Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра
на тему: «Розробка веб-застосунку для вибору відеоігор»

Виконав: студент групи ІПЗ21-2
Спеціальність 121 «Інженерія програмного забезпечення»
Вовчик Єгор
(прізвище та ініціали)
Керівник к.т.н., доц. Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент Дніпровського державного технічного університету
(місце роботи)
Доцент кафедри програмного забезпечення систем
(посада)
к.т.н., доцент Косухіна О.С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Вовчик Є. Розробка веб-застосунку для вибору відеоігор.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра за спеціальністю 121 «Інженерія програмного забепечення» – Університет митної справи та фінансів, Дніпро, 2025.

Кваліфікаційна робота присвячена розробці веб-застосунку для вибору відеоігор. З огляду на зростаючу кількість відеоігор користувачі потребують інструментів, які дозволяють ефективно орієнтуватися в доступному контенті та приймати обґрунтовані рішення щодо вибору.

У процесі дослідження було проаналізовано сучасні платформи, сервіси й методи добору ігор, а також засоби реалізації веб-застосунків. Обґрунтовано вибір технологій React.js, Node.js та Express.js для реалізації клієнтської та серверної частин, а також бази даних SQLite для зберігання ігрової інформації та оцінок користувачів.

У результаті реалізовано функціональний веб-застосунок, який дозволяє здійснювати фільтрацію, перегляд інформації про ігри та оцінювання, з урахуванням базових елементів персоналізованого добору. Застосунок адаптивний, доступний із різних пристроїв та має потенціал для подальшого розвитку як повноцінної рекомендаційної системи.

Результати роботи можуть бути використані для впровадження аналогічних рішень у сфері цифрового контенту, зокрема онлайн-магазинів та геймерських спільнот.

Ключові слова: веб-застосунок, вибір відеоігор, персоналізація, React.js, Node.js, рекомендаційна система.

ABSTRACT

Vovchyk Ye. Development of a Web Application for Video Game Selection.

Bachelor's thesis for obtaining a degree in Software Engineering, specialty 121. – University of Customs and Finance, Dnipro, 2025.

This qualification thesis is devoted to the development of a web application for video game selection. Given the growing number of video games, users require tools that allow them to efficiently navigate available content and make informed decisions about their choices.

The research includes an analysis of modern platforms, services, and methods for game selection, as well as tools for implementing web applications. The choice of technologies – React.js, Node.js, and Express.js – for the client-side and server-side parts, along with the SQLite database for storing game information and user ratings, is justified.

As a result, a functional web application was developed, enabling filtering, viewing information about games, and rating them, taking into account basic personalization features. The application is adaptive, accessible from various devices, and has the potential to evolve into a fully-fledged recommendation system.

The results of the work can be used to implement similar solutions in the field of digital content, particularly in online stores and gaming communities.

Keywords: web application, video game selection, personalization, React.js, Node.js, recommendation system.

ЗМІСТ

ВСТУП 5
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1. Опис предметної області 7
1.2. Аналіз існуючих платформ та систем рекомендацій ігор 8
1.3 Аналіз методів та підходів реалізації веб-застосунків 12
1.4 Висновок до першого розділу 15
РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ 16
2.1 Вибір програмних засобів для реалізації проекту 16
2.2. Засоби для розробки Front-end частини 19
2.3. Засоби для розробки Back-end частини 21
2.4 Висновок до другого розділу 26
РОЗДІЛ 3. СТВОРЕННЯ ВЕБ-ДОДАТКУ ДЛЯ ВИБОРУ ВІДЕОГОР 28
3.1 Актуальність створення веб-додатку 28
3.2 Інструменти розробки додатку 30
3.3 Розробка веб-додатку 37
3.4 Тестування додатку 50
3.5 Висновок до третього розділу 57
ВИСНОВОК 58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60

ВСТУП

Актуальність проблеми. Сучасна індустрія відеоігор характеризується стрімким зростанням кількості ігор та розмаїттям жанрів, що ускладнює процес вибору відповідної гри для користувача. Користувачі часто витрачають багато часу на пошук ігор, які відповідали б їхнім вподобанням. Важливо створювати зручні інструменти, які дозволяють автоматизувати процес добору ігор та покращити користувацький досвід. Виникає потреба у створенні прикладних засобів – рекомендаційних систем, здатних не лише зберігати й обробляти великі масиви даних про ігрові продукти, але й надавати користувачеві релевантні рекомендації з урахуванням його індивідуальних уподобань та ігрового досвіду. Рекомендаційні системи є важливим інструментом в електронній комерції та інтернет-сервісах, допомагаючи користувачам знайти продукти або послуги, які їм, найімовірніше, сподобаються, на основі їхніх попередніх взаємодій, відгуків або поведінки. Вони вирішують проблему перевантаження інформацією, фільтруючи важливі фрагменти з великої кількості динамічно генерованої інформації відповідно до уподобань користувача.

Веб-застосунки, що включають елементи персоналізації, є ефективним засобом вирішення цієї задачі, оскільки забезпечують доступність з будь-якого пристрою, простоту оновлення та інтеграцію з зовнішніми сервісами. Застосування сучасних веб-технологій, таких як React.js, Node.js та Express.js, дозволяє створювати продуктивні, масштабовані та адаптивні інтерфейси з можливістю подальшого розширення функціоналу.

Метою роботи є автоматизація вибору відеоігор шляхом створення веб-застосунку.

Методи дослідження: обробка та аналіз інформації, методи проектування та розробки веб-додатків.

У відповідності до поставленої мети в кваліфікаційній роботі поставлені наступні *завдання дослідження*:

1. Проаналізувати предметну область та платформи вибору відеоігор.

2. Дослідити сучасні засоби та підходи до створення веб-застосунків.
3. Обґрунтувати вибір інструментів для реалізації проєкту.
4. Розробити архітектуру та інтерфейс веб-застосунку.
5. Реалізувати функціональний прототип веб-застосунку.
6. Провести тестування основних функцій.

Об'єктом дослідження є процес взаємодії користувача з системами вибору цифрового контенту.

Предметом дослідження є апаратно-програмне забезпечення для розробки веб-додатків.

Застосунок, який необхідно розробити повинен обробляти асинхронні запити й адаптуватися під зміни умов, забезпечувати інтуїтивне взаємодію з інтерфейсом та надійну backend-логіку.

Практичне значення: результат кваліфікаційної роботи може бути використаним в веб-середовищі й використовуватися онлайн-магазинами ігор чи геймерськими спільнотами для підвищення задоволеності користувачів і покращення монетизації за рахунок персоналізованих рекомендацій

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел з 15 найменувань. Загальний обсяг роботи – 63 сторінки, 26 рисунків, 4 таблиці.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області

Сучасний світ характеризується стрімким розвитком інформаційних технологій, що призвело до безпрецедентного зростання та диверсифікації ринку відеоігор. За оцінками, щорічний обсяг глобального ринку відеоігор сягає сотень мільярдів доларів, а кількість гравців по всьому світу налічує мільярди. Відеоігри трансформувалися з простого розважального контенту до складних інтерактивних творів, що охоплюють широкий спектр жанрів, сетінгу та платформ.

Цей інтенсивний розвиток ринку та постійне розширення асортименту відеоігор супроводжується значним зростанням вимог до персоналізації контенту та підвищенням рівня конкуренції серед розробників. Зі збільшенням кількості доступних ігор, користувачі все частіше стикаються з проблемою вибору, яку гру спробувати наступною, як знайти нові ігри, що відповідають їхнім інтересам, або як орієнтуватися у величезному обсязі нових проєктів. Нерідко виникає ситуація, коли геймери не знають, в яку відеогру їм пограти, або ж, згідно зі звітами, не грали у значну частину придбаних ігор. Це підкреслює актуальність ефективної організації процесу добору ігор та необхідність оптимізації часу користувача на пошук нового контенту [3].

Саме в цих умовах виникає потреба у створенні прикладних засобів – рекомендаційних систем, здатних не лише зберігати й обробляти великі масиви даних про ігрові продукти, але й надавати користувачеві релевантні рекомендації з урахуванням його індивідуальних уподобань та ігрового досвіду. Рекомендаційні системи є важливим інструментом в електронній комерції та інтернет-сервісах, допомагаючи користувачам знайти продукти або послуги, які їм, найімовірніше, сподобаються, на основі їхніх попередніх взаємодій, відгуків або поведінки. Вони вирішують проблему перевантаження інформацією, фільтруючи важливі фрагменти з великої кількості динамічно генерованої інформації відповідно до уподобань користувача.

Впровадження таких систем має значне практичне значення, оскільки дозволяє підвищити задоволеність користувачів, покращити їхній ігровий досвід, оптимізувати витрати часу та коштів, а також може призвести до збільшення рівня монетизації для постачальників послуг за рахунок покращеної точності рекомендацій [1], [2].

1.2. Аналіз існуючих платформ та систем рекомендацій ігор

Сучасний ринок відеоігор представлений численними онлайн-платформами та магазинами, які не тільки надають доступ до широкого асортименту ігор, але й активно впроваджують власні рекомендаційні системи. Аналіз цих існуючих рішень дозволяє виявити їхні сильні та слабкі сторони, а також визначити потенційну нішу для нового веб-застосунку.

1.2.1. Огляд популярних платформ та магазинів ігор

Серед найвпливовіших гравців на ринку цифрової дистрибуції відеоігор та рекомендаційних систем варто виділити наступні:

- Steam – це одна з найбільших платформ для цифрової дистрибуції відеоігор, розроблена компанією Valve. Вона пропонує ігри для ПК (Windows, MacOS, Linux) та Steam Deck. Рекомендаційна система Steam базується на машинному навчанні та використовує колаборативну фільтрацію. Вона аналізує, в які ігри грає користувач та інші гравці зі схожими звичками, щоб робити обґрунтовані пропозиції. На рекомендації Steam також впливають мітки ігор (наприклад, екшн, для одного гравця), відгуки користувачів, популярність ігор на даний момент, а також наявність ігор у друзів та в списку лідерів продажів.
- Epic Games Store – це ще одна велика платформа для цифрового розповсюдження відеоігор, власником якої є компанія Epic Games, розробник популярної відеогри Fortnite. Пропонує ігри для ПК (Windows, MacOS, Linux). Її рекомендаційна система, як і у Steam, також переважно ґрунтується на

колаборативній фільтрації, використовуючи великі обсяги даних про вподобання користувачів.

- PlayStation Store – це магазин відеоігор для користувачів PlayStation від японської корпорації Sony. Він пропонує ігри для консолей PlayStation. Рекомендаційні системи, такі як у PlayStation Store, базуються на колаборативній фільтрації, враховуючи історію покупок та взаємодії користувачів.
- Xbox Store – цифровий магазин відеоігор від Microsoft. Пропонує ігри для консолей Xbox та ПК (Windows). Його рекомендаційні системи також орієнтовані на колаборативну фільтрацію, що дозволяє прогнозувати вподобання користувачів на основі поведінки широкої аудиторії.
- GOG.com – це цифрова платформа розповсюдження відеоігор, відома своєю політикою проти DRM (захист від копіювання). Пропонує ігри для PC-платформи.
- Nintendo eShop – офіційний магазин для купівлі цифрових ігор на консолях Nintendo.

1.2.2. Огляд спеціалізованих веб-сайтів для порівняння та рекомендацій

Окрім великих дистриб'юторів ігор, існують спеціалізовані веб-сайти, які зосереджені на інших аспектах вибору ігор, зокрема на порівнянні системних вимог та комплектуючих ПК:

- Games Finder – це вебсайт, що надає рекомендації відеоігор, які базуються на змісті. Він використовує алгоритми фільтрації за змістом для підбору ігор, які користувачу можуть сподобатися на основі тих ігор, які він вже грав і оцінив.
- RAWG та Internet Game Database (IGDB) – це великі бази даних відеоігор з вбудованими рекомендаційними системами, які можуть використовувати різні підходи, включаючи контент-базовані.
- BenchGame – сайт, який допомагає знайти та вибрати ігри для ПК, а також перевірити, чи підходить ПК до системних вимог гри та кількість FPS.

Однак, він не показує приблизну кількість FPS, а лише рекомендовані налаштування графіки.

- Technical.city – веб-сайт, що надає інформацію для порівняння процесорів та відеокарт, а також приблизну кількість FPS у ігрових проєктах. Його недоліком є те, що при розрахунку FPS враховується лише потужність відеокарти, без урахування процесора.
- Overclockers.ua, Chaynikam.info та SysRqmts.com – ці ресурси також пропонують порівняння характеристик відеокарт та процесорів, а також дані щодо FPS у різних іграх. Загальним недоліком подібних сайтів є нечіткі та складні для розуміння інтерфейси з багатьма елементами [4].

Таблиця 1.1.

Порівняльний аналіз існуючих рішень для вибору ігор

Критерій порівняння	Steam/Epic Games Store/ PlayStation Store/Xbox Store	Games Finder/ RAWG/ IGDB	BenchGame	Technical .city
Основний функціонал	Продаж та дистрибуція ігор. Вбудовані рекомендаційні системи.	Рекомендації ігор на основі схожості, бази даних ігор	Перевірка системних вимог ПК для ігор, рейтинг ігор, трейлери, новини	Порівняння характеристик процесорів та відеокарт, приблизний FPS
Механізм рекомендацій	Колаборативна фільтрація	Контент-базова фільтрація	Порівняння конфігурації ПК з мінімальними/рекомендованими системними вимогами гри	Розрахунок FPS з акцентом лише на відеокарту
Точність рекомендацій FPS	Не застосовується	Не застосовується	Обмежена, лише рекомендовані налаштування графіки, без конкретних значень FPS	Обмежена, враховується тільки відеокарта
Прозорість рекомендацій	Алгоритми є комерційною таємницею	Залежить від реалізації, зазвичай прозорість вища	Не застосовується	Обмежена

Взаємодія з користувачем	Реєстрація, купівля, встановлення, відгуки	Пошук, фільтрація, перегляд інформації	Введення комплектуючих, вибір гри, перегляд звіту	Введення комплектуючих, перегляд FPS
Інтерфейс користувача	Розвинений, часто насичений функціоналом, може бути перевантаженим	Різний, деякі можуть бути складними для навігації	Може бути важким для навігації	Може бути важким для навігації
Проблема "холодного старту"	Присутня для нових користувачів та нових ігор	Менш виражена для нових елементів, але залежить від наявності атрибутів	Не застосовується напряму	Не застосовується напряму
Обсяг даних та джерела	Власні величезні бази даних користувацьких взаємодій	Збирають дані з відкритих джерел	Власні бази даних комплектуючих та ігор	Власні бази даних комплектуючих та ігор

1.2.3. Визначення ніші для розроблюваного веб-застосунку

Аналіз існуючих рішень виявляє, що попри їхній функціонал, залишаються невирішені проблеми, які можуть бути подолані в рамках даної дипломної роботи. Зокрема, розроблюваний веб-застосунок має потенціал заповнити нішу, пропонуючи:

- Інтуїтивно зрозумілий дизайн та наочне порівняння – проєкт прагне усунути проблему нечітких та складних для навігації інтерфейсів, які мають деякі аналоги. Буде реалізовано інтуїтивно зрозумілий дизайн та зручне, наочне порівняння процесорів та відеокарт, із зазначенням, яка саме характеристика краща та наскільки у відсотках.

