МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Відокремлений структурний підрозділ
«Тернопільський фаховий коледж
Тернопільського національного технічного університету імені Івана Пулюя»

Відділення телекомунікацій та електронних систем

Циклова комісія комп'ютерних наук

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра

на тему: Розробка вебсервісу дистрибуції комп'ютерних ігор «KeyGames»

Виконав: студент IV курсу, групи КН-421
спеціальності: 122 Комп'ютерні науки
Антон ШЕВЯКОВ

Керівник Руслан СЛОБОДЯН

Рецензент
(ім'я та прізвище)

м. Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерних наук
Освітньо-професійний ступінь «фаховий молодший бакалавр»
Спеціальність 122 Комп'ютерні науки
Галузь знань 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерних наук
_____ Галина МАРЦІЯШ
« 02 » березня 2026 року

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

Шевякову Антону Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка вебсервісу дистрибуції комп'ютерних ігор «KeyGames»

керівник роботи Слободян Руслан Олесійович,
(прізвище, ім'я, по батькові)
затверджені наказом вищого навчального закладу № 4/9-132 від 27.02.2026 р.
2. Строк подання студентом роботи: 19.06.2026 р.
3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мови програмування: Java, JavaScript, HTML; фреймворк Spring, СКБД PostgreSQL, стандарти ДСТУ ISO/IEC/IEEE 29148:2025, ДСТУ ISO/IEC/IEEE 29119-1:2017, ДСТУ 3008:2015, ДСТУ 8302:2015.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1 Загальний розділ

1.1 Аналітичний огляд існуючих рішень

1.2 Технічне завдання

1.2.1 Найменування та область застосування

1.2.2 Призначення розробки

1.2.3 Вимоги до програмного забезпечення

1.2.4 Вимоги до програмної документації

1.2.5 Техніко-економічні показники

1.2.6 Стадії та етапи розробки

1.2.7 Порядок тестування та прийому

2 Розробка технічного та робочого проєкту

2.1 Розробка структури вебсервісу

2.2 Створення та верстка сторінок вебсервісу

2.3 Розробка структури бази даних вебсервісу

2.4 Програмування вебсервісу

2.4.1 Написання клієнтської частини

2.4.2 Написання серверної частини

2.5 Тестування вебсервісу

3 Спеціальний розділ

3.1 Інструкція з розгортання вебсервісу

3.2 Інструкція з обслуговування та наповнення вебсервісу

3.3 Інструкція з популяризації та підтримки вебсервісу

4 Економічний розділ

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення розробки вебсервісу

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

4.3 Розрахунок витрат на електроенергію

4.4 Розрахунок суми амортизаційних відрахувань

4.5 Обчислення накладних витрат

4.6 Складання кошторису витрат та визначення собівартості вебсервісу

4.7 Розрахунок ціни вебсервісу

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

5 Охорона праці, техніка безпеки та екологічні вимоги

5.1 Аналіз небезпечних та шкідливих виробничих факторів

5.2 Вимоги до приміщення та організації робочого місця

6 Висновки

Додаткові вказівки: виконання кваліфікаційної роботи із розробкою програмного продукту – вебсервісу «KeyGames».

5. Перелік графічного матеріалу:

1. Схема структурна вебсервісу
2. UML-діаграма класів серверної частини вебсервісу
3. ER-діаграма бази даних
4. Таблиця техніко-економічних показників

6. Консультанти розділів роботи

Розділ	**Ім'я, прізвище та посада консультанта**	**Підпис, дата**	
		завдання видав	**завдання прийняв**
Економічний розділ	Любов КАЛУШКА		
Охорона праці, техніка безпеки та екологічні вимоги	Генадій ГОРЯЧЕК		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	20.03.2026	
2	Збір і узагальнення інформації	01.05.2026	
3	Написання першого розділу	15.05.2026	
4	Розробка технічного та робочого проєкту	29.05.2026	
5	Написання спеціального розділу	05.06.2026	
6	Розрахунок економічної частини	08.06.2026	
7	Написання розділу охорони праці	09.06.2026	
8	Виконання графічної частини	10.06.2026	
9	Оформлення пояснювальної записки	11.06.2026	
10	Погодження нормоконтролю	12.06.2026	
11	Попередній захист кваліфікаційної роботи	.06.2026	
12	Захист кваліфікаційної роботи	.06.2026	

7. Дата видачі завдання: 06 березня 2026 р.

Студент _____ Антон ШЕВЯКОВ
(підпис)

Керівник роботи _____ Руслан СЛОБОДЯН
(підпис)

ЗМІСТ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Розробка вебзастосунку дистрибуції комп'ютерних ігор «KeyGames».

Мета роботи — створення програмного комплексу для продажу та дистрибуції ліцензійних ключів комп'ютерних ігор через вебінтерфейс з розмежуванням прав користувача та адміністратора.

У пояснювальній записці наведено шість основних розділів. У загальній частині виконано аналітичний огляд існуючих платформ цифрової дистрибуції ігор та сформульовано технічне завдання на розробку. У другому розділі описано архітектуру застосунку, варіанти використання, систему класів, структуру бази даних PostgreSQL, реалізацію клієнтської та серверної частин вебзастосунку, а також результати тестування. Спеціальний розділ містить інструкції з інсталяції, тестування та експлуатації. Економічний розділ містить розрахунок собівартості та ефективності розробки. Питання охорони праці розглянуто у п'ятому розділі.

Середовище розробки**:** операційна система Microsoft Windows 11; середовище IntelliJ IDEA; фреймворк Spring Boot[1] 3.3.1; СУБД PostgreSQL; клієнтська частина — HTML5[4], CSS3, JavaScript. Мова програмування серверної частини**:** Java 17.

Результат**:** функціонуючий вебзастосунок із каталогом ігор, кошиком, оформленням замовлень, sandbox-оплатою, видачею ключів, списком бажань, відгуками та адміністративною панеллю.

Обсяг пояснювальної записки — 110 сторінок, 27 рисунків, 26 таблиць, 18 джерел. Графічна частина — 4 аркуші формату А1.

Ключові слова**:** вебзастосунок, дистрибуція ігор, Spring Boot[1], PostgreSQL, електронна комерція, цифровий ключ.

ABSTRACT

Topic of the qualification thesis: Development of the "KeyGames" web application for computer game distribution.

The purpose of the thesis is to create a software system for the sale and distribution of licensed computer game keys via a web interface, featuring a separation of user and administrator privileges.

The explanatory note consists of six main chapters. The general section provides an analytical review of existing digital game distribution platforms and formulates the technical specifications for development. The second chapter describes the application architecture, use cases, class system, PostgreSQL database structure, client-side and server-side implementations, as well as testing results. A dedicated section contains deployment, testing, and operation instructions. The economic section includes calculations of production cost and development efficiency. Occupational health and safety issues are addressed in the fifth chapter.

Development environment: Microsoft Windows 10/11 operating system; IntelliJ IDEA IDE; Spring Boot[1] 3.3.1 framework; PostgreSQL DBMS; client-side – HTML5[4], CSS3, JavaScript[5]. Server-side programming language: Java 17.

Result: A fully functional web application featuring a game catalog, shopping cart, checkout, sandbox payment integration, key issuance, wishlist, user reviews, and an administrative panel.

The explanatory note comprises 122 pages, 27 figures, 26 tables, and 18 references. The graphical part consists of 4 sheets of A1 format.

Keywords: web application, game distribution, Spring Boot[1], PostgreSQL, e-commerce, digital key.

ВСТУП

Ринок відеоігор у світі та в Україні демонструє стійке зростання. За даними аналітичних агенцій, основна частка продажів ПЗ припадає на цифрові магазини — Steam, Epic Games Store, GOG та інші платформи. Користувачі очікують зручного каталогу, миттєвої доставки ключів після оплати, персоналізованих списків бажань і можливості залишати відгуки. Для навчальних і демонстраційних цілей актуальною є розробка власного вебзастосунку, що імітує ключові процеси комерційної дистрибуції ігор.

Актуальність теми зумовлена потребою фахівців у вмінні проєктувати та реалізовувати повноцінні інформаційні системи електронної комерції[12]: серверною логікою, реляційною базою даних, автентифікацією, адмініструванням контенту та інтеграцією платіжного сценарію. Розробка вебзастосунку «KeyGames» дозволяє закріпити теоретичні знання з комп’ютерних наук, баз даних, вебтехнологій і безпеки на практичному прикладі.

Мета кваліфікаційної роботи — розробка вебзастосунку дистрибуції комп'ютерних ігор «KeyGames», що забезпечує перегляд каталогу, оформлення замовлень, тестову оплату, видачу ліцензійних ключів, ведення списку бажань і відгуків, а також адміністрування асортименту.

Для досягнення мети необхідно вирішити такі задачі**:**

- провести аналіз існуючих рішень цифрової дистрибуції ігор;
- сформувати технічне завдання на розробку програмного забезпечення;
- спроєктувати архітектуру системи, базу даних та інтерфейс користувача;
- реалізувати серверну частину на Spring Boot[1] і клієнтську — на HTML[4]/CSS/JavaScript[5];
- забезпечити розмежування ролей «користувач» та «адміністратор»;
- провести тестування функціоналу та оформити програмну документацію;
- виконати економічні розрахунки та розглянути питання охорони праці.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Цифрова дистрибуція комп'ютерних ігор — один із найбільших сегментів ринку програмних продуктів. Нижче наведено огляд платформ, які визначають сучасні вимоги до функціоналу системи «KeyGames».

Steam — найбільший світовий магазин PC-ігор. Забезпечує каталог із фільтрами, кошик, бібліотеку придбаних ігор, відгуки користувачів, бажаний список, регіональні ціни. Оплата здійснюється через вбудований гаманець і зовнішні платіжні системи. Ключова особливість — миттєва активація в клієнті Steam без окремого ключа для більшості товарів.

Першим об’єктом дослідження обрано магазин Steam показаний на рисунку1.1 .Платформа демонструє розвинений каталог із боковими фільтрами, рейтингами та блоком рекомендацій.

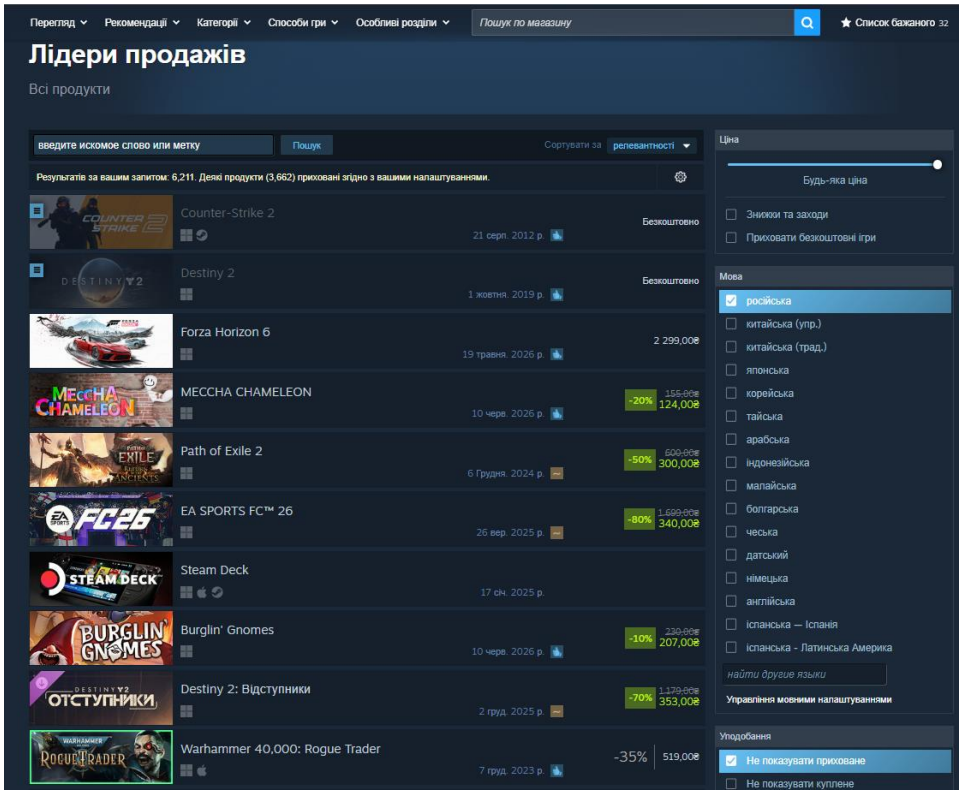


Рисунок 1.1 – Інтерфейс каталогу інтернет-магазину Steam

На основі аналізу Steam виявлено такі переваги:

– глибока система фільтрації за жанром, ціною, тегами та платформою;
– інтегровані відгуки користувачів і агрегований рейтинг;
– бібліотека придбаних ігор і список бажаного в одному акаунті.

