевненість у коректній роботі інтерфейсу незалежно від внутрішньої реалізації компонентів.

Навантажувальне тестування є окремим видом автоматизованого тестування, що спрямоване на перевірку продуктивності системи під великим навантаженням. Для веб-платформ такого типу як Ludara навантажувальне тестування дозволяє визначити максимальну кількість одночасних користувачів, яку система здатна обслуговувати без деградації продуктивності. Серед найбільш популярних інструментів для навантажувального тестування Node.js застосунків виділяються Artillery та k6. Artillery дозволяє описувати сценарії навантажувального тестування у форматі YAML та генерувати детальні звіти про продуктивність системи включно з часом відповіді, кількістю успішних та невдалих запитів і пропускною здатністю сервера. k6 є сучасним інструментом навантажувального тестування, що використовує JavaScript для опису тестових сценаріїв і надає детальну метрику продуктивності у реальному часі.

У рамках розробки платформи Ludara основним підходом до тестування обрано ручне функціональне тестування, що є доцільним рішенням для індивідуального навчального проєкту. Ручне тестування забезпечило повне покриття всіх функціональних вимог та дозволило виявити і усунути дефекти на всіх етапах розробки. При подальшому масштабуванні платформи та збільшенні команди розробників впровадження автоматизованого тестування через Jest та Supertest для серверної частини стало б логічним наступним кроком, що дозволило б підвищити надійність системи та прискорити процес виявлення регресійних помилок. Навантажувальне тестування через Artillery або k6 було б доцільним при розгортанні платформи у виробничому середовищі для визначення оптимальних параметрів серверної інфраструктури.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Розробка програмного забезпечення є діяльністю, що передбачає тривалу роботу за комп’ютером в умовах офісного або домашнього робочого середовища. Такі умови праці пов’язані з рядом факторів що можуть негативно впливати на здоров’я та працездатність розробника. У даному розділі розглядаються питання впливу електромагнітних полів на організм людини та вимоги до організації робочого місця оператора персонального комп’ютера з метою забезпечення безпечних і комфортних умов праці.

4.1 Безпека життєдіяльності

Електромагнітні поля є невід’ємним супутником сучасної цивілізації. Комп’ютерна техніка, мобільні пристрої, Wi-Fi роутери, монітори та інше офісне обладнання є джерелами електромагнітного випромінювання різних частот і інтенсивностей [24]. Розробник програмного забезпечення проводить за комп’ютером значну частину робочого часу, нерідко від шести до десяти годин на добу, що робить питання впливу електромагнітних полів на організм людини особливо актуальним [25].

Електромагнітне поле являє собою особливу форму матерії, що виникає навколо провідників зі струмом та джерел змінної напруги. Воно характеризується двома складовими: електричною та магнітною, які нерозривно пов’язані між собою. Основними характеристиками електромагнітного поля є частота випромінювання, інтенсивність та тривалість впливу на організм людини. Саме ці три параметри визначають ступінь потенційної шкоди для здоров’я [24].

Вплив електромагнітних полів на організм людини залежить від діапазону частот випромінювання. Монітори комп’ютерів та системні блоки генерують поля наднизьких і низьких частот від 3 Гц до 300 кГц. Дослідження показують, що тривалий вплив таких полів може призводити до порушень у роботі центральної

нервової системи, серцево-судинної системи та імунної системи організму [26]. Серед найбільш поширених симптомів хронічного впливу електромагнітних полів на операторів ПК виявлено підвищену втомлюваність, головний біль, погіршення концентрації уваги, порушення сну та дратівливість [25].

Особливої уваги заслуговує вплив електромагнітних полів на орган зору. Тривала робота з монітором в умовах електромагнітного навантаження може спричиняти синдром комп'ютерного зору, це комплекс симптомів що включає сухість та подразнення очей, нечіткість зображення, підвищену чутливість до світла та біль у скронях [26]. За даними досліджень цей синдром виявляється у значної частини фахівців що систематично працюють за комп'ютером більше чотирьох годин на добу.

Основними джерелами електромагнітного випромінювання на робочому місці розробника програмного забезпечення є монітор комп'ютера, системний блок або ноутбук, клавіатура та миша з бездротовим підключенням, Wi-Fi адаптер та роутер, а також зарядні пристрої для мобільних телефонів та інших гаджетів. Сукупний вплив усіх цих джерел на організм людини є предметом постійних наукових досліджень [27].