Можливість доповнення бази даних користувачами – можливість доповнення бази даних інформацією користувачів безпосередньо через сайт. Це

дозволить у реальному часі оновлювати інформацію, покращуючи точність даних, та вирішить проблему нестачі даних для деяких комплектуючих.

Адаптивність – веб-застосунок буде адаптивним для різних розмірів екранів, що полегшить навігацію та зробить користування сайтом комфортнішим для користувачів з різних пристроїв (ПК, планшети, смартфони) [7].

1.3 Аналіз методів та підходів реалізації веб-застосунків

Розробка сучасних веб-застосунків вимагає глибокого розуміння різноманітних архітектурних підходів, методологій та технологій, що забезпечують їхню ефективність, масштабованість, безпеку та зручність у використанні. Обрані підходи суттєво впливають на життєвий цикл програмного продукту, від початкового планування до подальшого супроводу.

1.3.1. Архітектури веб-застосунків

Веб-додатки функціонують за принципом клієнт-серверної архітектури, яка передбачає логічне розділення функцій між серверною та клієнтською частинами додатку. Це рішення сприяє високій прогнозованості, можливості розширення та зручності підтримки проєкту. Архітектура проекту:

1) Клієнтська частина (Frontend).
2) Серверна частина (Backend).
3) База даних.

Окремо виділяються такі архітектурні підходи:

– Монолітна архітектура – це коли весь функціонал додатку реалізовано в рамках єдиного коду та розгортається як один цілий компонент. Це спрощує розробку на початкових етапах, але може ускладнювати масштабування та підтримку великих проєктів.

– Мікросервісна архітектура – додаток розбивається на невеликі, незалежні сервіси, кожен з яких виконує певну бізнес-функцію і може бути

розроблений, розгорнутий та масштабований окремо. Цей підхід забезпечує більшу гнучкість, але додає складності в управлінні та комунікації між сервісами.

– Clean Architecture – цей підхід передбачає розбиття проєкту на декілька шарів з чітко визначеними залежностями: зовнішні шари залежать від внутрішніх, але не навпаки. Це забезпечує гнучкість, легкість тестування та можливість заміни зовнішніх компонентів, наприклад бази даних або UI, без значного перепроєктування основної логіки. Цей підхід дозволяє легко створювати новий функціонал з мінімальною кількістю змін у вже існуючому коді [9].

1.3.2. Методології розробки веб-застосунків

У процесі розробки веб-застосунків застосовуються різні методології, які структуруються відповідно до фаз життєвого циклу програмного забезпечення:

1) Методологія Fournie – зосереджується на розробці інформаційної архітектури та проєктуванні технічної архітектури системи. Використовує спільні фасилітовані сесії для визначення вимог користувачів. Не розглядає стратегічні та бізнес-аспекти проєкту.

2) Методологія проектування інтрамережі (IDM) – зосереджена на розробці додатків для внутрішніх мереж і включає 10 етапів: техніко-економічне обґрунтування, збір вимог, проєктування, реалізацію та тестування.

3) Методологія розробки веб-сайтів (Howcroft & Carroll) – розглядає розробку веб-сайтів як системний підхід, що складається з фаз аналізу (вимоги користувачів, контекст проєкту, технологічний та інформаційний аналіз), дизайну (структура сторінок, взаємодія елементів, графічний дизайн) та реалізації (розробка веб-сайту).

4) Методологія розширення веб-додатків (Web Application Extension - WAE) – використовує об'єктно-орієнтований підхід до розробки програмного забезпечення для веб-додатків. Акцентує на розширенні та повторному

використанні програмних компонентів , а також має еволюційну перспективу. Включає етапи аналізу вимог, проєктування, реалізації, тестування, впровадження, підтримки та розвитку [6].

1.3.3. Загальні вимоги до веб-додатків для вибору ігор

Для успішної реалізації веб-застосунку для вибору ігор та забезпечення його конкурентоспроможності, необхідно врахувати наступні ключові вимоги:

1) Доступність та кросплатформенність:

Застосунок має бути доступним з різних пристроїв та операційних систем, що забезпечується адаптивним дизайном.

2) Зручність користувацького інтерфейсу (UI/UX)

Дизайн має бути інтуїтивно зрозумілим, привабливим та легким у використанні, що сприяє високій задоволеності користувачів.

3) Швидкодія та стабільність

Застосунок повинен забезпечувати швидке завантаження сторінок та оперативну обробку запитів, що важливо для комфортного користувацького досвіду.

4) Можливість інтеграції

Гнучка архітектура має дозволяти інтеграцію із зовнішніми джерелами даних (API) та іншими сервісами.

5) Безпека та конфіденційність

Обов'язковою є забезпечення безпеки та конфіденційності обробки та передачі інформації, особливо користувацьких даних.

6) Масштабованість та гнучкість

Система повинна бути спроєктована таким чином, щоб її функціональність та можливості могли бути легко розширені, а також забезпечувалася підтримка зростаючої кількості користувачів.

7) Актуалізація даних

Для рекомендаційних систем критично важлива актуальність даних. Можливість користувачів вносити власні дані значно покращить точність та повноту інформації [7].

Враховуючи ці аспекти, розробка веб-застосунку для вибору ігор є комплексним процесом, що вимагає системного підходу, обґрунтованого вибору технологій та дотримання передових методологій розробки.

1.4 Висновок до першого розділу

У першому розділі проведено всебічний аналіз існуючих рішень для пошуку та рекомендації відеоігор. Розглянуто можливості провідних цифрових магазинів-платформ разом зі спеціалізованими веб-сайтами, виявлено їхні сильні та слабкі сторони.

Проаналізовано методи й підходи до побудови рекомендаційних систем та проаналізовано їх застосування на різних платформах. Розглянуто архітектурні рішення для розробки веб-застосунків, методології повного життєвого циклу програмного забезпечення, що забезпечують масштабованість, тестованість і можливість розширення функціоналу . Сформульовано загальні нефункціональні вимоги.

Таким чином, проведений аналіз підтверджує доцільність створення інтегрованої платформи, яка поєднає прозорий алгоритм персоналізованих рекомендацій, інтуїтивно зрозумілий інтерфейс та краудсорсинговий механізм оновлення бази даних. Отримані результати визначають вимоги до архітектури й вибору технологій для розробки системи.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ

2.1 Вибір програмних засобів для реалізації проекту

Обґрунтований вибір технологічного стеку є фундаментальним етапом у процесі розробки веб-застосунку, оскільки він безпосередньо впливає на його ефективність, масштабованість, зручність супроводу та відповідність сучасним вимогам програмної інженерії. На сучасному ринку інформаційних технологій представлено широкий спектр інструментів та фреймворків, призначених для реалізації клієнтської, серверної частин веб-додатків та управління базами даних.

2.1.1. Архітектури веб-застосунків

Веб-додатки функціонують за принципом клієнт-серверної архітектури, яка передбачає логічне розділення функцій між серверною та клієнтською частинами додатку. Це рішення сприяє високій прогнозованості, можливості розширення та зручності підтримки проєкту.

1) Клієнтська частина (Frontend) – являє собою графічний інтерфейс, що відображається в браузері користувача. Відповідає за безпосередню взаємодію з користувачем, динамічне відображення даних та формування запитів до сервера.

2) Серверна частина (Backend) – це програма, що виконується на віддаленому комп'ютері (сервері). Вона відповідає за обробку запитів клієнта, взаємодію з базою даних, реалізацію бізнес-логіки та формування відповіді для клієнта.

3) База даних – програмне забезпечення на сервері, що забезпечує систематизоване зберігання, управління та видачу даних за запитом.

Сучасні веб-застосунки часто використовують багатошарову архітектуру, що ділить додаток на логічні та фізичні рівні. Кожен рівень несе певну

відповідальність: вищий рівень може використовувати послуги нижчого, але не навпаки. Традиційний трирівневий додаток має рівень презентації, середній рівень і рівень бази даних. Складніші програми можуть мати більше рівнів.

2.1.2. Основні патерни проектування веб-додатків

У контексті розробки веб-застосунків важливе місце посідають патерни проектування, які надають перевірені рішення для типових задач:

1) Model-View-Controller (MVC) – є архітектурним шаблоном, що дозволяє розділити логіку додатку на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Кожен з цих компонентів виконує свою відповідальність і співпрацює з іншими для створення функціонального та добре організованого веб-додатку. Модель відповідає за бізнес-логіку та дані, представлення – за відображення даних користувачу, а контролер – за обробку запитів та взаємодію з моделлю та представленням.

2) Model-View-ViewModel (MVVM) – патерн розділяється на три частини: Model (доменна модель, частина бізнес-логіки), View (графічний інтерфейс), та ViewModel (модель, яка тісно пов'язана з View, та оповіщає представлення про зміни). MVVM дозволяє ефективно розробляти UI застосунки зі зрозумілою архітектурою, розділяючи графічний інтерфейс на окремі частини.

3) Command and Query Responsibility Segregation (CQRS) – патерн, що дозволяє розділити операції читання (Queries) та модифікації (Commands) даних. Команди виконують операції вставки, оновлення та видалення, тоді як запити – операції виведення інформації.

Таблиця 2.1.

Порівняльний аналіз патернів проектування веб-застосунків

Назва патерну	Призначення	Переваги	Недоліки
Model-View-Controller (MVC)	Розділення логіки додатку на три основні компоненти: модель, представлення та контролер.	Сприяє чіткому розділенню відповідальності, покращує модульність, полегшує тестування та супровід коду.	Може бути складнішим для невеликих проєктів; вимагає знання патерну та його принципів.
Model-View-ViewModel (MVVM)	Розділення UI-логіки застосунку на три частини: Model (доменна модель), View (графічний інтерфейс) та ViewModel (модель представлення).	Дозволяє ефективно розробляти UI-застосунки зі зрозумілою архітектурою, розділяючи графічний інтерфейс на окремі частини. Сприяє легшому тестуванню UI-логіки, оскільки ViewModel є незалежною від View.	Може додавати зайву складність для простих UI; вимагає використання механізмів прив'язки даних.
Command and Query Responsibility Segregation (CQRS)	Розділення операцій читання (Queries) та модифікації (Commands) даних.	Дозволяє застосувати принцип єдиної відповідальності (Single Responsibility Principle), де кожна дія має окремий клас. Оптимізує продуктивність для систем з високим навантаженням на читання та запис.	Збільшує складність архітектури, вимагає додаткового керування консистентністю даних між моделями читання та запису.
Посередник (Mediator)	Інкапсулює спосіб взаємодії об'єктів, запобігаючи прямим зв'язкам між ними.	Зменшує залежності між об'єктами, сприяє повторному використанню компонентів та полегшує супровід коду. Особливо гарно поєднується з патерном CQRS.	Може стати "вузьким місцем" або занадто складним, якщо посередник починає керувати занадто великою кількістю взаємодій.

2.2. Засоби для розробки Front-end частини

Клієнтська частина веб-застосунку відповідає за безпосередню взаємодію з користувачем, візуалізацію даних та формування користувацького досвіду. Її реалізація вимагає використання сучасних веб-технологій, що забезпечують динамічний, інтерактивний та адаптивний інтерфейс. На сучасному ринку представлено широкий спектр інструментів для фронтенд-розробки, кожен з яких характеризується унікальними особливостями, що обумовлює необхідність їхнього всебічного аналізу для обґрунтованого вибору. Серед найбільш популярних на ринку варто виділити React.js, Angular та Vue.js.

1) React.js – це відкрита JavaScript-бібліотека, розроблена компанією Facebook, що призначена для створення користувацьких інтерфейсів. Вона відрізняється компонентно-орієнтованим підходом, що дозволяє розділити інтерфейс на незалежні та багаторазово використовувані блоки. Ключовою особливістю React.js є використання віртуального DOM. При змінах у стані компонентів React оновлює віртуальний DOM, а потім визначає мінімальний набір операцій для синхронізації з реальним DOM, що значно підвищує продуктивність та швидкість рендерингу. Для керування станом використовуються механізми props та state, а також React Hooks у функціональних компонентах. React.js легко інтегрується з RESTful API або GraphQL-сервісами за допомогою асинхронних запитів, що дозволяє клієнтській частині взаємодіяти із серверною для отримання та обробки даних. Маршрутизація в односторінкових застосунках (SPA) реалізується за допомогою таких бібліотек, як React Router. React.js також добре поєднується з UI-фреймворками (наприклад, Material-UI), що пропонують готові компоненти для адаптивного дизайну. Велика екосистема та спільнота розробників забезпечують доступ до численних плагінів та інструментів, що сприяє подальшому розширенню функціоналу та підтримці проєкту.

2) Angular – фреймворк, розроблений компанією Google, призначений для створення клієнтських веб-додатків. Спочатку Angular був створений на

JavaScript, але пізніше переписаний з використанням TypeScript. Завдяки використанню TypeScript, Angular користується популярністю серед .NET розробників, оскільки TypeScript є статично типізованою мовою, що покращує підтримку коду та виявлення помилок. Angular має власну CLI, що працює на основі Node.js, яка спрощує компіляцію проєкту та додавання нових компонентів. Фреймворк пропонує комплексний підхід до розробки, включаючи двостороннє зв'язування даних, модульну архітектуру та вбудовані рішення для маршрутизації та керування станом.

3) Vue.js – прогресивний JavaScript-фреймворк, що фокусується на декларативному рендерингу та компонентній композиції. Він відомий своєю легкістю інтеграції, простотою вивчення та гнучкістю, що робить його привабливим для широкого спектру проєктів, від невеликих компонентів до повноцінних односторінкових застосунків. Vue.js забезпечує високу продуктивність завдяки оптимізованій роботі з DOM та реактивній системі.