До недоліків можна віднести перевантаженість інтерфейсу промо-блоками та залежність від встановленого клієнта Steam для частини операцій.

Магазин Epic Games пропонує щотижневі безкоштовні ігри, ексклюзиви та інтеграцію з обліковим записом Epic. Інтерфейс мінімалістичний; акцент на промоціях і знижках. Для розробників доступний SDK і API каталогу.

Epic Games Store пропонує щотижневі безкоштовні ігри, ексклюзиви та інтеграцію з обліковим записом Epic. Інтерфейс мінімалістичний; акцент на промоціях і знижках.

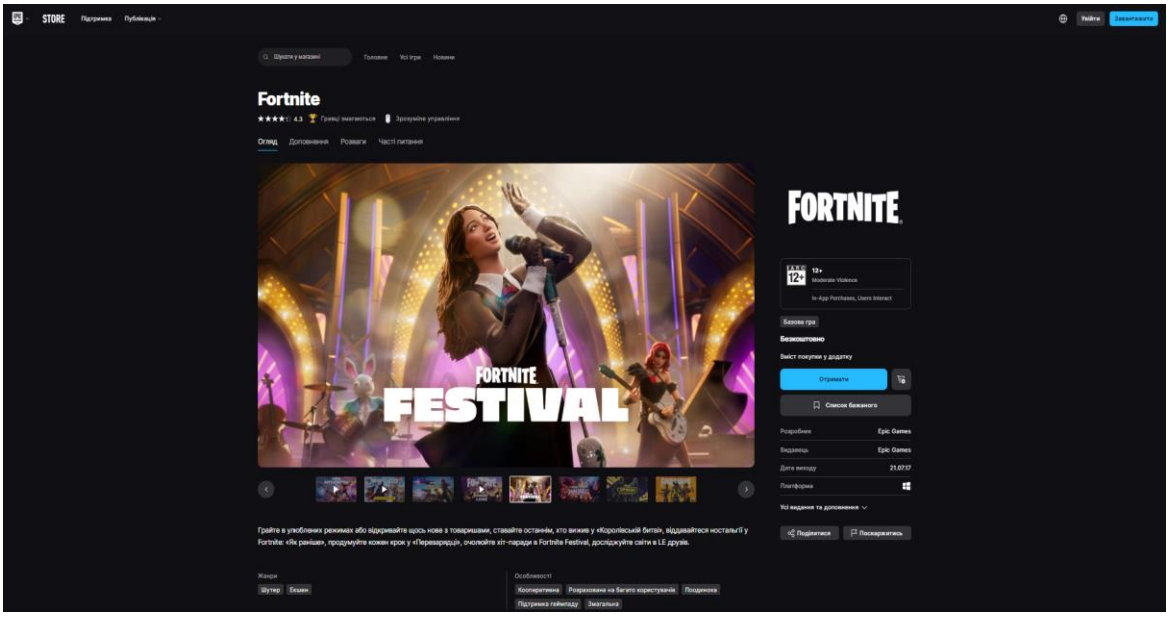


Рисунок 1.2 – Інтерфейс магазину Epic Games Store

GOG спеціалізується на іграх без DRM. Після покупки користувач отримує інсталятор або ключ. Магазин підтримує бібліотеку, бажане, відгуки, багатомовний інтерфейс.

Третім аналогом обрано GOG поданий у рисунку 1.3 — платформу, орієнтовану на продаж ключів і інсталяторів без прив’язки до одного лаунчера.

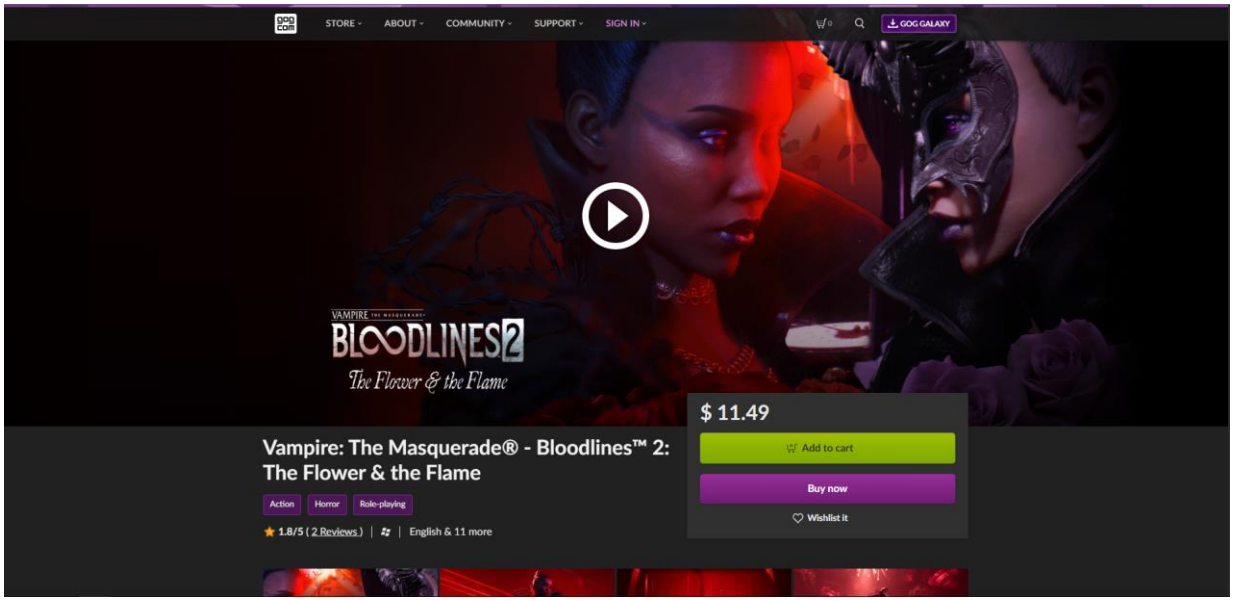


Рисунок 1.3 – Картка товару в магазині GOG

Порівняльна характеристика платформ представлено в таблиці 1.1.

Таблиця 1.1 – Порівняння аналогів і проєкту KeyGames

Критерій	Steam	Epic	GOG	KeyGames
Веб-каталог	+	+	+	+
Кошик/замовлення	+	+	+	+
Цифрові ключі	Частково	+	+	+
Відгуки	+	Обмежено	+	+
Список бажань	+	+	+	+
Адмін панель	Внутрішня	Внутрішня	Внутрішня	+(веб)
Власний backend	Закритий	Закритий	Закритйи	Spring Boot + PostgreSQL

Аналіз показує, що типовий інтернет-магазин ігор повинен містити: каталог із пошуком і фільтрами; обліковий запис користувача; кошик і оформлення замовлення; механізм оплати; доставку цифрового товару (ключ); соціальні функції (відгуки); персоналізацію (wishlist); адміністрування асортименту. Проєкт «KeyGames» реалізує зазначений набір у спрощеному навчальному варіанті з sandbox-оплатою.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Повна назва: вебсервіс дистрибуції комп'ютерних ігор «KeyGames».

Скорочена назва: KeyGames. Область застосування є продажу та видачі ліцензійних ключів комп'ютерних ігор через вебінтерфейс (діяльність інтернет-магазину цифрових товарів).

Середовище функціонування: персональний комп'ютер або мобільний пристрій із сучасним браузером;

серверна частина — Java 17 (Spring Boot 3.3.1[1]); база даних — PostgreSQL[3]. Локально — ПК розробника (http://localhost:8080); для публічної демонстрації — Vercel[9](frontend, https://key-games.vercel.app) та Railway[10](backend і БД).

1.2.2 Призначення розробки

Експлуатаційне призначення — автоматизація продажу ліцензійних ключів ігор через вебінтерфейс.

Функціональне призначення — надання користувачам засобів перегляду каталогу, купівлі, оплати та отримання ключів; адміністраторам — управління каталогом і замовленнями.

1.2.3 Вимоги до програмного забезпечення

Вхідні дані

До вхідних даних вебзастосунку «KeyGames» належать повідомлення та параметри, які система має приймати від користувача (покупця), адміністратора та клієнтських модулів браузера під час роботи інтернет-магазину. Вхідні дані передбачається подавати через HTML-форми та HTTP-запити до REST API[6]; на сервері вони повинні проходити перевірку (валідацію) і, за результатом обробки,

зберігатися в базі PostgreSQL або в локальному сховищі браузера (кошик).

За характером використання вхідні повідомлення планується поділити на разові (реєстрація, оформлення замовлення, sandbox-оплата, відгук) та тривалі (дані авторизації протягом сесії, позиції кошика в localStorage). Окремо виділяються доповнювані адміністратором дані каталогу ігор — метадані гри та файли обкладинок (multipart/form-data).

Перелік основних вхідних повідомлень із зазначенням форми представлення, періодичності надходження та джерела наведено в таблиці. 1.2.

Таблиця 1.2 – Перелік і опис вхідних повідомлень

Назва	форма	Періодичність	Джерело
Дані реєстрації	JSON (username, email, password)	За запитом	форма register.html
Дані авторизації	JSON + Basic Auth	За сесією	login.html
Параметри каталогу	query string (title, genre, price)	За запитом	games.html
Позиції кошика	JSON у localStorage	постійно	cart.js
Дані замовлення	JSON (items, nickname, email)	При checkout	checkout.html
Дані оплати	JSON (card, CVV, orderId)	При оплаті	payment.html
Відгук	JSON (rating, text)	За запитом	reviews.js
Дані гри (адмін)	JSON / multipart	За потреби	admin.html

Вихідними даними вебзастосунку «KeyGames» є повідомлення та структуровані відповіді, які система має формувати після обробки вхідних запитів і

передавати користувачеві, адміністратору або клієнтським модулям браузера. Вихідні дані передбачається повертати у форматі JSON через REST API[6] (Spring Boot[1] Controllers) і відображати на HTML-сторінках за допомогою JavaScript-модулів (api.js, games.js, profile-page.js тощо).

За призначенням вихідні повідомлення планується поділити на інформаційні (каталог ігор, деталі товару, агрегований рейтинг), транзакційні (замовлення зі статусом, ліцензійні ключі після оплати, підтвердження sandbox-платежу), інтерактивні (збережений відгук, оновлений wishlist) та службові (повідомлення про помилки валідації, авторизації або недостатній залишок ключів). У разі успішної обробки запиту HTTP-статус відповіді має бути 200 або 201; при порушенні правил доступу або некоректних даних — 400, 401, 403 або 404 із текстовим описом помилки в JSON.

Перелік основних вихідних повідомлень із зазначенням форми представлення та отримувача показано у таблиці 1.3.

Таблиця 1.3 – Перелік вихідних повідомлень

Назва	форма	Отримувач
Список ігор	JSON (GameResponse)	Браузер каталогу
Замовлення з ключами	JSON (OrderResponse)	Профіль orders
Статус оплати	JSON	Payment.html
Відгук та рейтиг	JSON	Картка гри
Повідомлення про помилки	JSON (message, status)	Rest клієнт

Опис функцій вебсервіс дистрибуції комп'ютерних ігор «KeyGames»:

– Реєстрація та вхід — створення облікового запису з роллю USER; автентифікація HTTP Basic[11].

– Каталог — перегляд, фільтрація за назвою, жанром, ціною; сортування за назвою; блок популярних ігор за рейтингом.

– Кошик — додавання/видалення позицій у localStorage (без серверного зберігання).

– Оформлення замовлення — створення запису orders зі статусом pending_payment.

– Оплата (sandbox) — валідація тестової картки; при успіху — статус paid, зменшення stock, прив'язка game_keys.

– Список бажань — збереження у БД wishlist_items.

– Відгуки — один відгук на гру від покупця; агрегація середнього рейтингу.

– Адміністрування — CRUD ігор, завантаження зображень, перегляд/видалення замовлень.

Часові характеристики

Відповідь REST API[6] на запит каталогу — до 2 с при локальному розгортанні. Оформлення замовлення та sandbox-оплата — до 5 с.

До розроблюваного вебсервісу поставлено вимоги до надійності

– валідація вхідних даних (Bean Validation, перевірка CVV — 3 цифри);

– контроль наявності ключів перед оплатою;

– обробка винятків через RestExceptionHandler;

– захист адмін- операційанотацією @PreAuthorize("hasRole('ADMIN')");

– хешування паролів BCrypt.