Для зниження негативного впливу електромагнітних полів на організм розробника застосовується комплекс технічних та організаційних заходів. З технічної точки зору рекомендується використовувати сучасні монітори з низьким рівнем випромінювання, що відповідають стандарту TCO та мають захисне покриття екрану [27]. Системний блок слід розміщувати на відстані не менше півметра від оператора, бічна та задня поверхні системного блоку випромінюють значно більше ніж передня панель монітора. Бездротові пристрої рекомендується замінювати провідними аналогами там, де це можливо оскільки Wi-Fi та Bluetooth адаптери є постійними джерелами випромінювання радіочастотного діапазону [25].

З організаційної точки зору ключовим заходом є дотримання режиму праці та відпочинку. Санітарні норми рекомендують робити перерву тривалістю десять-п'ятнадцять хвилин кожну годину роботи за комп'ютером [28]. Під час перерви рекомендується залишати робоче місце і виконувати легку фізичну активність або

вправи для очей, що сприяє відновленню функцій нервової системи та зниженню загального електромагнітного навантаження на організм. Робоче приміщення має регулярно провітрюватись оскільки іонізація повітря від електронного обладнання погіршує мікроклімат та підвищує стомлюваність [26].

Гранично допустимі рівні електромагнітних полів для робочих місць з комп'ютерною технікою регламентуються державними санітарними правилами і нормами ДСанПін 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [28]. Відповідно до цього документу напруженість електричного поля в діапазоні частот 5 Гц – 2 кГц не повинна перевищувати 25 В/м, а в діапазоні 2 кГц – 400 кГц не повинна перевищувати 2,5 В/м. Дотримання цих норм є обов'язковою умовою організації безпечного робочого місця оператора ПК.

Електромагнітні поля є реальним виробничим фактором, що впливає на здоров'я розробника програмного забезпечення. Комплексне застосування технічних засобів захисту у поєднанні з правильною організацією режиму праці та відпочинку дозволяє суттєво знизити негативний вплив електромагнітного випромінювання та зберегти працездатність фахівця на належному рівні [26].

4.2 Основи охорони праці

Організація робочого місця оператора персонального комп'ютера є одним з ключових факторів, що визначають продуктивність праці та збереження здоров'я фахівця в довгостроковій перспективі. Ергономіка, як наука вивчає взаємодію людини з технічними системами та робочим середовищем з метою оптимізації умов праці та підвищення її ефективності [29]. Правильно організоване робоче місце дозволяє мінімізувати фізичне та психологічне навантаження на оператора і знизити ризик розвитку професійних захворювань.

Вимоги до організації робочого місця оператора ПК регламентуються рядом нормативних документів. Основними з них є ДСанПін 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами

електронно-обчислювальних машин» [28], НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» та ДСТУ ISO 9241-5:2004 «Ергономічні вимоги до роботи з відеотерміналами в офісі» [31]. Ці документи встановлюють конкретні параметри, щодо розміщення обладнання, освітлення, мікроклімату та режиму праці.

Робоче місце розробника програмного забезпечення має відповідати ряду ергономічних вимог, щодо просторової організації. Площа на одне робоче місце з комп’ютером має становити не менше шести квадратних метрів, а об'єм приміщення не менше двадцяти кубічних метрів. Робочий стіл повинен мати висоту від 680 до 800 мм залежно від зросту оператора та достатню площу поверхні для розміщення монітора, клавіатури, миші та супутніх матеріалів без скупченості [29]. Робоче крісло має бути регульованим по висоті та куту нахилу спинки, підтримувати природне положення хребта під час роботи [31].

Особливої уваги заслуговує правильне розміщення монітора відносно оператора. Відстань від очей до екрану монітора має становити від 600 до 700 мм, ця відстань є оптимальною для мінімізації навантаження на орган зору [28]. Верхній край екрану монітора має знаходитись на рівні очей або дещо нижче, що дозволяє утримувати шию у природньому положенні без надмірного нахилу вперед або назад. Кут нахилу монітора відносно вертикальної площини не повинен перевищувати десяти – п’ятнадцяти градусів [31]. Монітор слід розміщувати таким чином, щоб виключити потрапляння прямих сонячних променів на екран та відбиття від поверхні, що спричиняє засвічування зображення та підвищує зорову втому.