Таблиця 2.2.

Порівняльний аналіз Front-end фреймворків

Критерій	React.js	Angular	Vue.js
Розробник	Facebook	Google	Незалежна спільнота
Тип	Бібліотека для UI	Повноцінний фреймворк	Прогресивний фреймворк
Основна мова	JavaScript, може використовувати TypeScript	TypeScript	JavaScript, може використовувати TypeScript
Архітектура	Компонентна, віртуальний DOM	Компонентна, MVC/MVVM-подібна, модульна	Компонентна, реактивна
Крива навчання	Помірна, гнучка, але вимагає вибору додаткових бібліотек	Крутіша, багато концепцій для вивчення	Більш полога, легкий старт
Продуктивність	Висока, завдяки віртуальному DOM	Висока, завдяки оптимізації та AOT-компіляції	Дуже висока, легкий та швидкий
Масштабованість	Висока, для великих та складних застосунків	Висока, для застосунків корпоративного рівня	Висока, від малих до великих проєктів

Екосистема	Велика та активна, багато сторонніх бібліотек та інструментів (Redux, MobX, React Router)	Комплексна, багато вбудованих рішень, інтеграція з Material Design, власна CLI	Зростаюча, гнучка, пропонує офіційні бібліотеки (Vuex, Vue Router)
Застосування	Односторінкові застосунки (SPA), мобільні додатки (React Native), складні UI	Великі корпоративні додатки, SPA	Малі та середні проєкти, SPA, прототипування, інтеграція в існуючі проєкти

На основі проведеного порівняльного аналізу, для реалізації клієнтської частини веб-застосунку з рекомендації ігор було обрано бібліотеку React.js. Цей вибір зумовлений її компонентно-орієнтованим підходом та ефективним використанням віртуального DOM, що забезпечує високу продуктивність інтерфейсу та швидке реагування на дії користувача. Гнучкість React.js у керуванні станом за допомогою React Hooks та легкість інтеграції з API сприяють розробці модульного та масштабованого коду. Широка екосистема, активна спільнота та сумісність з UI-фреймворками, такими як Material-UI, додатково обґрунтовують його вибір для створення функціонального та візуально привабливого інтерфейсу.

2.3. Засоби для розробки Back-end частини

Серверна частина веб-застосунку є основою системи, що відповідає за обробку запитів клієнтів, реалізацію бізнес-логіки, управління даними та взаємодію з базами даних. Вибір технологій безпосередньо впливає на продуктивність, безпеку, масштабованість та ефективність супроводу застосунку. Сучасний ринок пропонує розмаїття програмних засобів для реалізації серверної логіки, що вимагає ретельного аналізу для обґрунтованого вибору.

Розглянемо найдоступніші на ринку засобів для Back-end розробки:

1) Node.js.

Серверна платформа, що дозволяє виконувати JavaScript-код поза межами браузера. Вона забезпечує створення високопродуктивних веб-серверів із неблокуючою архітектурою вводу-виводу. Завдяки однопотоковому, подієво-орієнтованому підходу, Node.js здатна обробляти велику кількість одночасних з'єднань без необхідності створення додаткових потоків. Підтримує концепцію модульності шляхом використання системи пакетів npm, що полегшує підключення сторонніх бібліотек для реалізації додаткової функціональності.

2) Express.js

Виступає як мінімалістичний та гнучкий фреймворк для Node.js, який спрощує створення веб-серверів та налагоджує маршрутизацію HTTP-запитів. Він надає розробнику можливість організувати обробку запитів у вигляді конвеєра з проміжними посередниками (middleware), що відповідають за авторизацію, валідацію даних, логування та інші аспекти функціонування серверної частини. Express.js забезпечує зручні механізми для роботи з роутами (шляхами URL), що дозволяє чітко структурувати RESTful API додатку.

3) PHP

Одна з найпопулярніших мов програмування для веб-розробки, що використовується для створення динамічних веб-сторінок і додатків. Підтримується майже всіма хостинг-компаніями та є одним із лідерів серед мов програмування для написання динамічних веб-сторінок. PHP має велику спільноту розробників, простий синтаксис, що робить її легкою для вивчення та використання.

4) Python (з Django/Flask): Python є однією з найпопулярніших мов програмування, особливо для таких професій, як Data Scientist та Machine Learning Engineer, які часто розробляють рекомендаційні системи.

5) .NET – відкрита платформа для розробки різноманітних типів застосунків, включаючи веб, мобільні, настільні, геймінг та IoT застосунки, хмарні сервіси, а також машинне навчання.

6) Ruby on Rails – веб-фреймворком, написаним на мові Ruby, що працює за принципом "convention over configuration", спрощуючи швидку розробку веб-додатків.

Таблиця 2.3.

Порівняльний аналіз засобів для Back-end розробки

Технологія	Призначення	Переваги	Недоліки
Node.js	Серверна платформа для виконання JavaScript-коду поза браузером.	Високопродуктивні веб-сервери з неблокуючою архітектурою вводу-виводу, однопотоковий подієво-орієнтований підхід, обробка великої кількості одночасних з'єднань. Уніфікація мови програмування з фронтендом (Full-stack JavaScript).	Вимагає високої кваліфікації розробників для написання неблокуючого коду. Складність керування помилками в асинхронному середовищі. Може бути інтенсивною для CPU-bound операцій (обчислювальні завдання).
Express.js	Мінімалістичний та гнучкий фреймворк для Node.js, що спрощує створення веб-серверів та налагоджує маршрутизацію HTTP-запитів.	Організація обробки запитів у вигляді конвеєра з middleware (авторизація, валідація, логування). Гнучке визначення endpoint, чітка структура RESTful API. Сприяє розподіленій обробці логіки, полегшує підтримку та масштабування. Низький наклад на ресурси.	Мінімалістичний характер вимагає додаткових бібліотек для розширеного функціоналу (наприклад, ORM, валідація). Не має жорстких архітектурних обмежень, що може призвести до "спагеті-коду" без належного проектування.
PHP	Скриптова мова для створення динамічних	Широка популярність та підтримка більшістю хостинг-провайдерів. Простий синтаксис для	Історично мала репутацію менш структурованої та безпечної мови (хоча сучасні версії значно

	веб-сторінок та додатків.	вивчення та використання. Велика спільнота розробників та велика кількість документації/готових рішень. Потужні можливості для роботи з базами даних (MySQL). Платформонезалежність. Висока продуктивність, ефективна обробка великих навантажень.	покращилися). Відносна повільність у порівнянні з компільованими мовами.
Django	Потужний фреймворк "батарейки в комплекті" для швидкої розробки веб-додатків.	Потужний інструментарій, висока безпека та захист від вразливостей.	Може бути "оверхед" для невеликих проєктів. Менша гнучкість у виборі компонентів порівняно з мікрофреймворками.
Flask	Легкий та гнучкий мікрофреймворк.	Мінімалістичний підхід, швидкий старт для невеликих проєктів.	Вимагає більшого обсягу коду для реалізації складних функцій (порівняно з Django). Не має вбудованих рішень для багатьох завдань.
ASP.NET	Технологія створення веб-застосунків та веб-сервісів.	Швидке розгортання веб-додатків. Підтримка різних видів застосунків (MVC для HTML-сторінок, Web API для JSON). Гарна інтеграція з Microsoft-екосистемою (MsSQL, Azure).	Може бути складнішим для вивчення для початківців. Дещо вищий поріг входу порівняно з PHP/Python.
Ruby on Rails	Веб-фреймворк, що працює за принципом "convention	Спрощує швидку розробку веб-додатків.	Менша популярність порівняно з іншими фреймворками (Node.js, Python, PHP). Продуктивність може

	over configuration".		знижуватися на високих навантаженнях порівняно з Node.js або Go.

При розробці серверної частини обов'язково треба зробити вибір системи управління базами даних. Розглянемо найпопулярніші з них:

1) SQLite: Легковажна, файлова система управління базами даних, що дозволяє створити базу даних, яка працюватиме через єдиний файл. Node.js може підключатися до неї та отримувати дані. Вона є ідеальним рішенням для локальної розробки та невеликих проєктів, де не потрібен окремий сервер бази даних [12].

2) MySQL: Одна з найпоширеніших та надійних реляційних баз даних у світі. Вона використовується для зберігання та управління великими обсягами даних у веб-додатках та підприємствах. MySQL заснована на мові SQL та забезпечує простоту використання, стабільність та масштабованість. Має широке співтовариство розробників.

3) PostgreSQL: Відкрита та потужна об'єктно-реляційна база даних, що пропонує розширені можливості, такі як підтримка географічних об'єктів, складні запити та транзакції. Вона володіє високою надійністю та безпекою даних.

4) MongoDB: Це NoSQL-база даних, що використовує підхід до зберігання неструктурованих даних у вигляді документів JSON. MongoDB дозволяє швидше зберігати та отримувати дані, що робить її популярним вибором для проєктів з великим обсягом неструктурованих даних.

5) Microsoft SQL Server (MS SQL): Це система управління базами даних, розроблена корпорацією Microsoft. Вона виконує головну функцію зі збереження та надання даних у відповідь на запити інших застосунків. Використовує мову Transact-SQL, що є реалізацією стандарту SQL з

розширеннями. Застосовується як для невеликих, так і для великих баз даних масштабу підприємства.

На основі аналізу доступних програмних засобів для реалізації серверної частини веб-застосунку з рекомендації ігор було обрано наступний технологічний стек: **Node.js** з фреймворком **Express.js** та **SQLite** як система управління базами даних. Цей вибір зумовлений його перевагами в уніфікації мови програмування (JavaScript Full-stack), високою продуктивністю та масштабованістю для обробки запитів у реальному часі, а також простотою та зручністю SQLite для розробки та ефективної організації даних у рамках даного проєкту.

2.4 Висновок до другого розділу

У другому розділі проведено аналіз засобів реалізації веб-застосунку. Розглянуто ключові архітектури та патерни проектування, такі як MVC, MVVM, CQRS та Mediator, які дозволяють забезпечити масштабованість, модульність і спрощене тестування системи. Окрему увагу приділено інструментам для реалізації клієнтської частини, зокрема фреймворкам React.js, Angular та Vue.js, які проаналізовано за критеріями продуктивності, архітектурного підходу, екосистеми та зручності використання. На основі порівняльного аналізу для реалізації інтерфейсу користувача обрано React.js – легку, гнучку та ефективну бібліотеку, яка забезпечує високий рівень інтерактивності та продуктивності.

При розробці серверної частини обов'язково треба зробити вибір системи управління базами даних. У цьому розділі було проаналізовані найпопулярніші з них: SQLite, MySQL, PostgreSQL, MongoDB, NoSQL, Microsoft SQL Server (MS SQL). Були виявлені їх особливості і переваги в застосуванні.

Розглянуті найдоступніші на ринку засоби для Back-end розробки Node.js., Express.js, PHP, Python (з Django/Flask), .NET, Ruby on Rails. Були виявлені їх переваги і недоліки.

Серед проаналізованих для серверної частини низки технологій, найбільш доцільним рішенням обрано Node.js з Express.js, що дозволяє створити масштабований backend із використанням єдиної мови програмування. Також досліджено СУБД, серед яких обрано SQLite як легковажне, зручне для розробки та налаштування рішення.

Результати аналізу підтверджують доцільність вибору технологічного стеку, що включає React.js для клієнтської частини, Node.js з Express.js для серверної логіки та SQLite як систему управління базами даних. Така комбінація забезпечує узгодженість між фронтендом і бекендом, спрощує розробку та супровід застосунку, а також відповідає вимогам до продуктивності, масштабованості та зручності використання в межах обраного проєкту.

РОЗДІЛ 3. СТВОРЕННЯ ВЕБ-ДОДАТКУ ДЛЯ ВИБОРУ ВІДЕОГОР

3.1 Актуальність створення веб-додатку

У сучасних умовах стрімкого розвитку інформаційних технологій та постійного розширення ринку відеоігор питання ефективної організації процесу добору ігор для кінцевого користувача набуває особливої актуальності. Щорічно виходить безліч нових проєктів різного жанру, стилю й орієнтації, що супроводжується як зростанням вимог до персоналізації контенту, так і підвищенням рівня конкуренції серед розробників. У зв язку з цим виникає необхідність створення прикладних засобів, здатних не лише зберігати й опрацьовувати великі масиви даних про ігрові продукти, але й надавати користувачеві релевантні рекомендації з урахуванням його індивідуальних уподобань та ігрового досвіду. Саме в цих умовах розробка спеціалізованого програмного рішення для рекомендації відеоігор набуває не лише теоретичного, а й практичного значення, оскільки дозволяє підвищити задоволеність користувачів та оптимізувати їхній час на пошук нового контенту.

Основною метою є дослідження існуючих підходів до організації систем рекомендацій та розробка власного програмного комплексу для автоматизованого підбору відеоігор із використанням методів фільтрації. Завданням є процес проектування й реалізації інтегрованого рішення, що забезпечує збирання інформації про користувачів та ігрові продукти, побудову моделей співставлення користувач–гра, а також генерацію якісних рекомендацій на основі накопичених даних про вподобання та поведінкові характеристики користувачів.

Потрібно також створити нову модульну архітектуру, яка забезпечує гнучке розширення системи новими алгоритмами без значного перепроєктування основного коду.

Результатом виконаної роботи стане функціонально завершений програмний комплекс, що дозволяє формувати персоналізовані списки

рекомендованих ігор із урахуванням історії оцінок користувача, жанрових уподобань та його поведінки. Практична значущість полягає в тому, що впровадження розробленого рішення може бути корисним для онлайн-магазинів ігрового програмного забезпечення, платформ стрімінгу або спільнот геймерів, оскільки впливає на підвищення задоволеності користувачів та збільшення рівня монетизації за рахунок поліпшеної точності рекомендацій.

Таким чином розробка сприятиме розвитку методології створення високоефективних рекомендаційних систем для відеоігор, а розроблений програмний комплекс може бути використаний як базова платформа для подальшого розширення та вдосконалення в умовах динамічних потреб даного ринку.

У цьому контексті виникає необхідність створення спеціалізованого програмного рішення, здатного обробляти великі обсяги інформації про ігрові продукти й надавати користувачеві релевантні рекомендації з урахуванням його індивідуальних уподобань. Вибір саме веб-додатку як платформи для реалізації системи рекомендації пояснюється його доступністю з різних пристроїв, відсутністю необхідності встановлення, простотою оновлення та масштабування, а також можливістю інтеграції з різноманітними зовнішніми сервісами та API у реальному часі [8].