Умови експлуатації

– ОС сервера: Windows 10/11 або Linux;

– JDK 17+, PostgreSQL 14+;

– клієнт: сучасний браузер (Chrome, Firefox, Edge);

– мережа: HTTP, порт 8080; локальне розгортання на ПК;

– вимоги до умов праці та безпеки — згідно з розділом 5.

1.2.4 Вимоги до програмної документації

Під час розробки вебзастосунку «KeyGames» формується комплект

програмної документації відповідно до державних стандартів та вимог до кваліфікаційних робіт. Склад документів визначається призначенням продукту (дистрибуція цифрових ключів ігор) та необхідністю супроводу системи адміністратором і кінцевим користувачем.

До складу програмної документації включено:

– завдання на кваліфікаційну роботу — затверджений перелік вимог до функціоналу, надійності та умов експлуатації;

– пояснювальна записка — опис архітектури Spring Boot[1] + PostgreSQL[3], структури БД, логіки REST API[6] та інтерфейсу користувача;

– текст програми — оформлений вихідний код Java та JavaScript у додатках до записки;

– керівництво системного програміста — розділи 3.1–3.2 (інсталяція JDK/PostgreSQL, запуск mvn spring-boot:run, резервне копіювання БД);

– керівництво користувача — розділ 3.3 (перегляд каталогу, кошик, оплата, отримання ключів, wishlist, відгуки);

– опис REST API[6] — перелік ендпоінтів /games, /orders, /api/payments/sandbox, /api/wishlist, /api/games/{id}/reviews з параметрами запитів і відповідей (розділ 2.4);

– readme проєкту — коротка інструкція запуску в корені репозиторію.

Вимоги до тексту програми (коментарі в коді):

– кожний програмний модуль (клас сервісу, контролер, JS-модуль) має містити початковий блок коментарів із зазначенням призначення**,** складу вхідних **і** вихідних даних**,** обмежень та дати останньої зміни;

– серверні методи організовані за принципом однієї точки входу/виходу на рівні публічного API контролера.

Додаткові інструкції, що входять до комплекту документації:

– інструкція з інсталяції — створення БД game_store, налаштування application.yml, запуск на localhost:8080 (розділ 3.1);

– інструкція з наповнення каталогу — додавання ігор через admin.html,

завантаження обкладинок (розділ 3.3);

– стратегія резервного копіювання — pg_dump бази game_store та каталогу uploads/games/.

1.2.5 Техніко-економічні показники

Для об'єктивного визначення обсягу ресурсів, необхідних для створення та успішного впровадження вебзастосунку «KeyGames», виконується детальний розрахунок основних техніко-економічних показників розробки, що дозволяє раціонально планувати етапи проєктування. Це дозволяє ефективно планувати роботу. Через складність архітектурних рішень на базі Spring Boot[1] та потребу у високій якості кінцевого продукту дистрибуції цифрових ключів, оцінка витрат інтелектуальних та часових ресурсів виконавця стає обов'язковим етапом перед початком написання коду. Ресурси використовуються максимально продумано та раціонально. Загальний обсяг трудових витрат проєктної групи становить 180 людино-годин, що відповідає орієнтовно 1,0–1,2 людино-місяця роботи та повністю укладається в цикл створення навчального веб-застосунку в межах дипломного проєкту (граничне значення за методичкою — 180 год).

Детальні розрахунки витрат на оплату праці, електроенергію, амортизацію, собівартість та ціну розробки наведено в четвертому розділі пояснювальної записки. Результати оцінки техніко-економічних показників проєкту наведено у графічній частині 2026.КВР.122.421.19.00.00 ТБ.

1.2.6 Стадії та етапи розробки

У підрозділі 1.2.6 «Стадії та етапи розробки» визначено плановану послідовність робіт і очікуваний функціонал, який має бути реалізований у вебзастосунку «KeyGames» на кожному етапі життєвого циклу проєкту. Розділ 1.2 формує вимоги до майбутньої розробки; фактична реалізація описується в розділі 2 пояснювальної записки.

Загальний обсяг трудових витрат проєктної групи заплановано на рівні 180 людино-годин (узгоджено з підрозділом 1.2.5, таблицяею 1.4. та таблицею 4.1 економічного розділу).

Життєвий цикл проєкту передбачає дев’ять основних стадій, представлених у таблиці 1.4.

Таблиця 1.4 – Стадії та етапи розробки вебзастосунку KeyGames

№	Етап розробки	Тривалість, год
1	Аналіз предметної області та формування ТЗ	18
2	Проєктування архітектури та інтерфейсу	25
3	Розробка бази даних	16
4	Реалізація серверної частини (Spring Boot)	45
5	Верстка вебсторінок	20
6	Програмування клієнтської частини (JavaScript)	30
7	Інтеграція оплати та видачі ключів	8
8	Тестування та налагодження	15
9	Розгортання та оформлення документації	3
	Разом	180

1. Аналіз предметної області та формування технічного завдання (18 год). На початку було вивчено роботу відомих платформ цифрової дистрибуції ігор — Steam, Epic Games Store та GOG. Порівняно їхній функціонал і визначено, які можливості потрібні для власного інтернет-магазину: каталог з фільтрами, кошик, оформлення замовлення, тестова оплата, автоматична видача ліцензійних ключів, список бажань (wishlist), відгуки покупців і адміністративна панель. За результатами аналізу оформлено технічне завдання (підрозділи 1.2.1–1.2.5) та таблиці порівняння з аналогами.

2. Проєктування архітектури та інтерфейсу (25 год). Другим кроком стало проєктування структури застосунку. Складено карту сайту, варіанти використання (UC-01–UC-08), ER-діаграму бази даних і обрано технології: Java 17 та Spring Boot[1] 3.3.1 для серверної частини, PostgreSQL[3] для зберігання даних, HTML5[4], CSS3, JavaScript для клієнта. Підготовлено матеріали графічної частини (плакати СС, ДК, ТП) та таблиці 2.1–2.3 пояснювальної записки.

3. Розробка бази даних (16 год). У PostgreSQL створено базу game_store з таблицями users, games, orders, order_items, game_keys, wishlist_items, reviews. Налаштовано зовнішні ключі та обмеження цілісності, щоб замовлення, ключі та відгуки коректно пов'язувались між собою. Java-класи (Entity) відображають ці таблиці через Spring Data JPA.

4. Реалізація серверної частини (45 год). Серверну частину реалізовано на Spring Boot[1] за схемою Controller → Service → Repository → Entity. Створено REST API[6] для реєстрації, каталогу, замовлень, sandbox-оплати, wishlist і відгуків. Налаштовано Spring Security[2] (HTTP Basic Auth[11], ролі USER/ADMIN, хешування паролів BCrypt). Backend працює локально на http://localhost:8080, у production — на Railway[10].

5. Верстка вебсторінок (20 год). Паралельно з backend виконано верстку HTML-сторінок (головна, каталог, кошик, оформлення, оплата, вхід, профіль, wishlist, адмін-панель) та таблиць стилів CSS. Модуль layout.js формує спільне бічне меню. Опис верстки наведено в підрозділі 2.2.

6. Програмування клієнтської частини (30 год). До HTML-сторінок підключено JavaScript-модулі: api.js (запити на сервер), cart.js (кошик у localStorage), games.js, checkout.js, payment.js та інші. Клієнт взаємодіє з backend через REST API[6] у форматі JSON. Детальний опис — у підрозділах 2.4 та 2.4.1.

7. Інтеграція оплати та видачі ключів (8 год). Реалізовано sandbox-оплату (тестова картка 4242 4242 4242 4242) та автоматичну видачу ліцензійних ключів після зміни статусу замовлення на paid. Ключі відображаються в профілі користувача. Перевірено повний цикл: кошик → замовлення → оплата → ключі.

8. Тестування та налагодження (15 год). Проведено функціональне тестування

в браузері та перевірку API через Postman. Складено контрольний список (таблиці 2.16, п. 2.5) . Під час розгортання на Vercel[9]і Railway[10]усунено помилки CORS та авторизації. Результати описано в підрозділі 2.5.

9. Розгортання та оформлення документації (3 год). Frontend розміщено на Vercel[9](https://key-games.vercel.app), backend і база даних — на Railway[10]. Оформлено пояснювальну записку, графічну частину, інструкції з розділу 3 та економічний розділ. Загальний час усіх стадій — 180 годин, що не перевищує граничне значення 180 год за методичними рекомендаціями.

1.2.7 Порядок тестування та прийому

Перевірка працездатності вебзастосунку «KeyGames» здійснюється шляхом проведення комплексного тестування всіх функціональних модулів на відповідність вимогам технічного завдання. Це робиться ретельно. Під час проведення контрольних випробувань особлива увага приділяється стабільності взаємодії між серверною частиною на Spring Boot[1] та клієнтським інтерфейсом, побудованим на статичних HTML-сторінках і модулях JavaScript.

Для забезпечення об'єктивного контролю працездатності системи встановлюються наступні умови експлуатації:

– наявність запущеної СУБД PostgreSQL[3] з базою даних game_store та користувачем KeyGames;

– успішний запуск серверної частини на http://localhost:8080 (JDK 17, Maven, Spring Boot 3.3.1)[1];

– використання актуальних версій веб-браузерів Google Chrome, Mozilla Firefox або Microsoft Edge;

– наявність тестових облікових записів user/user123 (роль USER) та admin/admin123 (роль ADMIN);

– підготовка набору тестових даних у каталозі ігор, ключів активації та замовлень для перевірки різних сценаріїв покупки та адміністрування.

Усі компоненти системи мають бути активними та доступними в локальному

середовищі розробника.

У межах контрольного прикладу передбачено виконання покрокової демонстрації роботи проєкту, яка імітує повний життєвий цикл замовлення цифрової гри:

– Реєстрація нового користувача через форму register.html з перевіркою валідації полів та подальша авторизація на login.html для отримання доступу до профілю.

– Перегляд головної сторінки index.html з блоком популярних ігор та перехід до каталогу games.html для ознайомлення з асортиментом магазину «KeyGames».

– Використання панелі фільтрації для відбору ігор за жанром, назвою та ціновим діапазоном для перевірки точності роботи REST API[6] (GET /games).

– Вибір гри, відкриття модального вікна з детальним описом, рейтингом і відгуками, додавання позиції до кошика з візуальним підтвердженням дії.

– Редагування вмісту кошика на сторінці cart.html (зміна кількості або видалення позиції) та перехід до форми оформлення замовлення checkout.html.

– Введення нікнейму та email, створення замовлення зі статусом pending_payment і перехід на сторінку sandbox-оплати payment.html.

– Проведення тестової оплати карткою 4242 4242 4242 4242 з коректним CVV, перевірка зміни статусу на paid та відображення ліцензійних ключів у профілі profile.html.

– Додавання гри до списку бажань на wishlist.html та залишення відгуку після підтвердженої покупки для перевірки обмежень доступу.

– Авторизація під обліковим записом адміністратора для доступу до admin.html та orders.html.

– Додавання нової гри до каталогу через адміністративну форму з завантаженням обкладинки та перевіркою коректності відображення нового товару на сайті.

– Редагування існуючої гри (зміна ціни, опису або залишку stock) та

видалення тестової позиції з каталогу для перевірки CRUD-операцій.

– Перегляд усіх замовлень у таблиці orders.html та видалення тестового замовлення з перевіркою відновлення складських залишків і ключів для статусу paid.

За допомогою інструменту Postman додатково проводиться тестування REST API[6] для контролю коректності передачі облікових даних HTTP Basic Auth[11] та захисту приватних маршрутів сервера. Безпека є пріоритетом. Шляхом імітації запитів від неавторизованих осіб перевіряється стійкість системи до спроб несанкціонованого доступу до адміністративних операцій (POST /games, GET /orders) та створення відгуку без підтвердженої покупки (очікувана відповідь 403).

Приймання розробленого програмного продукту здійснюється комісією після успішного проходження всіх етапів контрольних випробувань без виявлення критичних помилок. Виконавець представляє пояснювальну записку, графічну частину та демонстраційну версію застосунку. Процедура прийому передбачає коротку доповідь про архітектурні рішення,демонстрацію роботи вебзастосунку в браузері за адресою https://key-games.vercel.app та відповіді на запитання щодо реалізації технічного завдання.