Клавіатура розміщується на відстані 100–300 мм від переднього краю столу що забезпечує опору для передпліч оператора під час роботи [30]. Кут між передпліччям і плечем має становити близько дев’яноста градусів, таке положення мінімізує навантаження на зап’ястя та передпліччя і знижує ризик розвитку тунельного синдрому, що є поширеним професійним захворюванням серед програмістів [29]. Миша розміщується на одному рівні з клавіатурою і максимально близько до неї що дозволяє уникнути зайвих рухів рукою.

Освітлення робочого місця є важливим фактором, що впливає як на продуктивність праці так і на стан зору оператора. Природне освітлення є переважним і робоче місце слід орієнтувати таким чином, щоб вікна знаходились збоку від оператора переважно ліворуч, а не попереду або позаду що виключає засвічування екрану та небажані тіні [28].

Мікроклімат робочого приміщення безпосередньо впливає на самопочуття та продуктивність оператора. Оптимальна температура повітря в приміщенні, де встановлено комп'ютерну техніку має становити 21–23 градуси Цельсія в холодний та 22–24 градуси в теплий період року [30]. Відносна вологість повітря має підтримуватись в межах 40–60 відсотків, сухе повітря підвищує статичне електричне поле навколо монітора та спричиняє пересихання слизових оболонок [29]. Приміщення має провітрюватись не рідше двох разів на робочий день, що забезпечує достатній рівень кисню та знижує концентрацію озону, що виділяється при роботі електронного обладнання [30].

Режим праці та відпочинку є невід'ємною складовою охорони праці оператора ПК. Безперервна робота з комп'ютером не повинна перевищувати двох годин поспіль після чого обов'язкова перерва тривалістю не менше п'ятнадцяти хвилин [30]. Загальний час роботи з комп'ютером протягом робочого дня не повинен перевищувати шести годин для більшості категорій працівників [28]. Під час перерв рекомендується виконувати комплекс вправ для очей та розминку для м'язів шиї плечового поясу та спини що знімає м'язову напругу яка накопичується під час тривалої сидячої роботи [29].

Таким чином правильна організація робочого місця оператора персонального комп'ютера є комплексним завданням що охоплює просторове розміщення обладнання, параметри освітлення та мікроклімату і дотримання режиму праці та відпочинку. Виконання ергономічних вимог і санітарних норм дозволяє суттєво знизити ризик розвитку професійних захворювань та зберегти здоров'я і працездатність розробника програмного забезпечення протягом усього трудового стажу [31].

ВИСНОВКИ

У кваліфікаційній роботі вирішено задачу проєктування та розробки веб-платформи для оцінювання відеоігор «Ludara» з використанням технології Node.js. Розроблений програмний продукт є повнофункціональним веб-застосунком що реалізує повний цикл взаємодії користувача з відеоіграми.

У першому розділі проведено аналіз предметної області та розглянуто чотири основні існуючі платформи Metacritic, OpenCritic, RAWG та Steam. Виявлено що кожна з них вирішує лише частину потреб користувачів і має суттєві обмеження у функціональності. На основі проведеного аналізу сформульовано вісім конкретних завдань розробки, визначено чотири ролі акторів системи та описано ключові варіанти використання.

У другому розділі обрано ітеративну модель розробки як найбільш відповідну для індивідуального проєкту такого масштабу. Спроєктовано клієнт-серверну архітектуру на основі патерну Routes-Controller-Service з чітким розмежуванням відповідальності між компонентами. Розроблено схему бази даних що містить шість таблиць з відповідними зв’язками, унікальними обмеженнями та каскадним видаленням. Обрано та обґрунтовано технологічний стек Node.js з Express.js для серверної частини, React з Vite для клієнтської частини та PostgreSQL з Prisma ORM для роботи з даними. Реалізовано інтеграцію із зовнішніми сервісами RAWG API для отримання даних про ігри, Cloudinary для зберігання аватарів користувачів та JWT для автентифікації.

У третьому розділі проведено ручне функціональне тестування всіх модулів системи з розробкою детальних тестових сценаріїв, що охоплюють як стандартні так і граничні випадки використання. Загалом розроблено шість груп тестових сценаріїв що охоплюють модулі автентифікації, каталогу ігор, оцінювання та рецензування, статусів ігор, адміністративного керування та налаштувань профілю. За результатами верифікації підтверджено що платформа відповідає всім восьми функціональним вимогам визначеним на початку розробки.