Вибір веб-платформи обумовлений сучасними тенденціями користувацького досвіду: користувачі звикли отримувати інформацію та сервіси безпосередньо через браузер, що забезпечує універсальний доступ з різних операційних систем і пристроїв. Зокрема, веб-додаток дає можливість реалізувати інтерактивний інтерфейс, гнучко масштабувати серверну частину, забезпечити централізоване зберігання даних і гарантувати швидке розгортання оновлень без участі кінцевого користувача.

Архітектура розроблюваного рішення складається з трьох основних компонентів: клієнтський інтерфейс, серверна частина та база даних. Клієнтський інтерфейс веб-додатку реалізується з використанням сучасних JavaScript-фреймворків, що забезпечує швидке реагування на дії користувача та

адаптивний дизайн для різних розмірів екранів. Серверна частина містить логіку обробки запитів, управління сесіями, взаємодію з базою даних та виконання алгоритмів фільтрації [10].

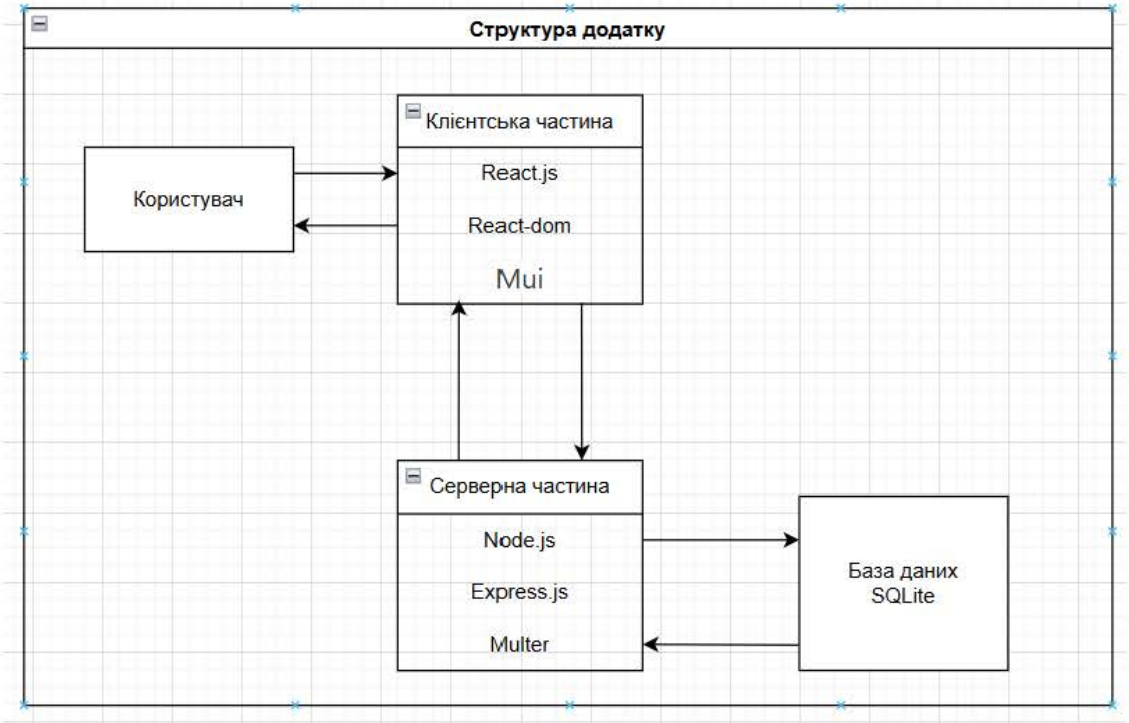


Рисунок 3.1 – Архітектура розроблюваного додатку

3.2 Інструменти розробки додатку

Серед основних бібліотек для відображення т візуального оформлення додатку використовується React.js

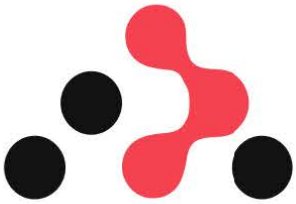
React.js є відкритою бібліотекою JavaScript, розробленою компанією Facebook для створення складних і динамічних користувацьких інтерфейсів. У контексті розробки вебдодатку для рекомендації ігор вона виконує роль основного інструменту для побудови клієнтської частини системи. Основними перевагами React.js є компонентно-орієнтований підхід, використання віртуального DOM та широка екосистема супутніх бібліотек і інструментів, що дозволяють оптимізувати процес розробки та покращити продуктивність інтерфейсу.

Компонентна архітектура React.js дозволяє розділити інтерфейс на незалежні, багаторазово використовувані блоки (компоненти), кожен із яких відповідає за певний фрагмент UI. У рамках веб-додатку для рекомендації ігор це спрощує реалізацію таких модулів, як список рекомендованих ігор, картка профілю користувача, фільтри пошуку та графічні елементи візуалізації (наприклад, рейтингові позначення на грі). Завдяки компонентам розробники можуть підтримувати чітку структуру коду, легко здійснювати повторне використання та модифікацію окремих елементів, що, в свою чергу, сприяє швидшому впровадженню змін та розширенню функціональності [15].

Однією з ключових особливостей React.js є використання віртуального DOM (Document Object Model) — JavaScript-об єкту, який репрезентує структуру реального DOM у пам'яті. Зміни в стані компонентів (state) або їхніх властивостях (props) призводять до оновлення віртуального DOM, після чого React визначає мінімальний набір операцій, необхідних для синхронізації з реальним DOM. Такий підхід значно зменшує кількість прямих маніпуляцій із деревом елементів у браузері, що позитивно впливає на продуктивність та швидкість рендерингу, особливо при роботі з великими обсягами інформації. У веб-додатку для рекомендації ігор це дозволяє миттєво оновлювати список рекомендацій або змінювати вигляд елементів інтерфейсу відповідно до дій користувача без затримок чи «мерехтінь» сторінки.

Для організації стану компонентів React.js використовує механізми props та state, а в сучасних версіях — також функціональні компоненти з React Hooks (наприклад, useState, useEffect, useContext). У проекті рекомендаційного додатку це забезпечує гнучку передачу даних між компонентами: наприклад, головний контейнер додатку може зберігати масив ігор, які рекомендовано користувачеві, та передавати цю інформацію дочірнім компонентам для відображення. Використання Hooks спрощує керування життєвим циклом компонентів, робить код більш лаконічним і зручним для розуміння, а також сприяє повторному використанню логіки (через створення власних хуків для, скажімо, роботи з API або обробки фільтрації даних).

Інтеграція React.js із RESTful API або GraphQL-сервісами реалізується за допомогою асинхронних запитів (наприклад, із використанням Fetch API чи бібліотеки axios). В даному випадку веб-додатку для рекомендації ігор клієнтська частина, побудована на React.js, відповідає за надсилання запитів до серверної частини (наприклад, для отримання списку ігор, збереження нових оцінок користувача або отримання оновлених рекомендацій).



React Router

Рисунок 3.2 – Маршрутизатор React Router

Важливим аспектом є можливість керування маршрутизацією клієнтської частини за допомогою додаткових бібліотек, наприклад React Router (рис. 3.2). У рекомендаційному додатку це дає змогу організувати кілька сторінок або «розділів»— реєстрацію/вхід користувача, перегляд профілю, сторінку з таблицею результатів рекомендацій, адміністративну панель для налаштування алгоритмів тощо, без необхідності повного перезавантаження сторінки. Клієнтська маршрутизація підтримує принцип односторінкового додатку (SPA), що робить взаємодію швидкою та плавною з точки зору UX.

Для забезпечення єдиного стилю та полегшення розробки інтерфейсу React.js добре поєднується з CSS-in-JS рішеннями (наприклад, styled-components або Emotion), а також із популярними UI-фреймворками, що пропонують готові компоненти (Material-UI, Ant Design тощо). У веб-додатку для рекомендації ігор використовується для швидкого створення адаптивних меню, кнопок, модальних вікон та віджетів, які коректно відображаються на різних розмірах екранів. При цьому можна зосередитися на логіці додатку, не витрачаючи багато часу на оформлення базових елементів дизайну.

Широка екосистема React.js і велика спільнота розробників забезпечують доступ до численних плагінів, бібліотек для управління станом (Redux, MobX), тестування (Jest, React Testing Library). Це відкриває можливості для подальшого розширення вебдодатку: наприклад, реалізації SSR (server-side rendering) для SEO-оптимізації, інтеграції з бібліотекою D3.js для побудови інтерактивних діаграм популярності ігор або застосування Redux Toolkit для централізованого управління станом у великому проєкті. Таке середовище дозволяє підтримувати високий рівень якості коду та масштабувати рішення з урахуванням майбутніх потреб.

Таким чином, React.js є логічним вибором для реалізації клієнтської частини веб-додатку, призначеного для рекомендації відеоігор. Він забезпечує модульність, масштабованість і високу продуктивність інтерфейсу, дозволяє легко інтегруватися з серверною частиною та сторонніми сервісами, а також створює сприятливі умови для подальшого розвитку та підтримки проекту в динамічному середовищі цифрового ринку.

Для реалізації серверної частини використовується Node.js+Express.js.

Node.js є серверною платформою, що дозволяє виконувати JavaScript-код поза межами браузера та забезпечує можливість створення високопродуктивних веб-серверів із неблокуючою архітектурою вводу-виводу. Завдяки однопоточного, подієво-орієнтованому підходу, Node.js здатна обробляти велику кількість одночасних з єднань без необхідності створення додаткових потоків. Це робить її ідеальною для реалізації серверної частини веб-додатків, зокрема тих, що вимагають інтенсивної взаємодії із зовнішніми джерелами даних та обробки запитів у реальному часі. Застосування Node.js у проєкті відеоігор дозволяє забезпечити швидку передачу даних між клієнтом та сервером, що особливо важливо при формуванні динамічних списків рекомендованих ігор і обробці запитів від великої кількості користувачів одночасно.

Express.js виступає як фреймворк для Node.js, який спрощує створення веб-серверів та налагоджує маршрутизацію HTTP-запитів. Він надає розробнику можливість організувати обробку запитів у вигляді конвеєра з проміжними

посередниками (middleware), що відповідають за авторизацію, валідацію даних, логування та інші аспекти функціонування серверної частини. Використання Express.js у складі серверного стеку проєкту дозволяє гнучко визначати endpoint для отримання інформації про користувача, передачі даних до модулів алгоритмів фільтрації та повернення результатів рекомендацій у форматі JSON. Така структура сприяє розподіленій обробці логіки програми та полегшує підтримку й масштабування бекенду [14].

Однією з ключових переваг Node.js є можливість розробки серверної логіки тією самою мовою програмування, яка використовується для клієнтської частини в React.js. Це дозволяє команді розробників застосовувати уніфіковані підходи до обробки даних, реалізації функціональних компонентів та підтримки загальних модулів. У контексті додатку це означає, що алгоритмічні модулі, що виконують обчислення чи рейтингової фільтрації, можуть бути написані на JavaScript та безпосередньо інтегровані в інфраструктуру сервера Node.js. Таким чином, забезпечується злагоджена взаємодія між компонентами системи, зменшується кількість технологічних бар єрів при обміні даними та зростає швидкість розробки.

Express.js надає зручні механізми для роботи з роутами (шляхами URL), що дозволяє чітко структурувати RESTful API додатку. У додатку це виражається в розподілі маршрутів за функціональними групами: наприклад, окремі шляхи для реєстрації та аутентифікації користувачів, отримання метаданих ігор, передачі оцінок користувачів та запитів до модуля рекомендацій. Завдяки підтримці параметризованих маршрутів і можливості обробки запитів різних типів (GET, POST, PUT, DELETE), Express.js забезпечує чітку та прозору взаємодію frontend-частини з backend. Крім того, middleware-функції дозволяють централізовано обробляти tokens аутентифікації та обмежувати доступ до окремих частин програми, що сприяє підвищенню безпеки системи.

Ще одним важливим аспектом є робота з базою даних. За допомогою модулів Node.js (наприклад, з використанням бібліотек для взаємодії з реляційними СУБД чи NoSQL-сховищами) можна ефективно організувати

доступ до інформаційної моделі, яка містить дані про профілі користувачів, історію оцінок, метадані ігор та статистичні показники взаємодій. Express.js інтегрується з цими модулями, забезпечуючи асинхронність запитів до бази даних, що зменшує час відповіді серверу та сприяє масштабованості. У випадку рекомендаційної системи це дає змогу швидко формувати матрицю «користувач–гра», проводити запити до алгоритмічних компонентів та повернути результат клієнту без значних затримок [13].

Node.js підтримує концепцію модульності шляхом використання системи пакетів npm, що полегшує підключення сторонніх бібліотек для реалізації додаткової функціональності. У рамках проєкту це включає обробку автентифікації через JWT, зберігання сесій користувачів, збирання та аналіз даних. Використання Express.js у цьому середовищі створює єдину точку входу для під єднання необхідних middleware, що дозволяє централізовано керувати конфігурацією сервера, налаштовувати обробку помилок та визначати політику CORS, необхідну для взаємодії фронтенду та бекенду через різні домени.

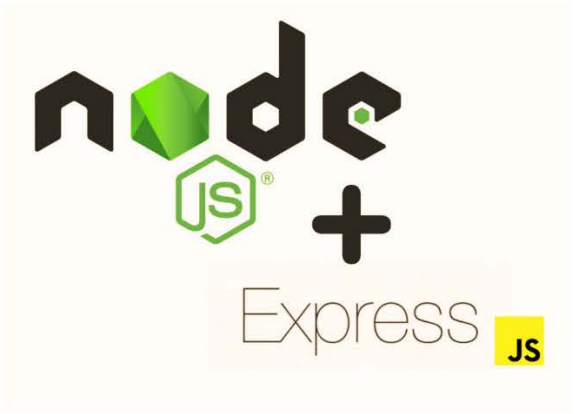


Рисунок 3.3 – Node.js+Express.js.

Щодо питань масштабування, Node.js у комбінації з Express.js (рис. 3.3) дозволяє реалізовувати горизонтальне масштабування за допомогою кластеризації процесів або розгортання мікросервісної архітектури. У випадку високого навантаження на додаток, серверні екземпляри можуть бути додатково розподілені між кількома машинами або контейнерами, що забезпечить

безперервну доступність і відмовостійкість системи. Використання Express.js у вигляді веб-фреймворка сприяє низькому накладу на ресурси, що полегшує запуск кількох частин додатку та зменшує час, необхідний для їхньої ініціалізації.