Система вважається успішно прийнятою, якщо:

– реалізовано всі функціональні можливості, описані у п. 1.2.3 технічного завдання (каталог, кошик, оплата, ключі, wishlist, відгуки, адмін-панель);

– час відгуку REST API[6] при виконанні основних операцій не перевищує встановлених у ТЗ норм (до 2 с для каталогу, до 5 с для оформлення та оплати);

– дані про замовлення, ключі активації та користувачів надійно зберігаються в базі PostgreSQL[3] без втрат;

– інтерфейс коректно відображається в сучасних браузерах на персональному комп'ютері;

– усі критичні пункти контрольного списку (табл 2.16, № 1–15) виконані зі статусом OK, а кількість дефектів, що блокують покупку або видачу ключів, дорівнює нулю.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка структури вебсервісу

Під час проєктування вебсервісу «KeyGames» першочерговим завданням виступає визначення його цільового призначення як сучасного інтернет-магазину цифрових ключів комп'ютерних ігор. Такий ресурс має бути зручним. Основна увага приділяється створенню інтуїтивно зрозумілого інтерфейсу, який забезпечує швидкий перехід від ознайомлення з каталогом до фінального отримання ліцензійного ключа після оплати. Шляхом аналізу потреб покупців та вимог технічного завдання було розроблено карту сайту, що відображає логічну ієрархію вебсторінок та формат подання інформації на кожному розділі.

В основу візуальної та програмної концепції покладено принципи мінімалізму, високої швидкодії та чіткого розмежування ролей користувачів. Дизайн і архітектура створюються поетапно. Спочатку формується карта сайту та перелік REST-модулів серверної частини, а після узгодження переходять до верстки сторінок (підрозділ 2.2) та програмування backend (підрозділ 2.4). Результатом цього етапу є набір функціональних вебсторінок і сервісів, що охоплюють усі сценарії взаємодії відвідувача з системою «KeyGames».

Застосунок побудовано за клієнт-серверною схемою з використанням REST API[6]. Це забезпечує чітке розмежування відповідальності між шарами. Взаємодія компонентів організована наступним чином:

– Клієнт (вебсторінки**)** — статичні HTML-сторінки, таблиці стилів CSS та модулі JavaScript (Vanilla JS). Кошик покупця зберігається в localStorage браузера до моменту оформлення замовлення.

– Сервер — Spring Boot 3.3.1[1]: REST-контролери, сервісний шар, JPA-репозиторії, Spring Security[2] (HTTP Basic Auth[11], BCrypt).

– База даних — PostgreSQL[3], схема game_store; обкладинки ігор — у каталозі uploads/games/

Зведена структура функціональних сторінок та основних модулів наведена у

таблиці 2.1.

Таблиця 2.1 – Структура функціональних сторінок та модулів вебсервісу KeyGames

№	Сторінка	Файл або компонент	Призначення	Рівень доступу
1	головна	index.html	Hero-блок, популярні ігри, навігація	Усі
2	Каталог ігор	games.html, games.js	Сітка ігор, фільтри, модальне вікно деталей	Усі
3	Реєстрація	register.html	Створення облікового запису USER	Усі
4	Вхід	login.html	HTTP Basic Auth, доступ до кошика	Усі
5	Кошик	cart.html, cart.js	Позиції в localStorage, підсумок суми	User/admin
6	Оформленя	checkout.html	Нікнейм, email, створення замовлення	User/admin
7	Оплата	payment.html	Sandbox-оплата, видача ключів	User/admin
8	Профіль	profile.html	Замовлення, статуси, ліцензійні ключі	User/admin
9	Список бажань	wishlist.html	Збережені ігри на сервері	User/admin
10	Адмін панель	admin.html, admin.js	CRUD ігор, завантаження обкладинок	admin
11	Замовлення	orders.html	Перегляд і видалення замовлень	admin
12	REST API	controller/, service/	Каталог, auth, orders, payments, reviews	За ролями
13	База даних	PostgreSQL game_store	Користувачі, ігри, ключі, відгуки	сервер

З точки зору відвідувача, сайт складається з логічно пов'язаних розділів: каталог і пошук → кошик і checkout → оплата → отримання ключів у профілі; для адміністратора — керування каталогом та моніторинг замовлень. Шляхом розробки

єдиної навігації (layout.js) забезпечується коректне відображення посилань залежно від ролі користувача. Це гарантує зручність користування.

Для формалізації варіантів використання вебсервісу визначено акторів системи наведено у таблиці 2.2, та перелік сценарії зведено у таблиці. 2.3.

UML-діаграма варіантів використання подана у графічній частині 2026.КВР.122.421.19.00.00 ДВ.

Таблиця 2.2 – Актори системи

Актор	Роль у системі	Основні можливості
Гість	Неавторизований відвідувач	Перегляд головної та каталогу, реєстрація
Користувач (user)	Зареєстрований покупець	Кошик, оплата, ключі, wishlist, відгуки
Адміністратор (admin)	Оператор магазину	CRUD ігор, перегляд і видалення замовлень

Таблиця 2.3 – Варіанти використання програми KeyGames

Код	Назва сценарію	Актор	Результат
UC-01	Перегляд каталогу ігор	Гість, USER	Відображення списку ігор із фільтрами
UC-02	Реєстрація / авторизація	Гість → USER	Доступ до захищених сторінок
UC-03	Додавання до кошика та wishlist	USER	Позиції в localStorage / БД
UC-04	Оформлення замовлення	USER	Замовлення зі статусом pending_payment

UC-05	Sandbox-оплата та отримання ключів	USER	Статус paid, ключі в профілі
UC-06	Написання відгуку	USER	Відгук після підтвердженої покупки
UC-07	Адміністрування каталогу	ADMIN	CRUD ігор, upload зображень
UC-08	Перегляд і видалення замовлень	ADMIN	Контроль транзакцій, відкат stock

Шляхом групування сценаріїв за акторами вдалося забезпечити повне покриття вимог технічного завдання. Кожен варіант використання реалізовано у вигляді HTML-сторінок, REST-ендпоінтів і сервісів Spring Boot[1], що спрощує тестування та супровід системи.

2.2 Створення та верстка сторінок вебсервісу

Після завершення етапу проєктування логічної структури (підрозділ 2.1) було виконано безпосередню реалізацію клієнтської частини інтернет-магазину цифрових ігор «KeyGames». Основним завданням цього етапу виступало перетворення затвердженої карти сайту у функціональні веб-сторінки з високим рівнем інтерактивності та зв'язком із REST API[6] серверної частини. Для розробки використовувалось середовище IntelliJ IDEA / Visual Studio Code, що забезпечує підтримку Java, HTML[4] та JavaScript[5] і інтеграцію з системою контролю версій Git.

В основу клієнтського стеку покладено статичні HTML-сторінки, таблиці стилів CSS та модулі Vanilla JavaScript без використання SPA-фреймворків. Швидкість завантаження є пріоритетом. Сторінки обслуговуються вбудованим сервером Spring Boot[1] із каталогу src/main/resources/static/, що спрощує локальне

розгортання на localhost:8080. Це значно прискорює цикл розробки та забезпечує легкість супроводу коду.

Верстка всіх сторінок виконувалась із використанням семантичних тегів HTML5[4], таких як <header>, <main>, <nav>, <section> та <footer>. Такий підхід покращує структуру документа та робить інтерфейс більш доступним. Усі сторінки побудовані за принципом адаптивності (media queries у CSS), що гарантує коректне відображення на персональних комп'ютерах із різною роздільною здатністю екрана. Побудова виконується дуже ретельно.

Для надання інтерфейсу сучасного вигляду та забезпечення надійної взаємодії з backend застосовано такі клієнтські модулі:

– layout.js — спільна бічна панель навігації, відображення посилань залежно від ролі, система сповіщень (рис.2.1);

– api.js — уніфікований HTTP-клієнт на базі fetch, збереження облікових даних Basic Auth у localStorage;

– auth.js — вихід із системи та оновлення стану sidebar після авторизації;

– cart.js — управління кошиком у localStorage (додавання, зміна кількості, підсумок);

– games.js — завантаження каталогу, фільтрація, модальне вікно деталей гри;

– reviews.js — відображення та відправка відгуків із перевіркою прав на сервері;

– checkout.js, payment.js, profile-page.js, wishlist-page.js, orders.js, admin.js — логіка відповідних сторінок.

Технології працюють злагоджено. Логічна структура сторінок наведена у таблиці 2.1; нижче описано реалізацію інтерфейсу з ілюстраціями.

Карта сайту та навігація

Ключовим елементом навігації виступає модуль layout.js, який підключається на кожній HTML-сторінці. Він формує бічну панель із посиланнями на каталог, кошик, wishlist, профіль, а також (для ADMIN) — на admin.html і orders.html. Для

гостя частина посилань прихована або перенаправляє на login.html. Це забезпечує розмежування доступу без окремого SPA-роутера.

Структура сайту наведа на рисунку 2.1

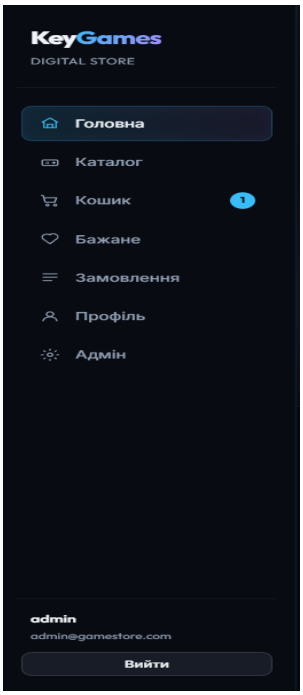


Рисунок 2.1 – Карта сайту вебзастосунку KeyGames

Головна сторінка (index.html, main.js)

Сторінка index.html виступає візитною карткою магазину «KeyGames». У блоці <main> розміщено hero-секцію з назвою сервісу, коротким описом дистрибуції ключів та кнопкою переходу до каталогу. Модуль main.js при завантаженні виконує запит GET /games/popular?limit=5 і динамічно формує картки популярних ігор із рейтингом. Для візуального акценту застосовано CSS-класи feature-pills (каталог, ключі, безпека). Головна сторінка показана на рисунку 2.2

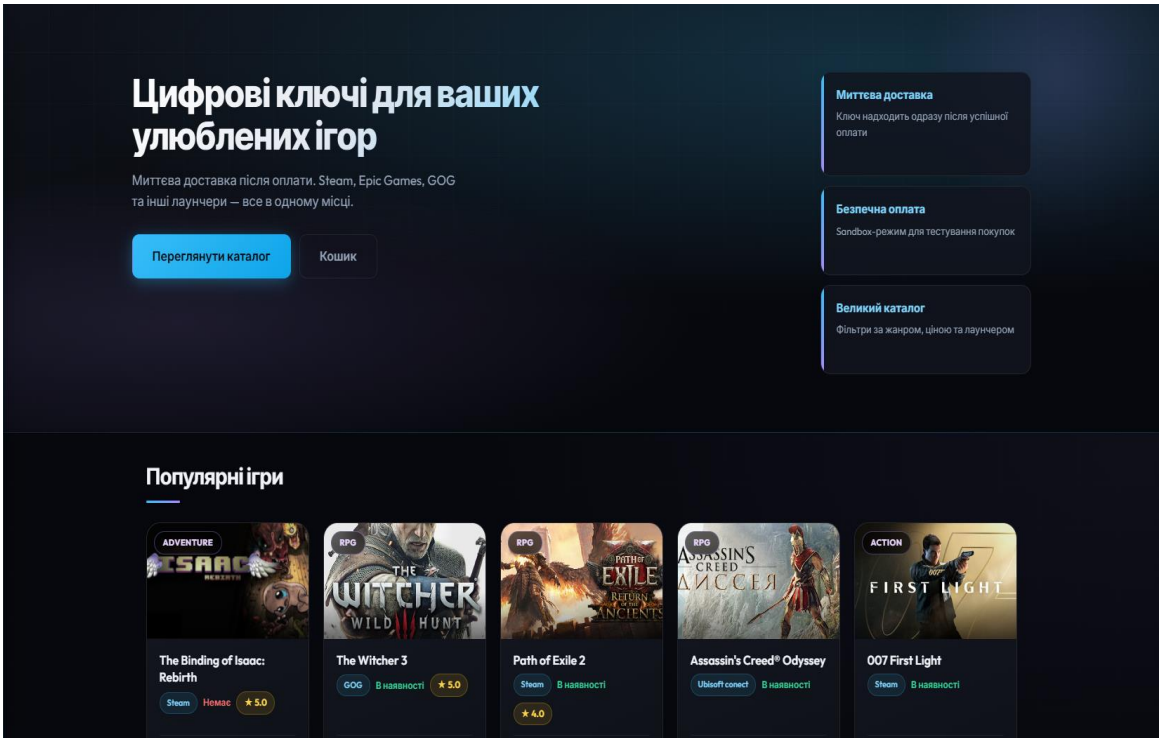


Рисунок 2.2 – Головна сторінка вебзастосунку KeyGames

Каталог ігор (games.html, games.js)

Розділ каталогу (games.html) є найскладнішим модулем клієнтської частини. Його верстка базується на двоколонковій структурі: зліва — панель фільтрів, справа — сітка карток ігор. Модуль games.js завантажує дані через GET /games з query-параметрами title, genre, minPrice, maxPrice.