У результаті виконання кваліфікаційної роботи розроблено повнофункціональну веб-платформу що реалізує такі можливості. Система автентифікації на основі JWT токенів забезпечує безпечну реєстрацію та вхід користувачів з автоматичним розлогіненням при протермінуванні токена. Каталог ігор з інтеграцією RAWG API надає доступ до бази даних відеоігор з підтримкою пошуку за назвою, фільтрації за жанром та роком релізу і сортування за різними критеріями. Система оцінювання дозволяє виставляти числові оцінки за шкалою від 1 до 10 та писати текстові рецензії з можливістю їх оновлення та видалення. Система статусів ігор забезпечує ведення особистої ігрової бібліотеки з трьома станами. Розмежування ролей між звичайними користувачами, критиками та адміністраторами забезпечує структуровану взаємодію спільноти. Адміністративна панель надає повний набір інструментів для керування користувачами та моніторингу статистики платформи. Публічні профілі користувачів відображають детальну статистику активності включно з середньою оцінкою, кількістю рецензій, отриманими вподобаннями та улюбленими жанрами.

Практична цінність розробленої платформи підтверджується тим, що всі реалізовані функції пройшли успішне ручне функціональне тестування та коректно взаємодіють між собою в рамках єдиної системи. Розглянуто підходи до автоматизованого тестування через Jest, Supertest та навантажувального тестування через Artillery і k6, що може бути впроваджено при подальшому масштабуванні платформи. Розроблена архітектура є масштабованою і може бути розширена додаванням нових функціональних модулів, системи рекомендацій, соціальних функцій або мобільного застосунку. Платформа може бути розгорнута у виробничому середовищі з використанням хмарних сервісів, що робить її доступною для широкої аудиторії користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 411 p. URL: https://ieeecs-media.computer.org/media/education/swebok/swebok-v4.pdf

2. Node.js Documentation. OpenJS Foundation, 2024. URL: https://nodejs.org/en/docs

3. Express.js Documentation. OpenJS Foundation, 2024. URL: https://expressjs.com

4. React Documentation. Meta Platforms, Inc., 2024. URL: https://react.dev

5. Prisma Documentation. Prisma Data, Inc., 2024. URL: https://www.prisma.io/docs

6. PostgreSQL Documentation. The PostgreSQL Global Development Group, 2024. URL: https://www.postgresql.org/docs

7. RAWG Video Games Database API Documentation. RAWG, 2024. URL: https://rawg.io/apidocs

8. Cloudinary Documentation. Cloudinary Ltd., 2024. URL: https://cloudinary.com/documentation

9. M. Jones, J. Bradley, N. Sakimura. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519

10. Vite Documentation. Evan You and Vite Contributors, 2024. URL: https://vitejs.dev

11. React Router Documentation. Remix Software, Inc., 2024. URL: https://reactrouter.com

12. Axios Documentation. Matt Zabriskie and contributors, 2024. URL: https://axios-http.com/docs/intro

13. TanStack Query Documentation. Tanner Linsley, 2024. URL: https://tanstack.com/query/latest/docs

14. Zod Documentation. Colin McDonnell, 2024. URL: https://zod.dev

15. R. Fielding. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation. University of California, Irvine, 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

16. S. Newman. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 612 p.

17. R. C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

18. M. Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560 p.

19. OWASP Foundation. OWASP Top Ten. 2021. URL: https://owasp.org/www-project-top-ten

20. bcrypt Documentation. npm, 2024. URL: https://www.npmjs.com/package/bcrypt

21. Multer Documentation. npm, 2024. URL: https://www.npmjs.com/package/multer

22. React Hot Toast Documentation. Timo Lins, 2024. URL: https://react-hot-toast.com

23. Zustand Documentation. pmndrs, 2024. URL: https://zustand-demo.pmnd.rs

24. Атаманчук П.С. Безпека життєдіяльності: навч. посіб. Київ: Центр учбової літератури, 2020. 276 с.

25. Мелех Л.В. Безпека життєдіяльності та охорона праці: навч. посіб. Львів: ЛДУ внутрішніх справ, 2022. 219 с.

26. Безпека життєдіяльності та охорона праці: підруч. / В.В. Сокуренко, О.М. Бандурка та ін. Харків: ХНУВС, 2021. 308 с.

27. Желібо Є.П. Безпека життєдіяльності: підручник / В.В. Зацарний. Київ: Каравела, 2023. 344 с.

28. ДСанПін 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин».