В якості середовища розробки використовується Visual Studio Code. Visual Studio Code (VS Code) є сучасним, крос-платформеним редактором коду, розробленим компанією Microsoft, який поєднує в собі функціональність повноцінного середовища розробки з легкістю та швидкістю звичайного текстового редактора. Його архітектура побудована на основі Electron, що забезпечує незалежність від операційної системи: VS Code однаково добре працює на Windows, macOS та Linux. В контексті розробки вебдодатку для рекомендації відеоігор цей інструмент відіграє ключову роль, оскільки дозволяє інтегрувати всі необхідні компоненти (редагування коду, налагодження, керування версіями, запуск скриптів) в єдине середовище, що сприяє підвищенню продуктивності та зручності роботи розробників.

Однією з основних переваг VS Code є система розширень (extensions), яка дозволяє адаптувати редактор під специфіку конкретного проєкту. Для розробки клієнтської частини на React.js та серверної частини на Node.js+Express.js рекомендовано використовувати такі розширення, як ESLint, Prettier, стилізатори коду для JavaScript/TypeScript, а також плагіни для роботи з фреймворками (наприклад, React PropTypes чи React Snippets). Це забезпечує автоматичну перевірку синтаксису, приведення коду до єдиного стилю та швидке створення шаблонів компонентів. У академічному контексті можна відзначити, що подібна система розширень сприяє зниженню кількості помилок на ранніх етапах розробки та полегшує підтримку кодової бази, адже стандарти форматування та стилю узгоджуються між усіма членами команди.

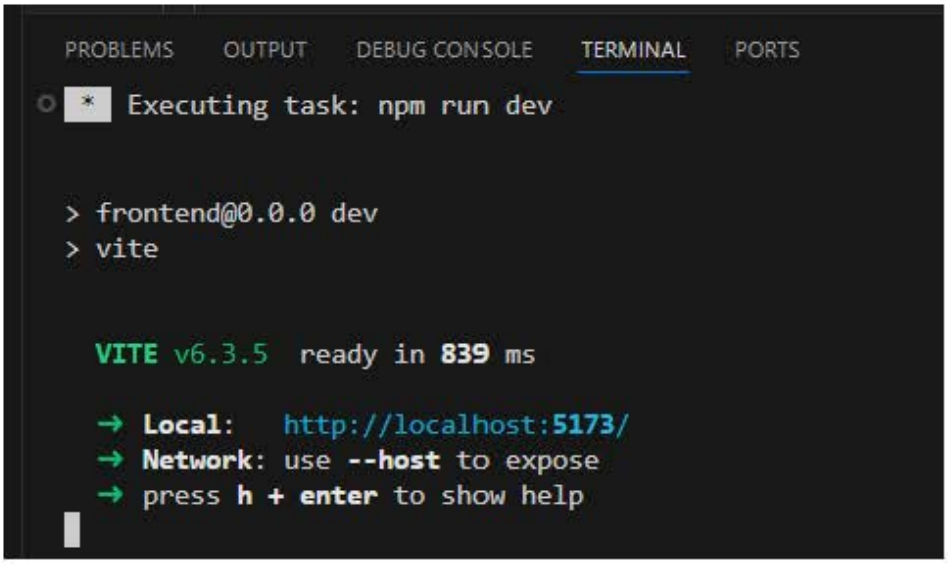


Рисунок 3.4 – Вбудований термінал VS Code

Важливим є інтегрований термінал, який дає змогу виконувати командний рядок без необхідності перемикатися на окремий емулятор. Для проєкту на Node.js це означає миттєве встановлення нових пакетів (через npm або yarn), запуск серверного процесу, виконання скриптів для побудови фронтенду на React.

3.3 Розробка веб-додатку

Для початку розробки спочатку потрібно визначити та створити робочу логіку backend проекту. Потрібно визначити структуру бази даних для зберігання та виведення інформації через frontend. В якості бази даних використовується SQLite, що дозволяє створити базу-даних, що працюватиме через єдиний файл, до якого Node.js підключатися та отримувати дані.

Структура бази даних складається з п яти сутностей, які взаємодіють між собою через визначені зв язки. Створюються дані структури за допомогою скрипту db.js:

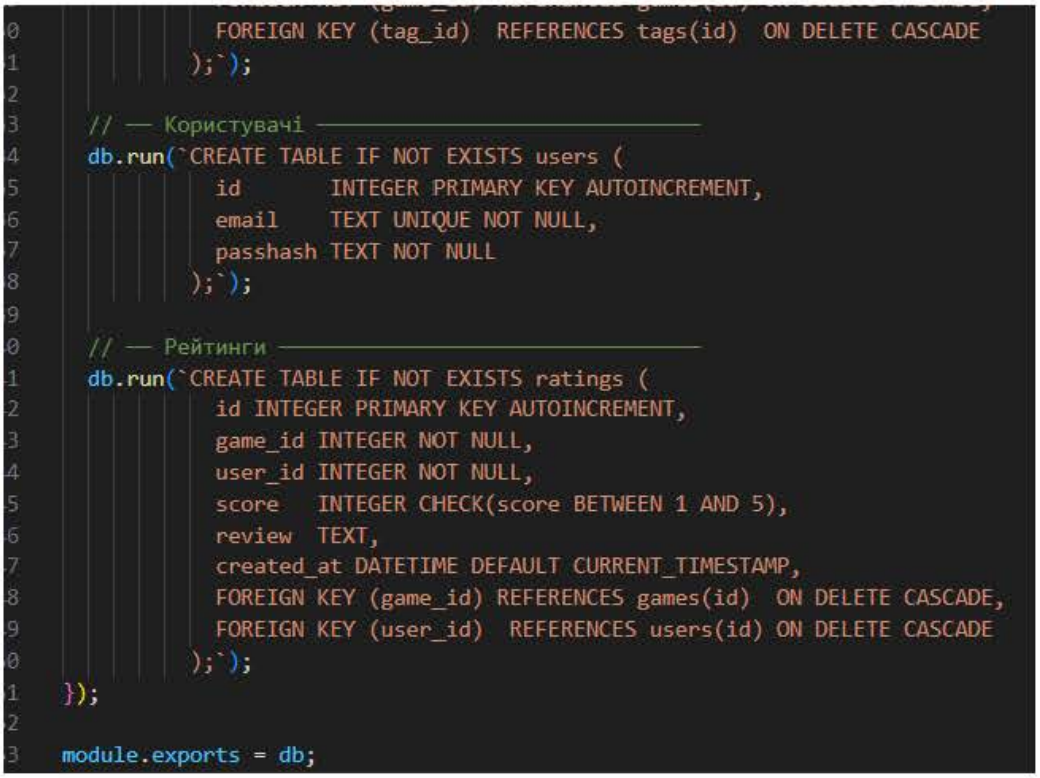


Рисунок 3.5 – Код db.js

Перша сутність — таблиця games, що зберігає інформацію про ігрові продукти. Поле id у цій таблиці є первинним ключем із автоматичною інкрементацією, що гарантує унікальність кожного запису. Поле title зберігає назву гри у вигляді тексту та є обов язковим для заповнення, тоді як поле coverUrl зберігає URL зображення обкладинки гри. Поле description містить текстовий опис гри з початковим значенням порожнього рядка.

Друга сутність — таблиця tags, призначена для зберігання тегів або жанрів, які можуть характеризувати ігри. Аналогічно до таблиці games, у ній є первинний ключ id із автоматичною інкрементацією. Поле name зберігає унікальне текстове значення назви тегу, що не може повторюватися в межах таблиці. Завдяки цьому забезпечується єдність кожного жанру або категорії ігор, що полегшує організацію пошуку та фільтрації.

Третя сутність відображає зв язок many-to-many між іграми та тегами і називається game_tags. У цій таблиці поля game_id і tag_id спільно утворюють первинний ключ, що забезпечує унікальність поєднання певної гри з певним

тегом. Поле game_id посилається на поле id у таблиці games, а поле tag_id — на поле id у таблиці tags. Наявність зовнішніх ключів із опцією ON DELETE CASCADE гарантує, що при видаленні гри чи тегу з відповідної головної таблиці автоматично буде видалено всі пов язані записи у таблиці game_tags.

Четверта сутність — таблиця users, що відповідає за зберігання облікових даних користувачів системи. У цій таблиці також є поле id, яке є первинним ключем. Поле email зберігає унікальну електронну адресу користувача та не може бути порожнім, що дозволяє ідентифікувати кожного користувача. Поле passhash містить результат хешування паролю і також є обов язковим для заповнення.

П'ята сутність — таблиця ratings, призначена для зберігання оцінок і відгуків користувачів щодо певних ігор. У цій таблиці поле id є первинним ключем із автоматичною інкрементацією. Поле game_id посилається зовнішнім ключем на id у таблиці games, а user_id — на id у таблиці users. Таким чином, кожен запис у таблиці ratings пов язує конкретного користувача з конкретною грою. Поле score містить числову оцінку гри в діапазоні від 1 до 5, що контролюється відповідним обмеженням CHECK. Опції ON DELETE CASCADE для зовнішніх ключів game_id та user_id забезпечують видалення усіх оцінок, пов язаних із грою або користувачем, у разі видалення відповідного запису зі сторони games чи users.

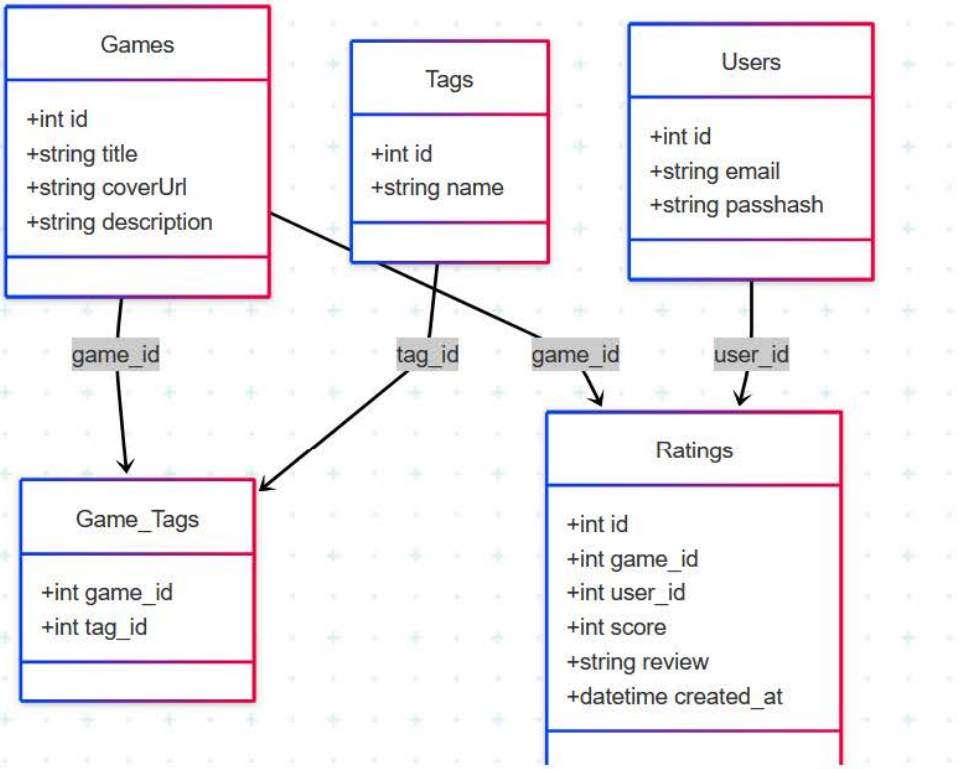


Рисунок 3.6 – Структура бази даних

За роботу та обробку запитів CURD-операцій використовується головний скрипт backend – index.js. В ньому зібрані всі роутери, що використовуються для відправки та отримання даних:

```
const authRoutes  = require('./routes/auth');
const gamesRoutes = require('./routes/games');
const tagsRoutes  = require('./routes/tags');

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

// підключаємо роутери
app.use('/api/auth',  authRoutes);   // -> /login, /register
app.use('/api/games', gamesRoutes);  // -> / , /:id/rate, …
app.use('/api/tags',  tagsRoutes);

// healthcheck
app.get('/api/health', (_, res) => res.json({ ok: true }));
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
app.listen(3001, () => console.log('✅  API on http://localhost:3001'));
```

Рисунок 3.7 – Код index.js

На почтаку відбувається підключення необхідних Node.js-модулів: бібліотеки Express для створення серверу, модуля CORS для налаштування

політик доступу з інших доменів та модуля Path для роботи з файловими шляхами. Виокремлено також три набори маршрутів (роутерів): authRoutes відповідає за маршрути, пов язані з аутентифікацією (реєстрація та вхід користувачів), gamesRoutes обробляє запити, пов язані з іграми (отримання списку ігор, додавання рейтингів тощо), а tagsRoutes призначений для управління тегами або жанрами.

Далі створюється екземпляр застосунку Express шляхом виклику express(). Безпосередньо після цього до нього підключаються два стандартні middleware-функції. Cors() дозволяє налаштувати політику крос-доменного доступу, що є необхідним для забезпечення безперешкодної взаємодії клієнтської частини (наприклад, React-додатку) з API на іншому хості чи порті. Express.json() відповідає за автоматичний розбір JSON-тіл вхідних HTTP-запитів, що дозволяє зручно працювати з даними, які надсилаються в форматі JSON (наприклад, дані про нову гру чи рейтинг).

Справжня «робота» серверу починається з підключення маршрутів за допомогою app.use(). Шлях /api/auth призначений для обробки запитів до маршрутів аутентифікації — наприклад, POST /api/auth/register або POST /api/auth/login. Маршрути, що стоять за шляхом /api/games , обробляють запити, пов язані з іграми: отримання загального списку ігор (GET /api/games), отримання даних конкретної гри (GET /api/games/:id) або встановлення/оновлення рейтингу (POST /api/games/:id/rate). Аналогічно, /api/tags призначений для роботи з тегами: додавання нових жанрів, отримання наявних тегів тощо.

Після налаштування основних маршрутів визначено просту стркутуру для перевірки стану (healthcheck) GET /api/health, який у відповідь повертає JSON-об єкт { ok: true }. Такий endpoint корисний для моніторингу доступності сервера чи налаштування автоматизованих перевірок (наприклад, у контексті CI/CD), щоб упевнитися, що API запущено та відповідає на запити.