Панель фільтрів реалізована через елементи <input> та <select>. Кожна зміна параметрів викликає повторний запит до API та оновлення сітки без перезавантаження сторінки. Кожна гра у каталозі представлена карткою, яка містить:

– зображення обкладинки з uploads/games/ або placeholder;
– назву, жанр, платформу (Steam, Epic тощо) та ціну;
– індикатор середнього рейтингу з таблиці reviews.

Каталог з фільтрами та пошуком ігор відображений на рисунку 2.3

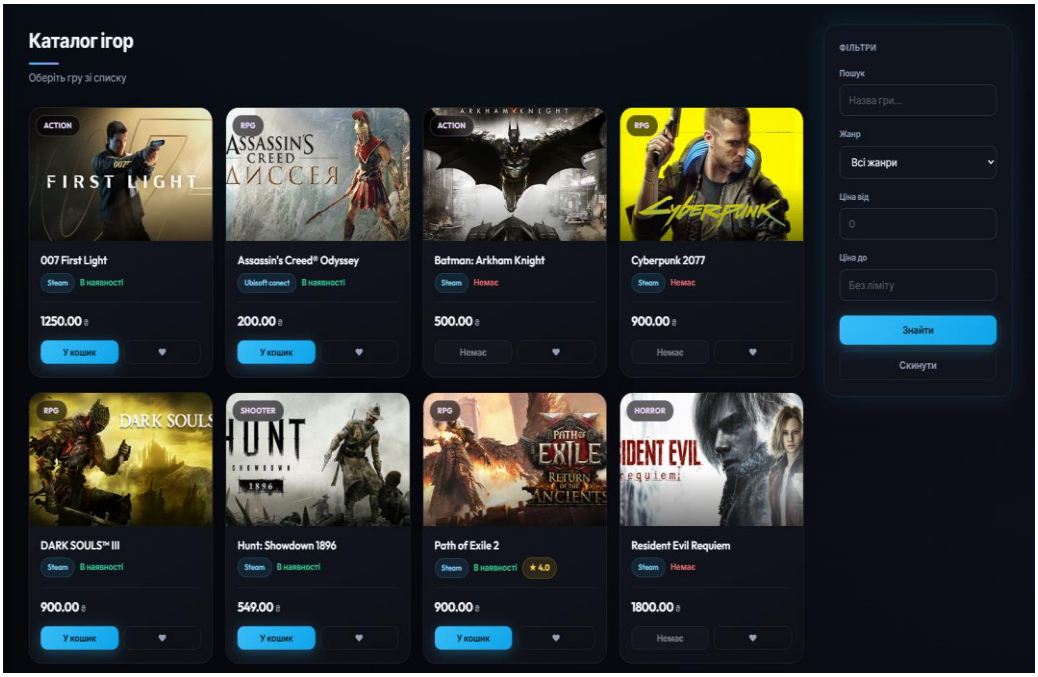


Рисунок 2.3 – Сторінка каталогу ігор з панеллю фільтрів

Клік по картці відкриває модальне вікно з повним описом гри. У модалці підключено reviews.js: відображаються відгуки покупців, форма додавання відгуку (після авторизації) та кнопки «Додати в кошик» і «Wishlist». Додавання в кошик викликає функції з cart.js.Модельне вікно гри зображенно у рисунку 2.4

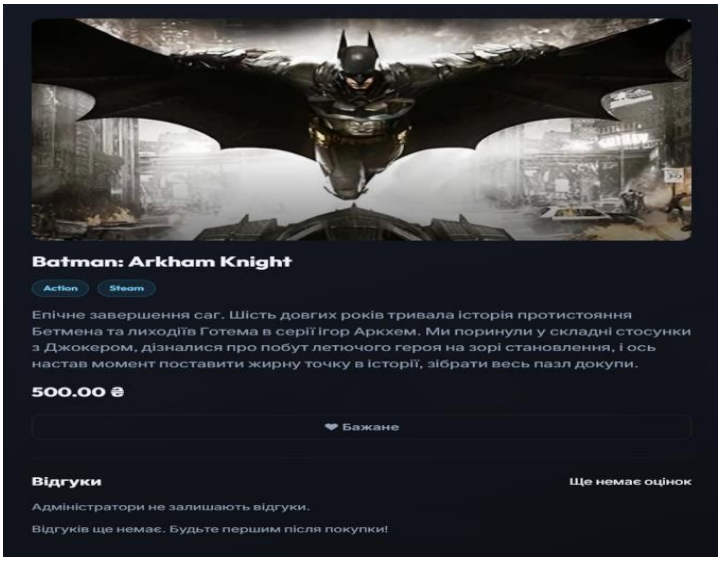


Рисунок 2.4 – Модальне вікно деталей гри та відгуків

Особлива увага приділена формам облікового запису. Сторінка register.html

містить поля username, email, password; модуль register.js відправляє POST /api/auth/register і при успіху перенаправляє на вхід. Форма login.html реалізована у login.js: після POST /api/auth/login облікові дані зберігаються для заголовка Authorization: Basic … у подальших запитах через api.js. Верстка форм включає вбудовану валідацію на стороні клієнта (обов'язкові поля, формат email).Форма реєстрації наведена у рисунку 2.5, також на рисунку 2.6 зображено вікно авторизації користувача

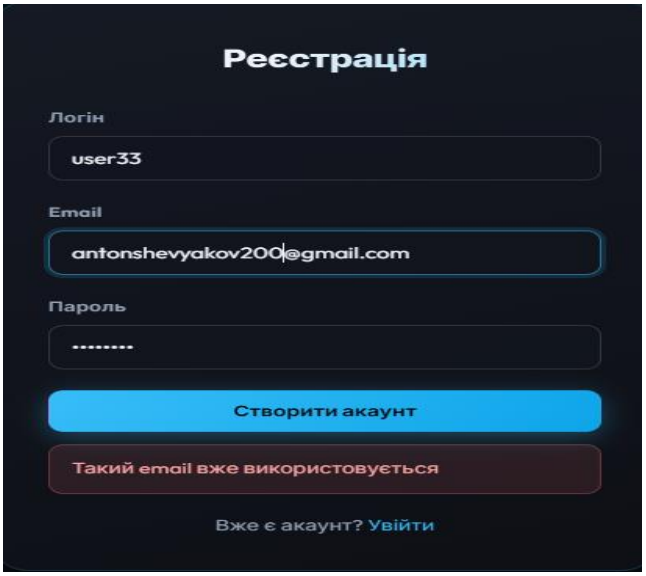


Рисунок 2.5 – Форма реєстрації користувача

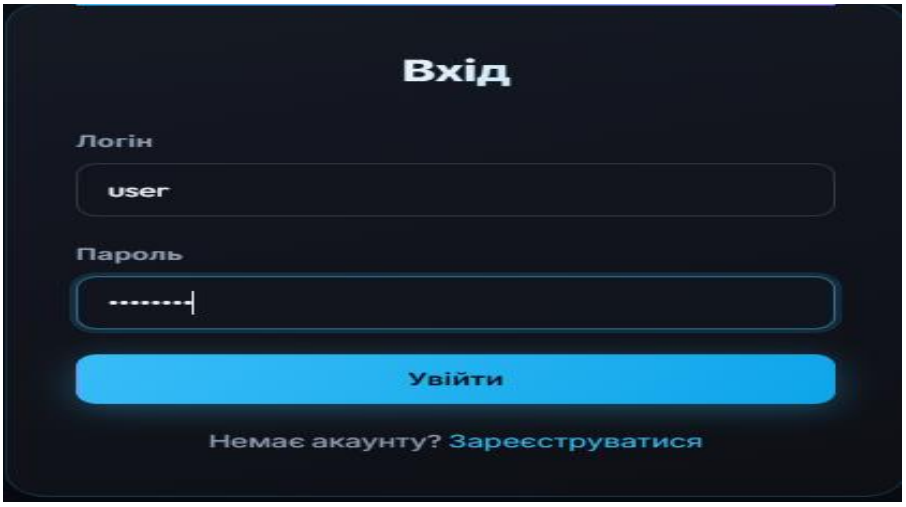


Рисунок 2.6 – Вікно авторизації користувача

Сторінка кошика (cart.html) відображає позиції з localStorage. Модуль cart-page.js будує таблицю товарів, дозволяє змінювати кількість або видаляти рядки,

автоматично перераховує загальну суму. Кнопка «До оформлення» доступна лише авторизованим користувачам; інакше виконується перенаправлення на login.html. Дані кошика зберігаються між сесіями браузера, що підвищує зручність для покупця. Сторінка кошика покупця відображенно у рисунку 2.7

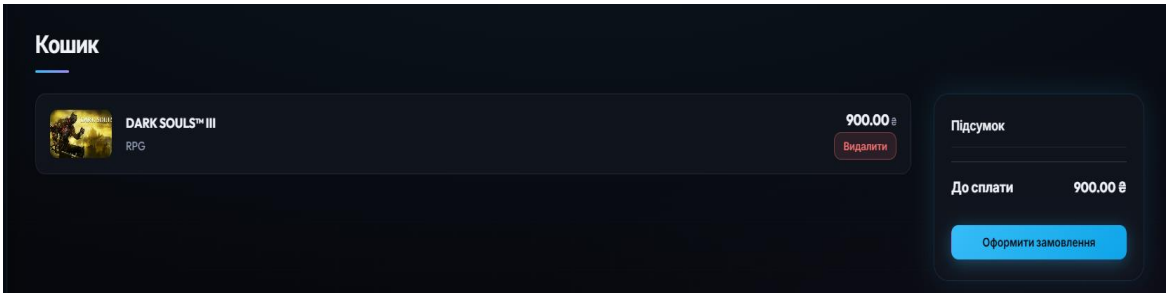


Рисунок 2.7 – Сторінка кошика покупця

Сторінка оформлення містить форму з полями нікнейм та email**,** блок підсумку позицій з кошика та кнопку підтвердження. Модуль checkout.js збирає дані, формує JSON і відправляє POST /orders. При успіху користувач перенаправляється на payment.html з ідентифікатором замовлення. Статус замовлення на сервері — pending_payment. Сторінка оформлення замовленнь надана у рисунку 2.8

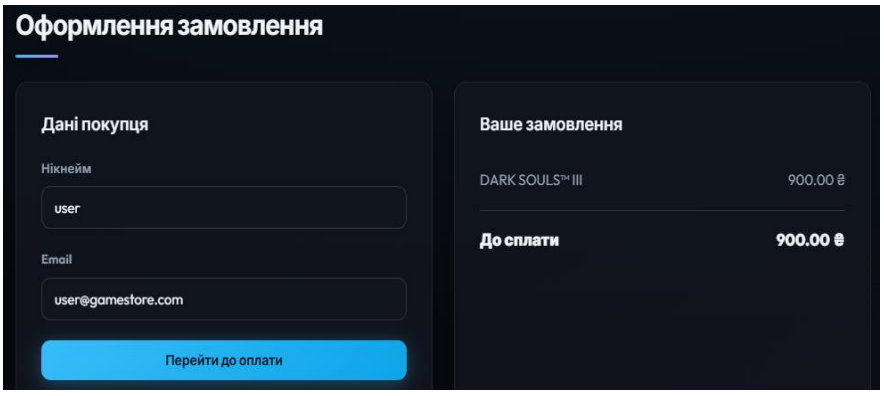


Рисунок 2.8 – Сторінка оформлення замовлення

Сторінка оплати імітує роботу платіжного шлюзу. Модуль payment.js передає номер картки, термін і CVV на POST /api/payments/sandbox. Сервер валідує CVV (3 цифри); тестова картка — 4242 4242 4242 4242. При успіху статус змінюється на paid, ключі з таблиці game_keys прив'язуються до позицій замовлення. Інтерфейс

відображає повідомлення про успіх або помилку валідації.Сторінка оплати відображенна на рисунку 2.9