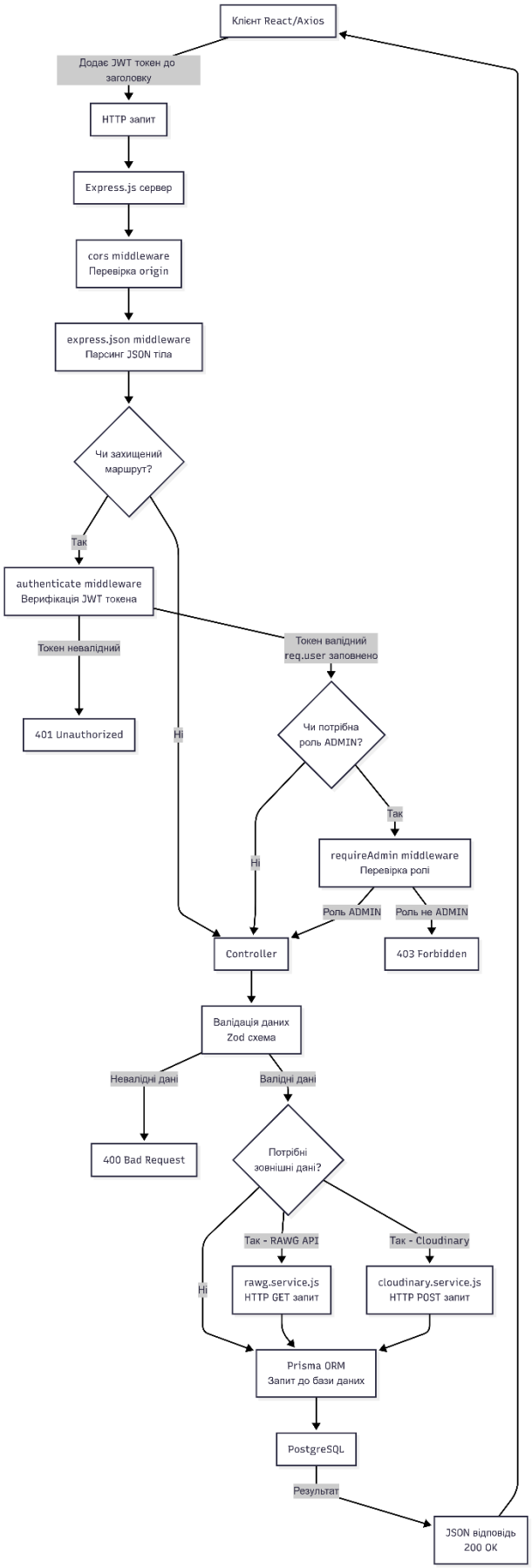
29. Жидецький В.Ц. Охорона праці користувачів комп'ютерів: підручник. Львів: Афіша, 2020. 176 с.

30. НПАОП 0.00-7.15-18 Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, від 14.02.2018 року №207.

31. ДСТУ ISO 9241-5:2004 Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до компонування робочого місця та до робочої пози.

ДОДАТКИ

Додаток А – Рисунок механізму обробки HTTP запиту



Додаток Б – Лістинг коду функції «authenticate»

```
const authenticate = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'No token provided' });
  }
  const token = authHeader.slice(7);
  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET);
    req.user = payload;
    next();
  } catch {
    return res.status(401).json({ message: 'Invalid or expired
token' });
  }
};
```

Додаток В – Лістинг коду функції «ensureGameExists»

```
const ensureGameExists = async (rawgId) => {
  let game = await prisma.game.findUnique({ where: { rawgId } });

  if (!game) {
    const data = await rawg.getGameById(rawgId);
    game = await prisma.game.create({
      data: {
        rawgId: data.id,
        title: data.name,
        coverUrl: data.background_image || null,
        description: data.description_raw || null,
        releaseYear: data.released ? new
Date(data.released).getFullYear() : null,
        genres: data.genres?.map((g) => g.name).join(', ') || null,
      },
    });
  }

  return game;
};
```

Додаток Д – Лістинг коду функції «banUser»

```
const banUser = async (req, res) => {
  const id = parseInt(req.params.id);
  if (isNaN(id)) return res.status(400).json({ message: 'Invalid
user id' });

  const result = banSchema.safeParse(req.body);
  if (!result.success) {
    return res.status(400).json({ message:
result.error.issues[0].message });
  }

  const { reason, banUntil } = result.data;

  const user = await prisma.user.findUnique({ where: { id } });
  if (!user) return res.status(404).json({ message: 'User not found'
});
  if (user.role === 'ADMIN') {
    return res.status(403).json({ message: 'Cannot ban an admin' });
  }

  const banUntilDate = banUntil ? new Date(banUntil) : null;

  await prisma.$transaction([
    prisma.user.update({
      where: { id },
      data: { isBanned: true, banUntil: banUntilDate, banReason:
reason },
    }),
    prisma.ban.create({
      data: { userId: id, adminId: req.user.id, reason, banUntil:
banUntilDate },
    }),
  ]);

  return res.json({
    message: banUntilDate ? `User banned until
${banUntilDate.toISOString()}` : 'User permanently banned',
  });
};
```