Останніми в додатку налаштовано статичну видачу файлів із директорії uploads. Виклик app.use(/uploads , express.static(path.join(__dirname, uploads)))

означає, що всі файли в папці uploads (наприклад, зображення обкладинок або інші завантажені ресурси) будуть доступні за URL-адресою, яка починається з /uploads. Це дозволяє клієнтській частині напряму отримувати необхідні медіафайли, використовуючи прості HTTP-запити типу GET /uploads/назва_файлу.jpg.

У завершальній частині файлу виконується запуск серверу на порту 3001. Виклик app.listen(3001, () => console.log(API on http://localhost:3001)); повідомляє Node.js почати використовувати вказаний порт, а також виводить у консоль повідомлення про успішний старт із зазначенням адреси. Якщо сервер запустився без помилок, у терміналі з явиться рядок «API on http://localhost:3001», що свідчитиме про готовність API до обробки вхідних запитів.

Далі потрібно розглянути саму реалізацію роботи роутів. Функціонал розподілений між 3 скриптами auth.js, games.js, tags.js (рис. 3.8):

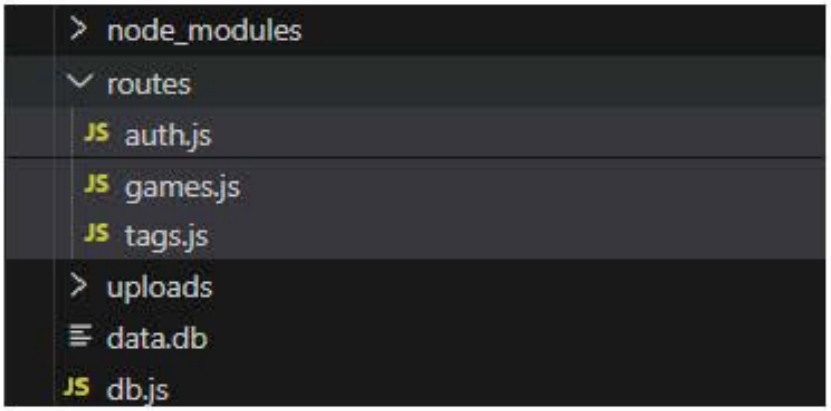


Рисунок 3.8 – Розподілення CURD операції на 3 скрипти

Файл auth.js (рис. 3.9) відповідає за реалізацію логіки реєстрації та входу користувачів. Спочатку підключаються необхідні модулі. У маршруті POST /api/auth/register відбувається обробка запиту на реєстрацію нового користувача. Спочатку з тіла запиту (req.body) витягуються поля email та password. Якщо одне з цих полів відсутнє, сервер повертає відповідь зі статусом 400 Bad Request і повідомленням про помилку. У разі наявності обох значень здійснюється

асинхронне хешування пароля з використанням функції bcrypt.hash(password, 10), де число 10 означає кількість раундів соління.

```
    return res.status(400).json({ error: 'Email and password re

try {
  const passhash = await bcrypt.hash(password, 10);
  db.run(
    'INSERT INTO users(email, passhash) VALUES(?,?)',
    [email, passhash],
    function (err) {
      if (err) {
        return res.status(400).json({ error: 'Email already t
      }
      // this.lastID — це id новозареєстрованого користувача
      const userId = this.lastID;
      const role = 'user';  // за замовчуванням у БД уже 'use
      const token = jwt.sign(
        { id: userId, email, role },
        JWT_SECRET,
        { expiresIn: '7d' }
      );
      res.json({ token });
    }
  );
} catch (e) {
  res.status(500).json({ error: 'Server error' });
}
```

Рисунок 3.9 – Скрипт auth.js

Після успішного отримання хешованого пароля (passhash) виконуються SQL-запит INSERT INTO users(email, passhash) VALUES(?,?), який додає новий запис до таблиці users. Використання плейсхолдерів ? та масиву [email, passhash] гарантує безпечне підставлення значень і захищає від SQL-ін єкцій. Якщо під час вставки виникає помилка (наприклад, email уже існує), у callback-функції це призводить до повернення статусу 400 із повідомленням "Email already taken". В іншому випадку в полі this.lastID зберігається ідентифікатор щойно створеного користувача.

Далі формується JWT-токен, та передаються дані користувача: його id, email та роль (за замовчуванням user). Використовується JWT_SECRET і встановлюється час життя токена expiresIn: 7d , що означає дійсний термін у сім днів. Після успішного підписання сервер повертає у відповіді JSON-об єкт з полем token, яке містить сформований JWT.

У маршруті POST /api/auth/login відбувається аутентифікація вже зареєстрованих користувачів. З тіла запиту також витягуються email та password.

За допомогою db.get(SELECT * FROM users WHERE email = ? , [email], ...) здійснюється пошук користувача за вказаною електронною адресою. Якщо під час виконання запиту сталася помилка або запис не знайдено (!user), сервер поверне 400 Bad Request із повідомленням "Invalid credentials".

Таким чином, скрипт auth.js реалізує два ключові ендпоінти для керування аутентифікацією: реєстрація (/register) та вхід (/login). Застосування bcrypt гарантує безпеку зберігання паролів у вигляді хешу, а використання JWT забезпечує безпечний механізм передачі інформації про автентифікованого користувача між клієнтом та сервером у подальших запитах.

Аналогічним працює інші скрипти. Різниця лише в кількості наборів даних та в яку таблицю вони зберігаються.

Для frontend-компоненту відображення починається App.jsx – що відповідає за створення початкової сторінки додатку (рис. 3.10):

```
  onLogout={() => { logout(); loadGames(); }}
  open={sidebarOpen}                    // відкриття чи зачинення
  onClose={() => setSidebarOpen(false)} // виклик, коли потрібно сховати
/>

    {/* 2. Основний контент */}
    <Box component="main" sx={{ flexGrow: 1, p: 3 }}>
      <Toolbar sx={{ display: 'flex', alignItems: 'center' }}>
      <IconButton
        edge="start"
        color="inherit"
        onClick={() => setSidebarOpen(prev => !prev)}
        sx={{ mr: 2 }}
        >
        <MenuIcon />
      </IconButton>
      <Typography variant="h6"></Typography>
      </Toolbar>
      <AuthDialog
        open={authOpen}
        onClose={() => setAuthOpen(false)}
        onAuth={() => { loadTags(); loadGames(); }}
      />

      <Container maxWidth="md" sx={{ py: 2 }}>
        <Typography variant="h4" align="center" gutterBottom>
          Ігрові рекомендації
        </Typography>
```

Рисунок 3.10 – App.jsx

Також даний скрипт реалізовує в собі роботу об єкту Sidebar – бокової панелі меню додатку.

Головний файл main.jsx формує єдину точку входу для React-додатку, стилі Material UI, реалізує підтримку контексту теми через власний компонент-провайдер, а також створює інфраструктуру для маршрутизації (рис. 3.11).

Рисунок 3.11 – Код скрипту main.jsx

У початковому блоці коду здійснюється підключення необхідних модулів та бібліотек, що забезпечують функціональність та оформлення React-додатку. Із пакету @mui/material імпортуються компоненти CssBaseline, ThemeProvider та функція createTheme, які використовуються для створення та застосування єдиної теми оформлення Material UI. Додатково підключається ThemeWrapper із власного файлу ThemeContext, призначений для налаштування та передачі стану теми у межах усього додатку. Основний компонент інтерфейсу імпортується як App. Маршрутизацію бере на себе компонент RoutesRoot, імпортований із папки routes. Для організації навігації у межах SPA (Single Page Application) використовується компонент BrowserRouter із бібліотеки react-router-dom.

У наступному фрагменті коду створюється об єкт теми за допомогою функції createTheme. У його конфігурації визначено палітру, в якій задаються параметр mode: dark для темного режиму інтерфейсу, а також кольори primary.main, background.default та background.paper. Значення кольорів було підібрано таким чином, щоб забезпечити високий контраст між елементами фону (#111 та #1A1A1A) і основним акцентним кольором (#00ADB5). Додатково налаштовано округлення кутів (borderRadius: 12) для більш м якого сприйняття інтерфейсу. У розділі components перевизначено поведінку компонента MuiCard , додавши плавний перехід для властивостей transform та box-shadow , що створює ефект підняття карточок при наведені курсора та надає додатку більшої динамічності й сучасного вигляду.

Після створення об єкта теми ініціалізується корінний елемент React-додатку з використанням createRoot , що відповідає новому API React 18 для рендерингу. Вказівник document.getElementById(root) дозволяє прив язати React-дерево до елементу з ідентифікатором root у HTML-файлі, що забезпечує точку входу для всіх компонентів інтерфейсу. Виклик root.render() містить JSX-структуру, у якій найзовнішнім елементом є ThemeWrapper .

Всередині ThemeWrapper розміщено компонент BrowserRouter , що відповідає за обробку URL-маршрутів та навігаційних подій у межах SPA. Використання BrowserRouter дозволяє компонентам нижчого рівня, зокрема RoutesRoot, перемикатися між окремими сторінками без повного перезавантаження сторінки. Саме RoutesRoot виступає єдиним елементом у цьому дереві тут — він містить усі оголошені маршрути й обробляє їхню візуалізацію. Таким чином, коли користувач переходить за різними URL, RoutesRoot передає відповідні компоненти для відображення залежно від шляху.

У результаті вказана структура забезпечує централізоване налаштування теми оформлення та маршрутизацію. Весь додаток отримує єдиний контекст теми через ThemeWrapper та доступ до Material UI-тематики через компоненти CssBaseline і ThemeProvider . Використання цих інструментів гарантує, що всі

елементи інтерфейсу дотримуватимуться заданих кольорових схем, відступів і анімаційних ефектів.

Для відображення карток ігор разом з їхніми тегами використовується Flip.jsx.

Компонент FlipCard реалізує інтерактивну картку гри з ефектом перевертання, використовуючи функціональні можливості React і стилі Material UI. На початку імпортуються необхідні залежності React— useState для керування станом перевертання та useEffect і useRef для відстеження подій миші. Додатково підключаються базові компоненти з бібліотеки @mui/material, такі як Box, Card, CardMedia, CardContent, Typography, Button та Chip, які відповідають за структурне та візуальне оформлення обох сторін картки (рис. 3.12).

```
// src/FlipCard.jsx
import { useState } from 'react';
import { useRef, useEffect } from 'react';
import {
  Box,
  Card,
  CardMedia,
  CardContent,
  Typography,
  Button,
  Chip,
} from '@mui/material';

export default function FlipCard({ game, onClick, sx }) {
  const [flipped, setFlipped] = useState(false);
  const ref = useRef(null);
   useEffect(() => {
    if (!flipped) return;

    function handleMouseMove(e) {
      const rect = ref.current.getBoundingClientRect();
      const { clientX: x, clientY: y } = e;
      // якщо за межами елемента — повернути
      if (x < rect.left || x > rect.right || y < rect.top || y > rect.bottom) {
        setFlipped(false);
        window.removeEventListener('mousemove', handleMouseMove);
      }
```

Рисунок 3.12 – Початок Flip.jsx

Всередині компонента створюється локальний стан flipped, ініціалізований значенням false. Цей стан контролює, чи відобразиться передня сторона карточки, чи її зворотна поверхня. За допомогою useRef створюється посилання на DOM-елемент контейнера (ref), що дає змогу отримати координати його

прямокутної області. UseEffect запускається кожного разу, коли стан flipped стає true, і додає глобальний обробник події mousemove. Якщо курсор миші виходить за межі елементу картки, відповідно до розрахунків за допомогою getBoundingClientRect(), викликається функція setFlipped(false), яка повертає картку у початкове положення (рис. 3.13).

```
export default function FlipCard({ game, onClick, sx }) {
 const [flipped, setFlipped] = useState(false);
 const ref = useRef(null);
  useEffect(() => {
   if (!flipped) return;

   function handleMouseMove(e) {
     const rect = ref.current.getBoundingClientRect();
     const { clientX: x, clientY: y } = e;
     // якщо за межами елемента — повернути
     if (x < rect.left || x > rect.right || y < rect.top || y > rect.bottom) {
       setFlipped(false);
       window.removeEventListener('mousemove', handleMouseMove);
     }
   }

   window.addEventListener('mousemove', handleMouseMove);
   return () => window.removeEventListener('mousemove', handleMouseMove);
 }, [flipped]);
```

Рисунок 3.13 – Компонент FlipCard

Кореневим елементом компонента є Box, який має стилі, що задають перспективу (perspective: 1000px), а також фіксовані розміри (width: 260px, height: 300px) і курсор у вигляді вказівника. Подія onMouseEnter на цьому контейнері встановлює стан flipped у true, що спричиняє початок анімації перевертання. Властивість sx компоненту Box приймає можливість передати додаткові стилі з батьківського компонента.

Всередині головного Box (рис. 3.13) знаходиться ще один вкладений Box, на якому встановлені стилі для анімації перевертання: властивості transformStyle: preserve-3d та transition: transform 0.6s ease забезпечують плавний 3D-перехід. Залежно від значення flipped, CSS-властивість transform задає обертання по осі Y на 180 градусів (rotateY(180deg)), та повернення у вихідне положення (rotateY(0deg)).

```
    window.addEventListener('mousemove', handleMouseMove);
    return () => window.removeEventListener('mousemove', handleMouseMove);
  }, [flipped]);
  return (
    <Box
      ref={ref}
      onMouseEnter={() => setFlipped(true)}
      sx={{
        perspective: '1000px',
        width: 260,
        height: 300,
        cursor: 'pointer',
        ...sx,
      }}
    >
      <Box
        sx={{
          position: 'relative',
          width: '100%',
          height: '100%',
          transformStyle: 'preserve-3d',
          transition: 'transform 0.6s ease',
          transform: flipped ? 'rotateY(180deg)' : 'rotateY(0deg)',
        }}
      >
        {/* Передня сторона */}
        <Card
          sx={{
            position: 'absolute',
            width: '100%',
            height: '100%',
            backfaceVisibility: 'hidden',
```

Рисунок 3.13 – Контейнер Box

Всередині цього розміщено зображення гри через CardMedia, яке автоматично підтягує URL обкладинки game.coverUrl або відображає шаблонне зображення, якщо обкладинка відсутня. Під зображенням знаходиться CardContent, у якому за допомогою Typography виводиться назва гри, а за допомогою Box та Chip — перелік тегів гри. Якщо у моделі гри є середній бал (game.avgScore), зверху праворуч відображається невеликий прямокутник з округленим числом, стилізований через sx, що нагадує бейдж із рейтингом.