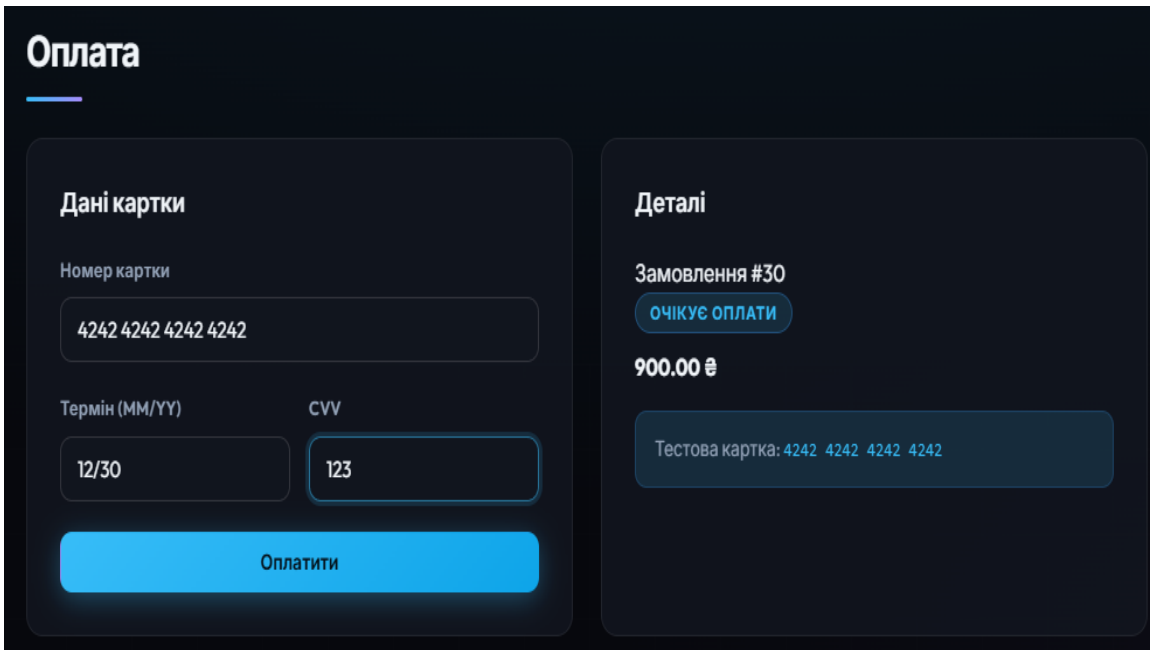


Рисунок 2.9 – Сторінка sandbox-оплати замовлення

Сторінка профілю побудована як структурований розділ із блоками замовлень. Модуль profile-page.js завантажує історію покупок поточного користувача; для оплачених замовлень відображаються ліцензійні ключі (key_code) по кожній позиції. Дані оновлюються динамічно після успішної оплати без перезавантаження всього застосунку. Профіль користувача з ключами зображенна на рисунку 2.10

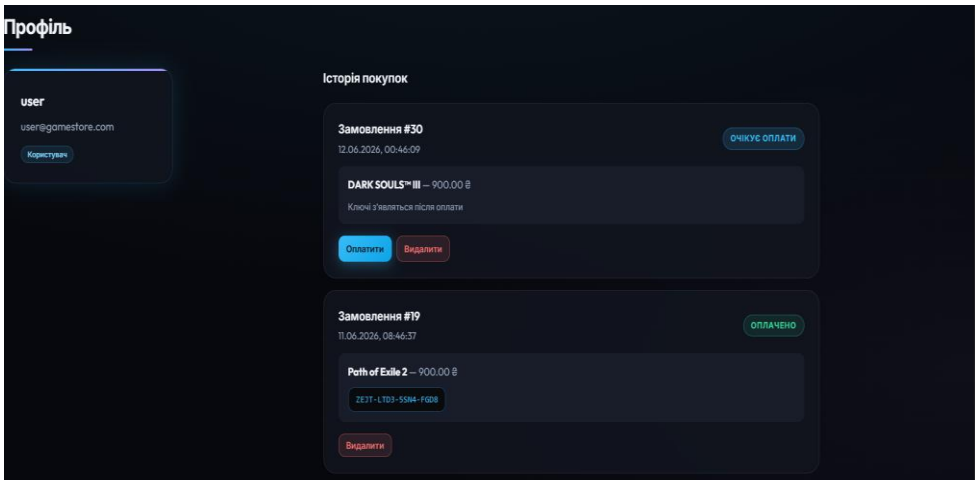


Рисунок 2.10 – Профіль користувача з ліцензійними ключами

Список бажань (wishlist.html, wishlist-page.js)

Wishlist реалізовано як серверний модуль (на відміну від кошика в localStorage). wishlist-page.js виконує GET /api/wishlist і відображає список обраних ігор; кнопка видалення викликає DELETE. Це дозволяє зберігати бажане між пристроями після входу в обліковий запис. Список бажаних ігор відображений у рисунку 2.11

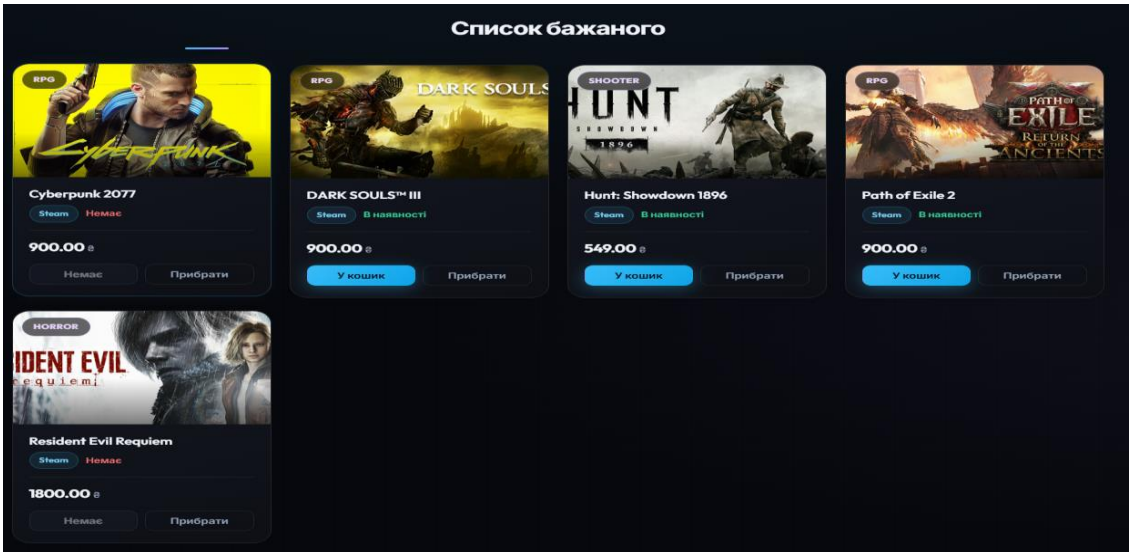


Рисунок 2.11 – Сторінка списку бажань

Адміністративна панель має власну структуру: форма додавання/редагування гри та інтерактивна таблиця каталогу. Модуль admin.js реалізує CRUD через REST (POST, PUT, DELETE /games), завантаження обкладинок через multipart/form-data. Доступ захищено роллю ADMIN на сервері; при спробі входу звичайного user відображається помилка 403. Інтерфейс адмін панелі показано у рисунку 2.12

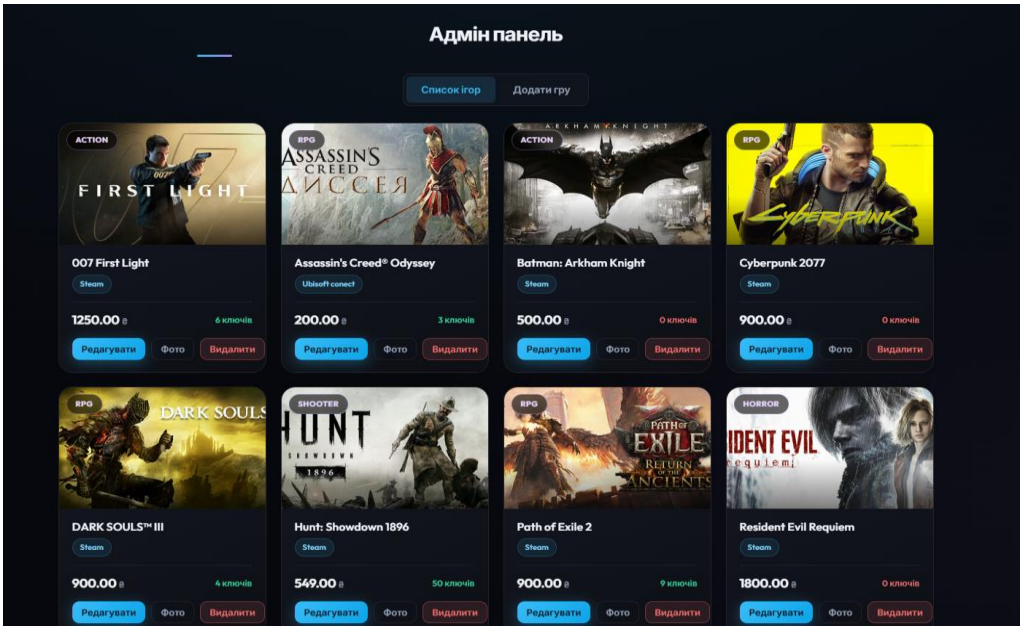


Рисунок 2.12 – Інтерфейс адміністративної панелі

Додавання та редагування ігор здійснюється через форму на тій самій сторінці. Вона містить:

– текстові поля для назви, опису, жанру, платформи, ціни та залишку stock;
– модуль завантаження зображення через <input type="file">;
– кнопки «Зберегти» та «Скасувати» з валідацією обов'язкових полів.

Форма додавання грі відображенна у рисунку 2.13, також форма редагування гри відображена на рисунку 2.14.

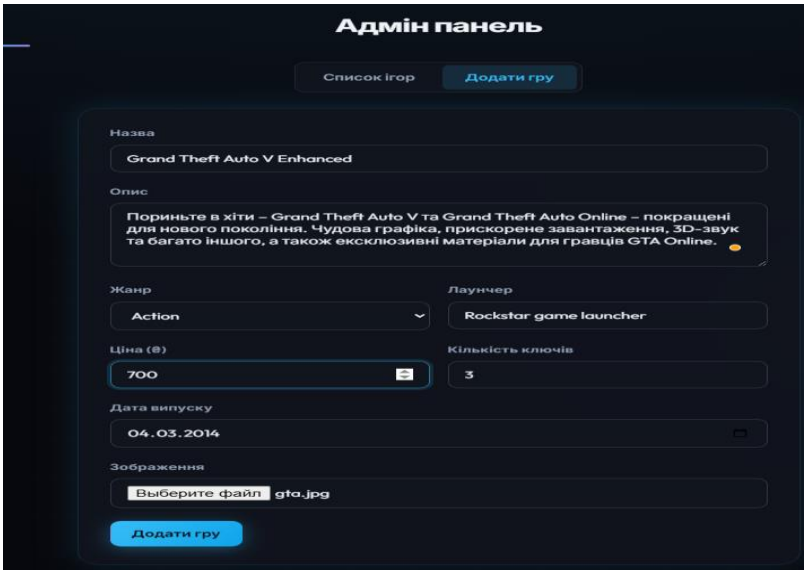


Рисунок 2.13 – Форма додавання гри в адмін панелі

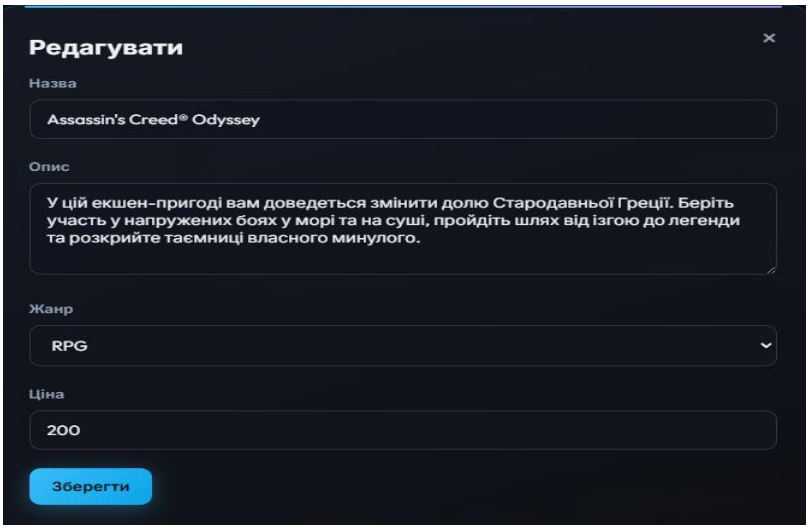


Рисунок 2.14 – Форма редагування гри в адмін панелі

Замовлення адміністратора (orders.html, orders.js)

Розділ orders.html дозволяє адміністратору переглядати повний список транзакцій. Модуль orders.js завантажує всі замовлення, відображає ID, дані клієнта, статус і дату. Операція видалення для оплачених замовлень супроводжується відкатом stock і звільненням ключів на сервері. Статуси візуально розрізняються CSS-класами (pending_payment, paid, failed). Сторінка перегляду замовлень адміністратором показана у рисунку 2.15.