Додаток Е – Лістинг коду функції «filterGames»

```
const BLOCKED_WORDS = [
  'porn', 'hentai', 'xxx', 'nude', 'naked', 'adult', 'erotic',
  'sex', 'nsfw', 'lewd', 'ecchi', '18+', 'mega pack', 'porn pack',
];
const BLOCKED_TAGS = [
  'nsfw', 'sexual-content', 'sex', 'erotic', 'adult', 'hentai',
'porn', 'nude', 'nudity',
];
const JUNK_WORDS = [
  'wiki', 'guide', 'tutorial', 'mod ', 'how to',
  'skin studio', 'papercraft', 'basics in',
];

const filterGames = (games) => games.filter((game) => {
  const name = (game.name || '').toLowerCase();
  if (BLOCKED_WORDS.some((w) => name.includes(w))) return false;
  if (Array.isArray(game.tags) && game.tags.some((t) =>
BLOCKED_TAGS.includes(t.slug))) return false;
  if (JUNK_WORDS.some((w) => name.includes(w))) return false;
  if (game.parent_game != null) return false;
  if (!Array.isArray(game.genres) || game.genres.length === 0)
return false;
  return true;
});
```

Додаток Ж – Лістинг коду функції «getGames»

```
const getGames = async (params = {}) => {
  const pageSize = parseInt(params.page_size) || 20;
  const startPage = parseInt(params.page) || 1;

  const base = {
    exclude_additions: true,
    platforms: '4,1,186,187,18,16',
    page_size: pageSize,
  };
  if (params.search)   base.search = params.search;
  if (params.genres)   base.genres = params.genres;
  if (params.ordering) base.ordering = params.ordering;
  if (params.dates)    base.dates = params.dates;

  const collected = [];
  let rawgPage = startPage;
  let rawgNext = null;
  let totalCount = null;

  while (collected.length < pageSize) {
    const data = await rawgFetch('/games', { ...base, page: rawgPage
});
    const batch = data.results || [];
    rawgNext = data.next || null;
    if (totalCount === null && data.count) totalCount = data.count;
    collected.push(...filterGames(batch));
    if (!rawgNext || batch.length === 0) break;
    rawgPage++;
  }

  const hasNext = collected.length > pageSize || !!rawgNext;

  return {
    results: collected.slice(0, pageSize),
    next: hasNext ? 'yes' : null,
    count: totalCount,
  };
};
```

Додаток З – Лістинг коду схеми валідації «Zod»

```
// auth.controller.js
const registerSchema = z.object({
  nickname: z.string()
    .min(3, 'Nickname must be at least 3 characters')
    .max(50, 'Nickname must be at most 50 characters'),
  email: z.string().email('Invalid email address'),
  password: z.string().min(8, 'Password must be at least 8
characters'),
});

const loginSchema = z.object({
  email: z.string().email('Invalid email address'),
  password: z.string().min(1, 'Password is required'),
});

// ratings.controller.js
const rateSchema = z.object({
  rawgId: z.number().int().positive('rawgId must be a positive
integer'),
  score: z.number().int().min(1, 'Score must be at least 1').max(10,
'Score must be at most 10'),
  reviewText: z.string().max(5000, 'Review must be at most 5000
characters').nullish(),
});

// statuses.controller.js
const setStatusSchema = z.object({
  rawgId: z.number().int().positive('rawgId must be a positive
integer'),
  status: z.enum(['WANT', 'PLAYING', 'COMPLETED'], {
    errorMap: () => ({ message: 'Status must be WANT, PLAYING, or
COMPLETED' }),
  }),
});

// admin.controller.js
const updateRoleSchema = z.object({
  role: z.enum(['USER', 'CRITIC'], {
    errorMap: () => ({ message: 'Role must be USER or CRITIC' }),
  }),
});

const banSchema = z.object({
  reason: z.string().min(1, 'Reason is required'),
  banUntil: z.string()
    .datetime({ message: 'banUntil must be a valid ISO datetime' })
    .optional(),
});
```