Позаду картки сформована іншим Card, який також має абсолютне позиціонування та сховану лицьову поверхню (backfaceVisibility: hidden. У цьому Card засобами Typography виводиться опис гри або стандартний текст «Опис відсутній», якщо опис не передано. Нижня частина цієї сторінки містить кнопку, створену через Button, яка у випадку натискання викликає функцію onClick.

Взаємодія між обома сторонами побудована таким чином, що коли курсор заходить на область картки, стан flipped стає істинним, і через стилі transform елемент плавно відвертається, демонструючи задню поверхню. Як тільки курсор виходить за межі прямокутника, обробник mousemove виводиться з контексту, стан повертається у false, і знову розгортається лицьовою стороною (рис. 3.14).

```jsx
{/* Задня сторона */}
<Card
  sx={{
    position: 'absolute',
    width: '100%',
    height: '100%',
    backfaceVisibility: 'hidden',
    transform: 'rotateY(180deg)',
    p: 2,
    display: 'flex',
    flexDirection: 'column',
    justifyContent: 'space-between',
  }}
>
  <Typography
    variant="body2"
    sx={{ textAlign: 'justify', flexGrow: 1, mb: 2, overflow: 'hidden' }}
  >
    {game.description || 'Опис відсутній.'}
  </Typography>

  <Box sx={{ textAlign: 'center', mt: 'auto' }}>
    <Button
      size="small"
      variant="contained"
      onClick={(e) => {
        e.stopPropagation();
        onClick();
      }}
    >
```

Рисунок 3.14 – Jsx-розмітка сторони картки позаду

3.4 Тестування додатку

Для тестування додатку потрібно завантажити проект використовуючи термінал Visual Studio Code, визначивши команду npm start, що завантажує проект та виводить за наступною адресою: http://localhost:5173/. Далі відкривається наступна сторінка додатку:

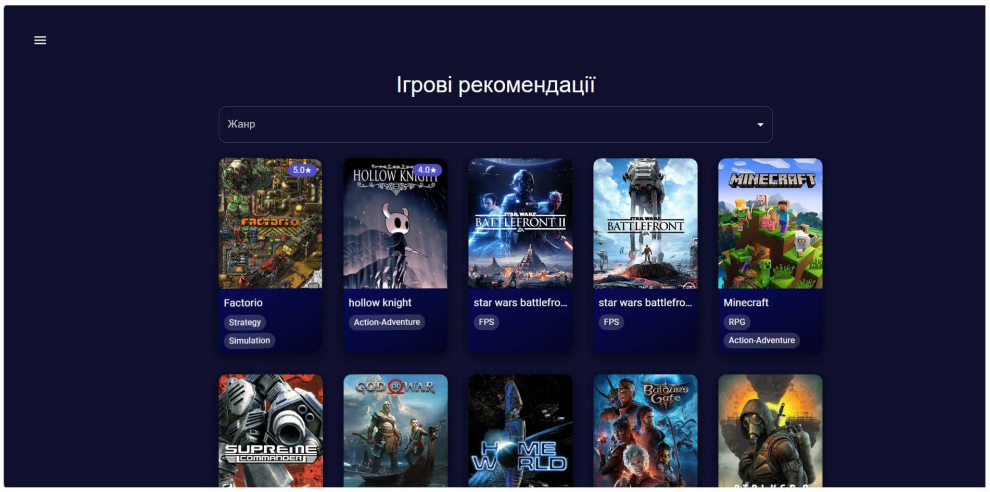


Рисунок 3.15 – Відкриття головної сторінки додатку

Після завантаження сторінки ми можемо спостерігати кнопку бокового меню додатку, по центру основний контент, та налаштування фільтрів для різних жанрів. Для обрання потрібного жанру потрібно натиснути на відповідне поле та відкриється випадаючий список з жанрами (рис. 3.16):

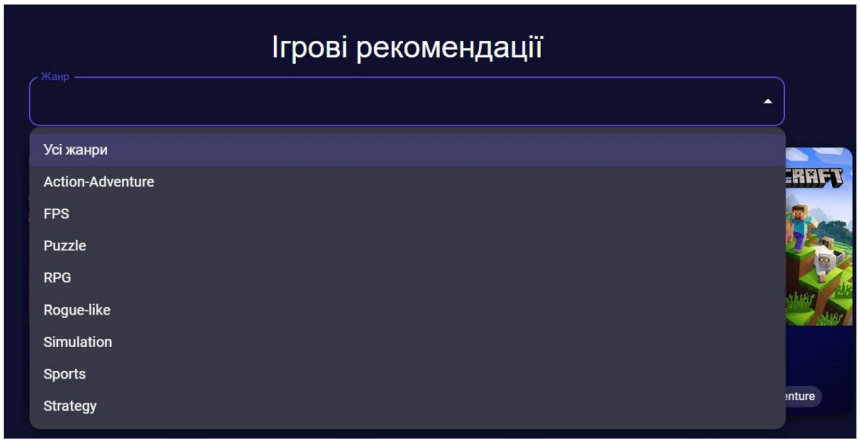


Рисунок 3.16 – Випадаючий список з жанрами

В даному випадку обрано «Усі жанри», спробуємо вибрати на вибір жанр «FPS», та розглянути чи коректно відображаються ігри (рис. 3.17):

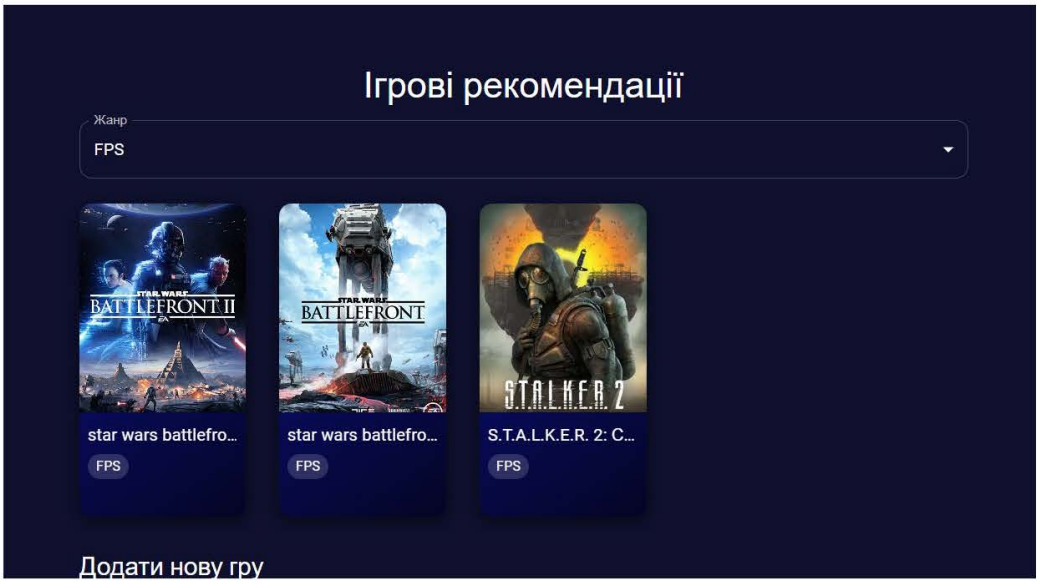


Рисунок 3.17 – Виведення ігор за жанром «FPS»

Крім цього також можна розглянути короткий опис цих ігор просто навівши курсор на одну з карток, відповідно додаток зреагує та переверне картку та відобразить текст (рис. 3.18):

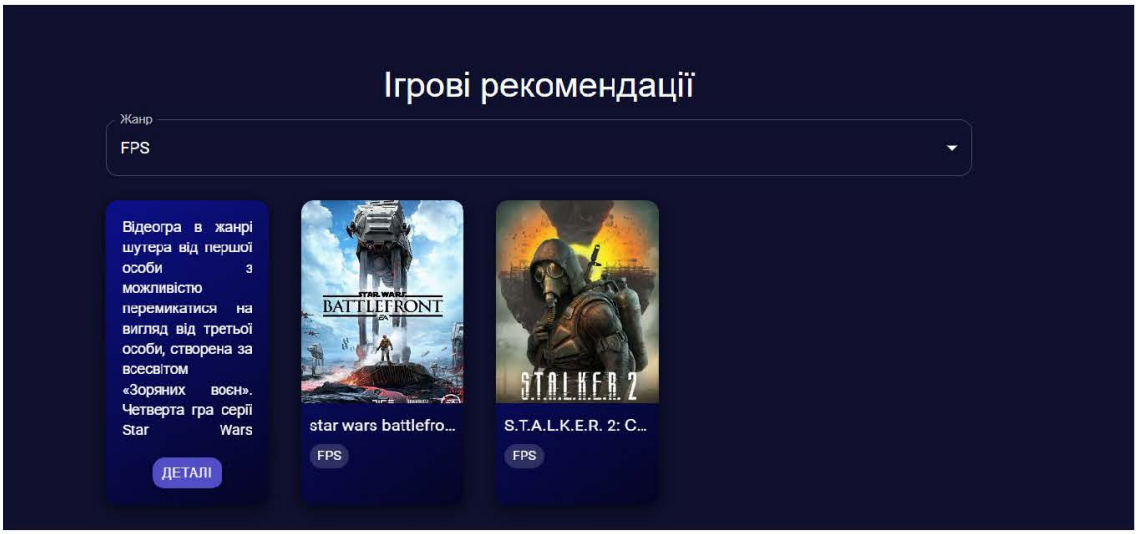


Рисунок 3.18 – Ефект Flip для картки

Далі розглянемо бокову панель меню. Для цього потрібно натиснути на наступну кнопку (рис. 3.19):

Рисунок 3.19 – Кнопка для відкриття бокової панелі меню

Далі потрібно натиснути на неї і з’явиться панель з якою користувач може взаємодіяти натиснувши на панелі меню (рис. 3.20):

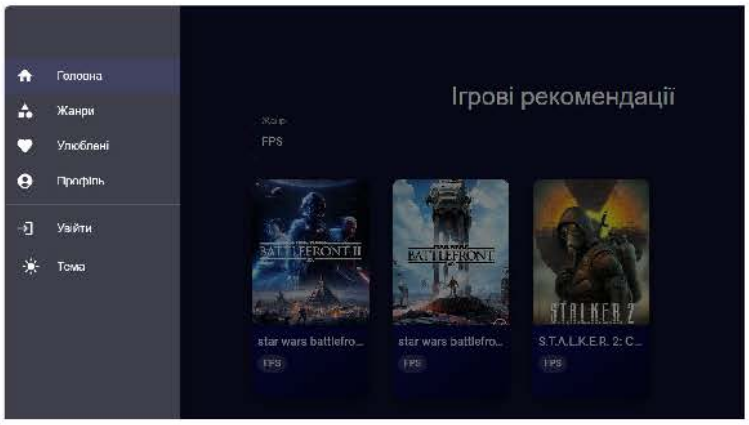


Рисунок 3.20 – Бокова панель меню

В меню можна змінити тему сторінки натиснувши на відповідну кнопку. Після цих дій на сторінці зміниться тема на більш світлу.

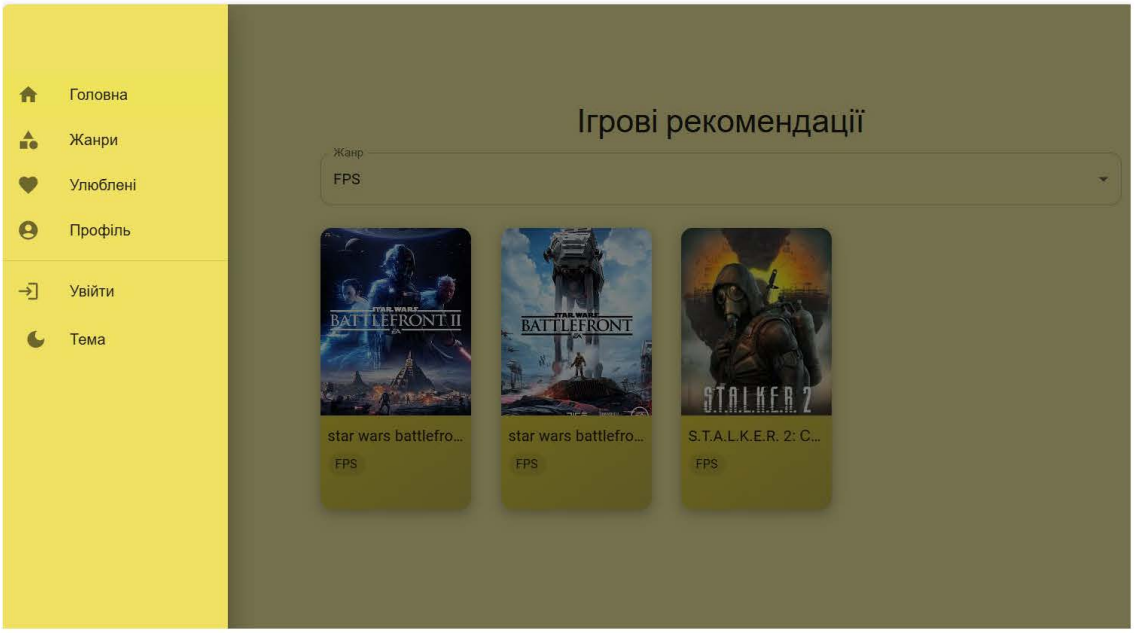


Рисунок 3.21 – Зміна теми на інший колір

Далі можна розглянути конкретні відомості про кожну гру: його опис фото та оцінку. Для цього потрібно перейти до конкретної гри натиснувши на відповідну кнопку «Деталі» (рис. 3.22):

Рисунок 3.22 – Розгляд відомостей конкретної гри

Повернувшись на головну сторінку користувач може зареєструватися за допомогою наступного модального вікна (рис. 3.23):