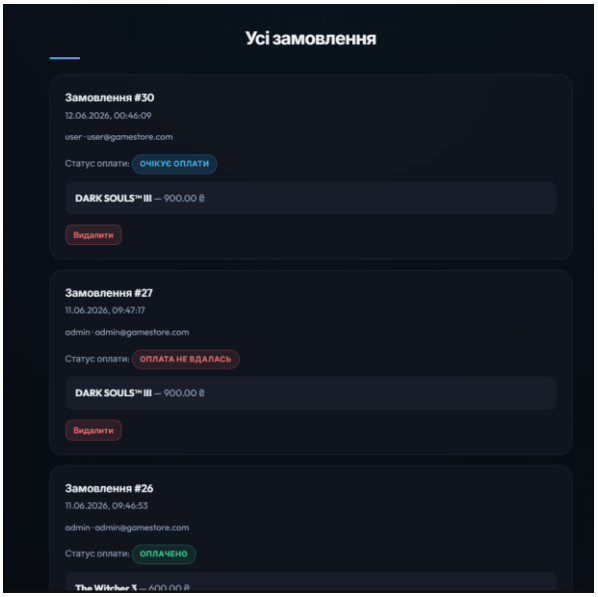


Рисунок 2.15 – Сторінка перегляду замовлень адміністратором

Результатом виконання етапу проєктування та верстки інтерфейсу стали повністю функціональні HTML-сторінки, які забезпечують комфортну роботу покупця та адміністратора. Всі клієнтські модулі інтегровані з REST API[6] Spring Boot[1] через api.js, що дозволяє системі «KeyGames» працювати як цілісний програмний продукт. Фрагменти вихідного коду JavaScript[5] та Java наведено у додатках до пояснювальної записки.

2.3 Розробка структури бази даних вебсервісу

Для забезпечення стабільної роботи інтернет-магазину «KeyGames» та надійного збереження інформації про асортимент ігор, замовлення та ліцензійні ключі у межах проєкту виконано ретельне проєктування архітектури бази даних. Вона виступає ядром web-сайту. Шляхом глибокого аналізу предметної області дистрибуції цифрових товарів вдається відобразити торговельні об'єкти у нормалізовану реляційну модель PostgreSQL[3], що є критично важливим для швидкодії каталогу та цілісності транзакцій.

В основу технологічного рішення покладено PostgreSQL — систему керування реляційними базами даних, яка забезпечує сувору цілісність даних через зовнішні ключі та транзакції. Завдяки використанню Spring Data JPA з'являється можливість відображати таблиці у Java[6]-класах (Entity) без ручного написання SQL для типових операцій. Це суттєво прискорює розробку backend-частини сайту.

Під час проєктування бази даних реалізовано наступні ключові етапи:

– за допомогою аналізу бізнес-процесів визначено основні таблиці для функціонування магазину «KeyGames»;

– шляхом деталізації полів установлено типи даних та обов'язковість заповнення;

– через налаштування зовнішніх ключів організовано цілісність замовлень, ключів і відгуків;

– за допомогою унікальних індексів забезпечено один відгук на гру та унікальність логінів.

У процесі проєктування були визначені типи взаємозв’язків між таблицями бази даних game_store: зв’язок «один-до-багатьох» між users і orders, між orders і order_items, між games і order_items, між games і game_keys; зв’язок «багато-до-багатьох» між users і games реалізовано таблицею wishlist_items; таблиця reviews пов’язує користувача з грою (один відгук на пару «користувач — гра»). Опис полів основних таблиць наведено в таблицях 2.4–2.10. ER-діаграма відображає взаємозв’язки між основними сутностями системи (користувачі, ігри, замовлення, ключі) і подана на плакаті 2026.КВР.122.421.19.00.00 БД у графічній частині кваліфікаційної роботи.

Таблиця 2.4 – Таблиця users

№	Поле	тип	опис
1	id	BIGSERIAL PK	Ідентифікатор
2	username	VARCHAR(100) UNIQUE	Логін
3	password	VARCHAR(255)	BCrypt-хеш
4	email	VARCHAR(255) UNIQUE	Email
5	role	VARCHAR	USER / ADMIN

Таблиця 2.5 – Таблиця games

№	Поле	Тип	Опис
1	id	BIGSERIAL PK	ID гри
2	title	VARCHAR(255)	Назва
3	description	TEXT	Опис
4	genre	VARCHAR(100)	Жанр
5	platform	VARCHAR(100)	Лаунчер
6	price	NUMERIC(10,2)	Ціна
7	stock	INTEGER	Наявність
8	release_date	DATE	Дата виходу
9	is_active	BOOLEAN	Активність у каталозі

10	image_url	VARCHAR(500)	Шлях до зображення

Таблиця 2.6 – Таблиця orders

№	поле	Тип	Опис
1	id	BIGSERIAL PK	ID замовлення
2	customer_name	VARCHAR	нікнейм
3	customer_email	VARCHAR	Email
4	created_at	TIMESTAMP	Дата створення
5	status	VARCHAR(50)	pending_payment / paid / failed
6	user_id	BIGINT	ID користувача

Таблиця 2.7 – Таблиця order_items

№	Поле	Тип	Опис
1	id	BIGSERIAL PK	
2	Order_id	FK → orders	
3	Game_id	FK → games	
4	quantity	INTEGER	Кількість
5	unit_price	NUMERIC(10,2)	Ціна на момент покупки

Таблиця 2.8 – Таблиця game_keys

№	Поле	Тип	Опис
1	Id	BIGSERIAL PK	
2	Game_id	FK → games	
3	Key_code	VARCHAR(64) UNIQUE	Ключ активації
4	assigned	BOOLEAN	Чи видано
5	Order_item_id	FK → order_items	Прив’язка до покупки

Таблиця 2.9 – Таблиця wishlist_items

№	Поле	Тип	Опис
1	Id	BIGSERIAL PK	
2	User_id	FK → users	
3	Game_id	FK → games	UNIQUE(user_id, game_id)

Таблиця 2.10 – Таблиця reviews

№	Поле	Тип	Опис
1	Id	BIGSERIAL PK	
2	User_id	FK → users	
3	Game_id	FK → games	
4	Rating	INTEGER	Оцінка від 1 до 5
5	Text	TEXT	Текст відгуку

Добре спроєктована структура бази даних дозволяє web-сайту «KeyGames» ефективно обробляти запити каталогу, фіксувати замовлення та видавати ключі без втрати цілісності. Це фундамент стабільності проєкту.

Реалізація backend-складової інтернет-магазину «KeyGames» базується на принципах модульної архітектури Spring Boot[1], що забезпечує чіткий розподіл бізнес-логіки, доступу до даних та маршрутизації HTTP-запитів. Такий підхід робить систему надійною. Завдяки використанню фреймворку Spring Boot 3.3.1[1] на Java 17[6] створено серверний застосунок, де кожна функціональна область (користувачі, ігри, замовлення, оплата) виділена в окремий пакет controller/service/repository. Взаємодія між web-сторінками та сервером побудована за стандартом REST API[6] з форматом JSON, що гарантує сумісність із статичним frontend.

В основу архітектури покладено клієнт-серверний механізм, де сервер виступає проміжним шаром між PostgreSQL[3] та інтерфейсом користувача в браузері. Загальна схема архітектурної взаємодії компонентів наведена на плакаті 2026.КВР.122.421.19.00.00 СС.

2.4 Програмування вебсервісу

Після того як було виконано верстку HTML-сторінок (підрозділ 2.2) і спроєктовано структуру бази даних PostgreSQL (підрозділ 2.3), настав етап програмування вебсервісу «KeyGames». На цьому етапі статичні сторінки та схема БД перетворюються на працюючий програмний продукт, з яким може взаємодіяти реальний користувач: переглядати каталог, оформлювати замовлення, отримувати ключі після оплати. Метою програмування є реалізація всіх функцій, описаних у технічному завданні (підрозділ 1.2.3), та забезпечення надійного обміну даними між браузером і сервером. Вебсервіс побудовано за клієнт-серверною архітектурою: клієнт відповідає за відображення інформації, сервер — за обробку запитів і збереження даних у базі.

Інтернет-магазин «KeyGames» функціонує як вебзастосунок з REST API[6]. Користувач працює зі звичайним сайтом у браузері, а всі операції з даними (реєстрація, каталог, замовлення, оплата, wishlist, відгуки) виконуються через HTTP-запити на серверну частину. Такий підхід є поширеним для сучасних web-рішень, оскільки дозволяє розділити frontend і backend і незалежно їх розвивати або змінювати.

Програмна реалізація розділена на дві самостійні частини, які описуються в окремих підрозділах:

– 2.4.1 Написання клієнтської частини — HTML5[4], CSS3, JavaScript[5] (Vanilla JS);

– 2.4.2 Написання серверної частини — Java 17, Spring Boot 3.3.1[1], Spring Data JPA, Spring Security[2], PostgreSQL[3].

Зведений перелік технологій наведено в таблиці 2.11.

Таблиця 2.11 – Технологічний стек програмування вебсервісу KeyGames

Компонент	Технології	Призначення
Клієнтська	HTML5, CSS3,	інтерфейс

частина	JavaScript	користувача в браузері
Серверна частина	Java 17, Spring Boot 3.3.1	REST API, бізнес-логіка
База даних	PostgreSQL	Зберігання користувачів, ігор, замовлень, ключів
Збірка backend	Maven	Компіляція та запуск Spring Boot
Авторизація	HTTP Basic Auth, BCrypt	Вхід,ролі USER/ADMIN
Розгортання	Vercel (frontend), Railway (backend)	Робота в мережі Інтернет
Обмін даними	REST API, JSON	Запити між клієнтом і сервером

Принцип взаємодії клієнта і сервера Взаємодія компонентів організована за стандартом REST (Representational State Transfer). Клієнтська частина відправляє HTTP-запити з певною адресою (endpoint) і методом:

- GET — отримати дані (наприклад, список ігор або профіль);
- POST — створити новий запис (реєстрація, замовлення, оплата);
- PUT — оновити існуючі дані (редагування гри адміном);
- DELETE — видалити запис (гра, замовлення, позиція wishlist).

Сервер обробляє запит, звертається до бази даних за потреби і повертає відповідь у форматі JSON — текстовому форматі, який легко читається і JavaScript, і Java. Якщо сталася помилка (неправильний пароль, немає прав доступу, не знайдено

гру), сервер повертає код помилки (400, 403, 404) і короткий опис у JSON.

Середовище та інструменти розробки Програмування виконувалось на персональному комп'ютері під керуванням Microsoft Windows 10/11. Для написання коду використовувались:

- IntelliJ IDEA — основне середовище для Java/Spring Boot[1], налагодження backend, роботи з Maven;
- Visual Studio Code — редагування HTML[4], CSS, JavaScript;
- PostgreSQL — локальна база даних game_store під час розробки;
- Git — контроль версій вихідного коду проєкту;
- Postman — перевірка REST API[6] без браузера;
- Google Chrome — тестування інтерфейсу та роботи fetch-запитів.

Етапи програмування вебсервісу Роботу над кодом виконано в логічній послідовності, що відповідає структурі проєкту:

1. Серверна частина (Java/Spring Boot[1]) — спочатку реалізовано REST API, моделі даних (Entity), сервіси та контролери, щоб був «фундамент» для клієнта.
2. Клієнтська частина (JavaScript) — підключено запити до API, авторизацію, каталог, кошик, оформлення замовлення та оплату.
3. Інтеграція — перевірено повний сценарій: реєстрація → вхід → кошик → замовлення → sandbox-оплата → ключі в профілі.
4. Розгортання — frontend викладено на Vercel[9], backend і БД — на Railway[10]; у vercel.json налаштовано проксі /api/* для коректної роботи без CORS.

Такий порядок дозволив поступово перевіряти кожен модуль і не накопичувати помилки до кінця розробки.

У процесі навчальної розробки вебсервіс працює в двох режимах: Локальний режим — усе на одному комп'ютері: PostgreSQL[3], Spring Boot[1] на порту 8080, сторінки з папки static/. Зручно для написання коду[8], налагодження та підготовки до демонстрації на захисті.

Мережевий режим (production) — frontend на https://key-games.vercel.app, backend на https://keygames-production.up.railway.app. Користувач відкриває сайт у

браузері; JavaScript звертається до /api/, а Vercel[9]перенаправляє ці запити на сервер Railway[10]. База PostgreSQL[3] також розміщена на Railway.