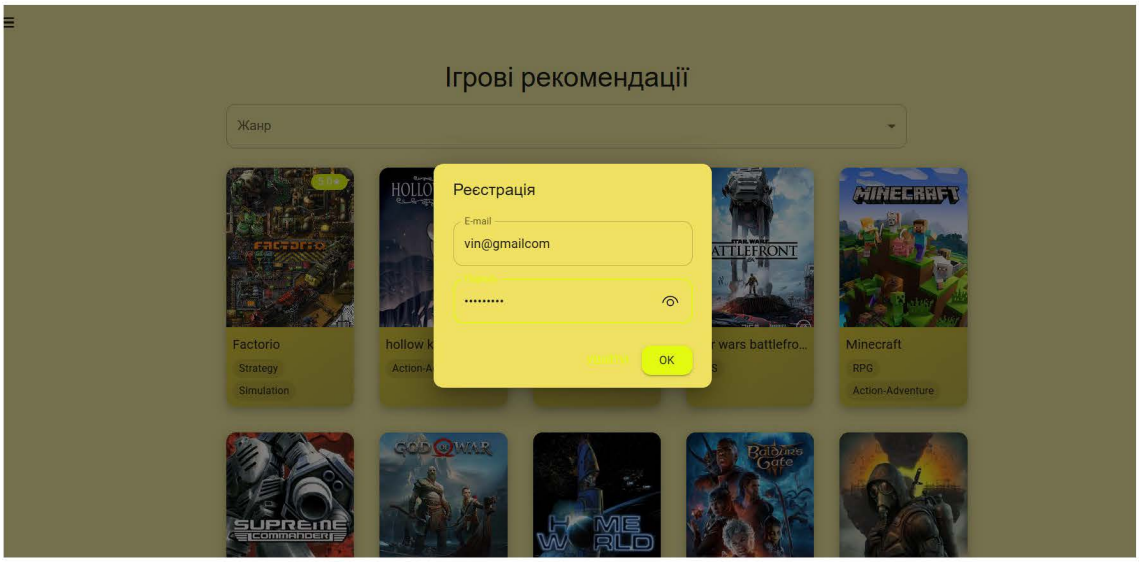


Рисунок 3.23 – Реєстрація користувача

Після того як користувач ввів відповідні дані потрібно натиснути «ОК». Відповідно користувач може спостерігати назву акаунту в боковій панелі (рис. 3.24):

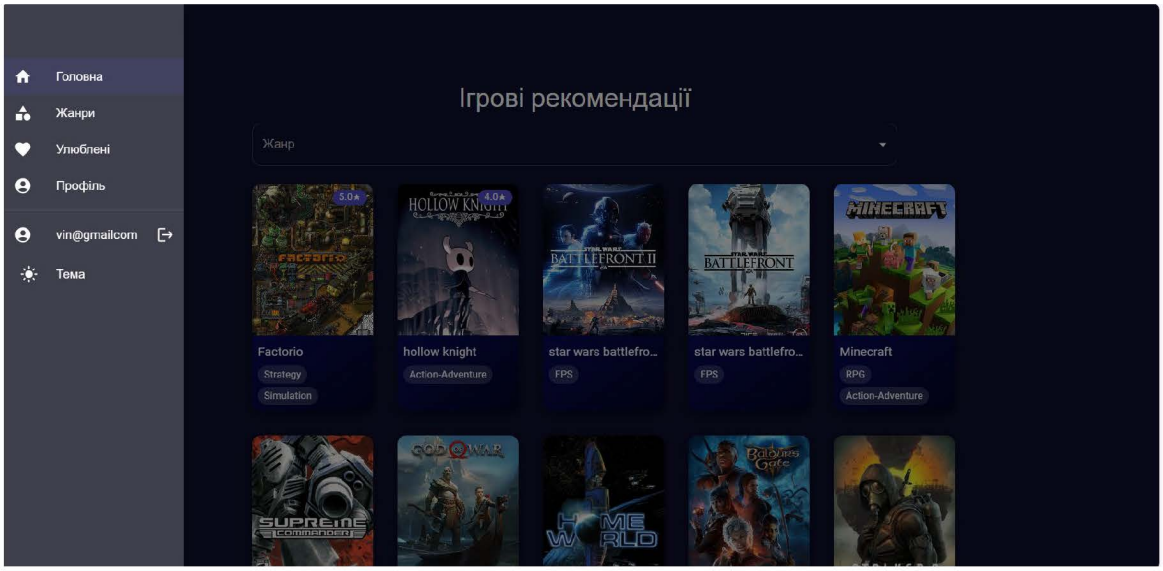


Рисунок 3.24 – Відображення ім’я акаунту на боковій панелі

Далі користувач після того як зайшов/зареєструвався, він може оцінити гру за власним бажанням вибравши 1-5 зірок:

Рисунок 3.25 – Оцінювання гри від 1 до 5

Далі повернувшись на головну сторінку користувач може спостерігати, що порядок ігор змінився, відповідно до його побажань (рис. 3.26):

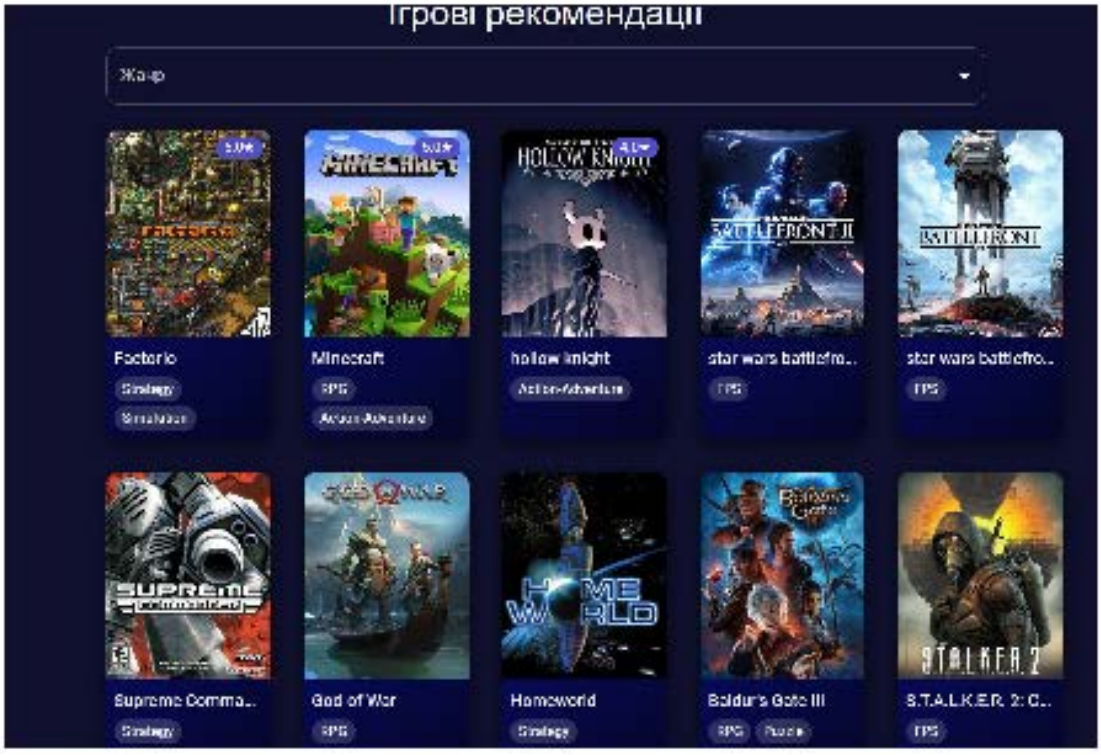


Рисунок 3.26 – Сортування ігор за рейтингом

За бажанням користувач може вийти з акаунту, відкривши бокове меню, та натиснувши на зображення виходу.

В даному випадку користувач вже не зможе залишати відгуки до кожної з представлених ігор, лише їх розглядати.

3.5 Висновок до третього розділу

Питання ефективної організації процесу добору ігор для кінцевого користувача набуває особливої актуальності у сучасних умовах стрімкого розвитку та постійного розширення ринку відеоігор. Саме в цих умовах розробка спеціалізованого програмного рішення для рекомендації відеоігор набуває не лише теоретичного, а й практичного значення, оскільки дозволяє підвищити задоволеність користувачів та оптимізувати їхній час на пошук нового контенту.

В результаті виконання роботи створено функціональний веб-додаток для рекомендації відеоігор, який забезпечує комплексний підхід до персоналізації підбору ігрового контенту. Веб-додаток успішно обробляє асинхронні запити й адаптується до зростання кількості користувачів завдяки масштабованій архітектурі.

Третій розділ кваліфікаційної роботи присвячено розробці застосунку.

Архітектура розроблюваного рішення складається з трьох основних компонентів: клієнтський інтерфейс, серверна частина та база даних. Клієнтський інтерфейс веб-додатку реалізується з використанням сучасних JavaScript-фреймворків, що забезпечує швидке реагування на дії користувача та адаптивний дизайн для різних розмірів екранів. Серверна частина містить логіку обробки запитів, управління сесіями, взаємодію з базою даних та виконання алгоритмів фільтрації.

Описані основні особливості застосування розглянутих в другому розділі роботи технологій створення веб застосунків при створені веб-застосунку вибору ігор.

Таким чином, створений програмний продукт відповідає поставленим цілям і завданням, забезпечуючи інтуїтивне взаємодію з інтерфейсом та надійну backend-логіку, що може бути розгорнута в веб-середовищі й використовуватися онлайн-магазинами ігор чи геймерськими спільнотами для підвищення задоволеності користувачів і покращення монетизації за рахунок персоналізованих рекомендацій.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці веб-застосунку для вибору відеоігор, що є актуальним завданням в умовах стрімкого зростання ігрової індустрії та великого обсягу доступної інформації. Метою роботи було створення зручного інструменту, який дозволяє користувачам орієнтуватися в різноманітті ігор та приймати більш обґрунтовані рішення щодо їх вибору, використовуючи елементи персоналізації.

Здійснено аналіз предметної області, досліджено сучасні онлайн-платформи, сервіси для пошуку ігор та існуючі підходи до організації рекомендацій. Виявлено основні проблеми користувачів при виборі ігор, серед яких — перевантаження інформацією, неінтуїтивні інтерфейси та відсутність інструментів для індивідуального підбору. Також розглянуто методології проєктування веб-застосунків та сформульовано загальні вимоги до майбутнього програмного продукту. Крім того розглянуті спеціалізовані веб-сайти, які зосереджені на інших аспектах вибору ігор, зокрема на порівнянні системних вимог та комплектуючих ПК.

У другому розділі обґрунтовано вибір програмних засобів для реалізації системи. Проведено порівняльний аналіз сучасних фреймворків та технологій для клієнтської і серверної частин.

На основі проведеного порівняльного аналізу, для реалізації клієнтської частини веб-застосунку з рекомендації ігор було обрано бібліотеку React.js. Цей вибір зумовлений її компонентно-орієнтованим підходом та ефективним використанням віртуального DOM, що забезпечує високу продуктивність інтерфейсу та швидке реагування на дії користувача. Гнучкість React.js у керуванні станом за допомогою React Hooks та легкість інтеграції з API сприяють розробці модульного та масштабованого коду. Широка екосистема, активна спільнота та сумісність з UI-фреймворками, такими як Material-UI, додатково обґрунтовують його вибір для створення функціонального та візуально привабливого інтерфейсу.

На основі аналізу доступних програмних засобів для реалізації серверної частини веб-застосунку з рекомендації ігор було обрано наступний технологічний стек: **Node.js** з фреймворком **Express.js** та **SQLite** як система управління базами даних. Цей вибір зумовлений його перевагами в уніфікації мови програмування (JavaScript Full-stack), високою продуктивністю та масштабованістю для обробки запитів у реальному часі, а також простотою та зручністю SQLite для розробки та ефективної організації даних у рамках даного проєкту.

У третьому розділі описано етапи розробки веб-застосунку. Створено базову архітектуру системи, реалізовано клієнтську та серверну частини, організовано обмін даними, а також реалізовано окремі елементи системи рекомендацій, зокрема механізми збору даних про користувача, структуру оцінювання ігор та інтерфейс для відображення персоналізованих результатів. Проведене тестування підтвердило функціональність та стабільність роботи розробленого застосунку.

Розроблений веб-застосунок демонструє можливість часткової автоматизації процесу вибору відеоігор та є ефективним інструментом з елементами рекомендаційної системи. Отримані результати можуть слугувати основою для подальшого розвитку проєкту, зокрема впровадження повноцінних алгоритмів фільтрації, покращення аналітики та розширення функціоналу відповідно до потреб користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chalyi, S. Temporal modeling of user preferences in recommender system / S. Chalyi, V. Leshchynskyi // CEUR Workshop Proceedings. – 2020. – Vol. 2711. – P. 518–528.

2. Chalyi, S. Method of constructing explanations for recommender systems based on the temporal dynamics of user preferences / S. Chalyi, V. Leshchynskyi // EUREKA: Physics and Engineering. – 2020. – № 3. – P. 43–50.

3. Байдак, В.Є. Комбінований підхід до побудови рекомендаційної системи для онлайн-системи продажу електронних ігор / В.Є. Байдак, О.О. Мазурова // Інноваційні технології : матеріали наук.-техн. конф. студентів, аспірантів, докторантів та молодих учених, Київ, 25-26 листоп. 2020 р. – Київ, 2020. – С. 106–110.

4. Scholz, S.S. Contemporary scientometric analyses using a novel web application: the science performance evaluation (SciPE) approach / S.S. Scholz, M. Dillmann, A. Flohr, C. Backes et al. // Clinical Research in Cardiology. – 2020. – Т. 109. – № 7. – С. 810–818.

5. Fredj, O.B. An OWASP top ten driven survey on web application protection methods / O.B. Fredj, O. Cheikhrouhou et al. // International Conference on Risks and Security of Internet and Systems. – Cham : Springer, 2020. – С. 235–252.

6. Isinkaye, F.O. Recommendation systems: Principles, methods and evaluation / F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh // Egyptian Informatics Journal. – 2018. – Vol. 16, Is. 3. – P. 261–273.

7. S.Springer. React: The Comprehensive Guide to Mastering React.js with Hands-on Examples, Expert Tips, and Everything You Need to Build Dynamic, Scalable User Interfaces (Rheinwerk Computing). 2023. 676 p.

8. Ricci, F. Recommender Systems Handbook / F. Ricci, L. Rokach, B. Shapira. – Springer, 2011. – P. 1–35.

9. Martin, R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R.C. Martin. – Boston : Prentice Hall, 2018. – P. 58.

10. DOU. Розуміння Клієнт-Серверної Архітектури на прикладах. URL: https://dou.ua/forums/topic/44636/

11. Training-qatestlab. Клієнт-серверна архітектура. URL:https://training.qatestlab.com/blog/technical-articles/client-server-architecture/

12. SQLite3. Documentation URL: https://www.sqlite.org/docs.html

13. Node.js. Documentation URL: https://nodejs.org/docs/latest/api/

14. Express.Guide URL: https://expressjs.com/en/guide/routing.html

15. React URL: https://react.dev/learn