Обидва режими використовують один і той самий вихідний код — змінюється лише конфігурація підключення (файли config.js, application.yml, vercel.json).

2.4.1 Написання клієнтської частини

Мови та технології: HTML5[4], CSS3, JavaScript (Vanilla JS, без React/Vue/Angular).

Середовище розробки: IntelliJ IDEA, Visual Studio Code.

Розташування файлів: src/main/resources/static/ (html, css, js).

Клієнтська частина — це те, що користувач бачить у браузері: сторінки сайту і скрипти, які завантажують дані з сервера. Окремий frontend-фреймворк не використовувався — лише статичні HTML-сторінки і JavaScript-файли, підключені до них.

Ланцюжок такий: HTML-сторінка → JavaScript → REST API → Spring Boot → PostgreSQL.

Складну логіку (оплата, ключі, перевірка прав) виконує сервер. JavaScript показує дані, обробляє кліки і форми, відправляє запити через fetch().

Основні JavaScript-модулі перелік показаний у таблиці. 2.12.

Таблиця 2.12 – Клієнтські JavaScript-модулі

Модуль	Сторінка	Призначення
config.js	усі	Адреса API (localhost / Vercel / Railway)
api.js	усі	HTTP-запити, авторизація, обробка помилок
layout.js	усі	Бічне меню, сповіщення (toast)
auth.js	усі	Вихід з облікового запису
main.js	index.html	Популярні ігри на головній
games.js	games.html	Каталог, фільтри, модальне вікно

reviews.js	games.html	Відгуки в модальному вікні
cart.js	games, cart	Кошик у localStorage
cart-page.js	cart.html	Відображення кошика
login.js	login.html	Вхід
register.js	register.html	Реєстрація
checkout.js	checkout.html	Оформлення замовлення
payment.js	payment.html	Тестова оплата (sandbox)
profile-page.js	profile.html	Замовлення та ключі
wishlist-page.js	wishlist.html	Список бажань
admin.js	admin.html	CRUD ігор (для ADMIN)
orders.js	orders.html	Перегляд замовлень (для ADMIN)

Модуль api.js це головний файл клієнтської частини.

У ньому:

– apiRequest() — відправляє запити на сервер;

– authAPI, gamesAPI, ordersAPI, paymentsAPI, wishlistAPI — готові методи для різних операцій;

– після входу в localStorage зберігаються authToken і currentUser;

– escapeHtml() — захист від некоректного тексту на сторінці.

Приклад: gamesAPI.listGames() завантажує каталог, ordersAPI.createOrder() створює замовлення, paymentsAPI.sandboxPay() проводить тестову оплату.

Авторизація та кошик login.js — відправляє логін і пароль на /api/auth/login. Якщо все добре, дані зберігаються в localStorage, і користувач може оформлювати замовленн:

– cart.js — кошик зберігається в браузері (localStorage, ключ keygames_cart). Додати гру можна без входу. При оформленні замовлення дані з кошика йдуть на сервер.

– layout.js — показує різне меню для гостя, USER і ADMIN (пункти «Адмін» і «Замовлення» лише для адміністратора).

Основні сторінки

– games.html + games.js — каталог з фільтрами (назва, жанр, ціна), модальне вікно з деталями гри;

– checkout.html + checkout.js — форма оформлення, POST /api/orders;

– payment.html + payment.js — тестова картка 4242 4242 4242 4242, POST /api/payments/sandbox;

– profile.html + profile-page.js — історія покупок і ліцензійні ключі;

– admin.html + admin.js — додавання, редагування і видалення ігор (лише ADMIN).

2.4.2 Написання серверної частини

Мова: Java 17.

Фреймворк: Spring Boot 3.3.1[1] (Spring Web, Spring Data JPA, Spring Security[2]).

База даних: PostgreSQL[3].

Середовище: IntelliJ IDEA, Maven.

Точка входу: GameStoreApplication.java.

Серверна частина — це backend вебсервісу «KeyGames». Вона приймає HTTP-запити від JavaScript-клієнта, виконує бізнес-логіку (реєстрація, замовлення, оплата, ключі) і зберігає дані в PostgreSQL. Відповіді повертаються у форматі JSON.

– Архітектура backend Backend побудовано за шаровою архітектурою:

– Controller — приймає HTTP-запит, повертає JSON;

– Service — бізнес-логіка (створення замовлення, оплата, видача ключів);

– Repository — робота з БД через Spring Data JPA;

– Entity[1] (model) — Java-класи, що відповідають таблицям PostgreSQL (підрозділ 2.3).

Схема взаємодії компонентів — на плакаті 2026.КВР.122.421.19.00.00 СС. UML-діаграма класів — на плакаті 2026.КВР.122.421.19.00.00 ДК.

Основні Entity-класи нгаведено в таблиці. 2.13.

Таблиця 2.13 – Основні Entity-класи серверної частини

Клас	Таблиця БД	Призначення
User	Users	Обліковий запис, роль USER/ADMIN
Game	Games	Гра в каталозі
Order	Orders	Замовлення
Orderitem	Order_items	Позиція замовлення
Gamekey	Game_keys	Ліцензійний ключ
Wishlistitem	Wishlist_items	Список бажань
Review	Reviews	Відгук і рейтинг

Сервісний шар Бізнес-логіка винесена в таблиці 2.14.

Таблиця 2.14 – Сервіси backend

Сервіс	Відповідальність
UserService	Реєстрація, пошук користувача
GameService	CRUD ігор, фільтри, популярні
OrderService	Створення/видалення замовлень
PaymentService	Sandbox-оплата, зміна статусу
GameKeyService	Генерація та прив'язка ключів
WishlistService	Список бажань
ReviewService	Відгуки, перевірка покупки
ImageStorageService	Збереження обкладинок у uploads/games/

REST API[6] Контролери обслуговують HTTP-запити. Основні ендпоінти — у таблиці. 2.15.

Таблиця 2.15 – Основні REST-ендпоінти

Метод	Endpoint	Опис	Доступ
Post	/api/auth/register	Реєстрація	Публічний
Post	/api/auth/login	Вхід,Basic auth	Публічний
Get	/games	Каталог з фільтрами	Публічний
Get	/games/popular	Топ за рейнтингом	Публічний
Get	/game{id}	Деталі гри	Публічний
Post	/orders	Нове замовлення	User
Get	/ orders	Список замовлень	User
Post	/ api/payments/sandbox	Sandbox-оплата	User
Get	/ api/wishlist/game	Список бажань	User
Post	/ api/wishlist/game/{gameId}	Додати у wishlist	User
Delete	/ api/wishlist/game/{gameId}	Видалити з wishlist	User
Post/Put/Delete	/games/games/{id}	CRUD каталогу	Admin

Покупка гри на сервері проходить так:

1. OrderService.createOrder() — створює замовлення зі статусом pending_payment;

2. PaymentService.processSandboxPayment() — перевіряє тестову картку і CVV;

3. при успіху: статус → paid, зменшується stock, GameKeyService прив’язує ключі до позицій; к

4. лючі повертаються в JSON і відображаються в профілі користувача.

При видаленні оплаченого замовлення адміном відновлюється залишок і ключі звільняються.

Клас SecurityConfig налаштовує доступ:

– публічні: GET /api/games/**, /api/auth/**, статичні HTML/CSS/JS;
– для USER: /api/orders, /api/wishlist, /api/payments;
– для ADMIN: POST/PUT/DELETE /api/games.

Паролі зберігаються як BCrypt-хеші. Авторизація — HTTP Basic Auth[11]. Помилки обробляє RestExceptionHandler (коди 400, 403, 404 у JSON).

Пакет/каталог	призначення
controller/	AuthController, GameController, OrderController, PaymentController, WishlistController, ReviewController
service/	Бізнес-логіка замовлень, оплати, ключів, відгуків
repository/	JPA-доступ до таблиць PostgreSQL
model/	Entity - класи
dto/	Об'єкти запитів і відповідей
security/	SecurityConfig, UserDetailsServiceImpl
config/	WebConfig, DataInitializer, DatabaseSchemaUpdater

Конфігурація порту і підключення до БД — у файлі application.yml. Локально backend запускається командою mvn spring-boot:run на порту 8080. У production — на Railway[10]з PostgreSQL[3].

У результаті серверна частина забезпечує повний функціонал вебсервісу: API для клієнта, збереження даних і контроль доступу. Фрагменти Java-коду наведено в

Додатках до записки.

2.5 Тестування вебсервісу

Під час завершення активної фази програмування інтернет-магазину «KeyGames» обов'язковим етапом стає проведення всебічного тестування для підтвердження стабільності всіх модулів web-сайту. Це гарантує високу якість продукту. Перевірка здійснюється з метою верифікації функціональних можливостей, оцінки швидкодії інтерфейсу та контролю безпеки передачі даних між браузером і сервером.

Для отримання об'єктивних результатів реалізовано декілька стратегій перевірки:

– функціональне тестування для контролю роботи каталогу, кошика, оплати та видачі ключів;

– інтеграційне тестування для перевірки обміну JSON-даними між HTML/JS frontend та Spring Boot backend;

– негативне тестування валідації (CVV, ролі, відгуки без покупки);

– тестування безпеки для підтвердження стійкості HTTP Basic Auth[11] та захисту адмін-маршрутів.

На початковому етапі виконано перевірку модуля автентифікації через web-сторінки login.html та register.html. Після входу користувач отримує доступ до кошика, профілю та wishlist. Далі тестується каталог із фільтрами — кожен вибір ініціює запит до GET /games і має повертати актуальний список ігор.

Особлива увага приділяється процесу додавання гри до кошика, оформлення замовлення та sandbox-оплати. Тестувальник перевіряє зміну статусу на paid та наявність ключів у профілі. Для адміністративної частини сайту проведено серію тестів CRUD ігор та перегляду замовлень під обліковим записом admin/admin123. На рисунку 2.16 відображена перевірка формату CVV,також відмова у створенні відгуку зображено на рисунку 2.17.

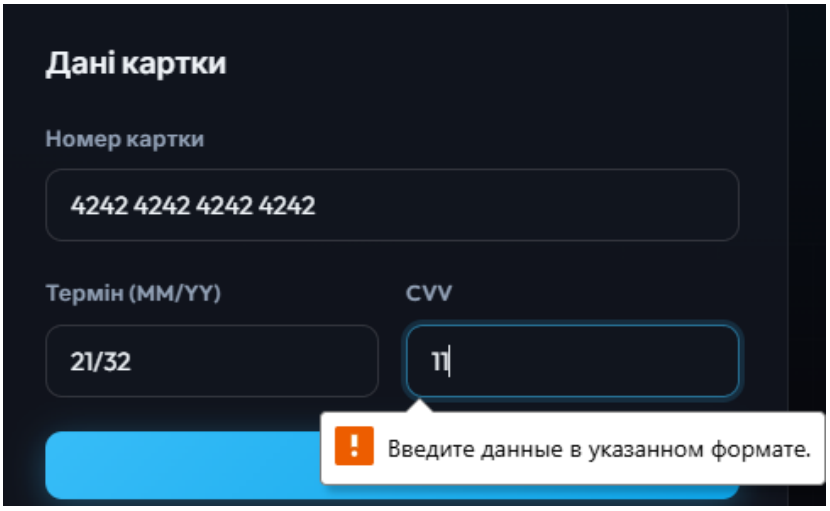


Рисунок 2.16 – Перевірка формату CVV на стороні клієнта

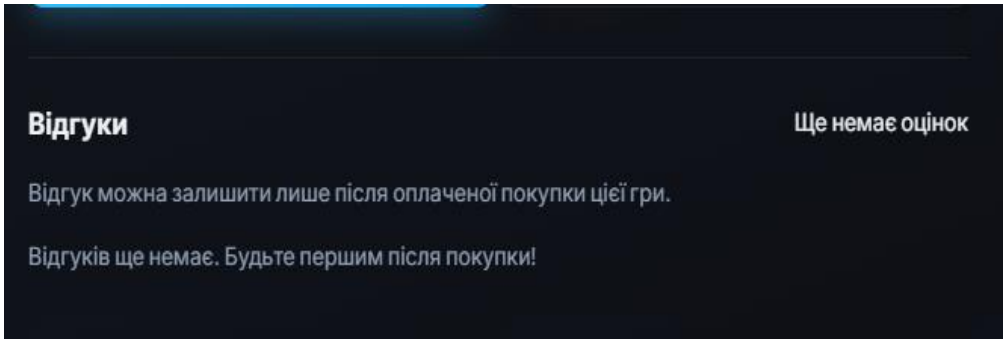


Рисунок 2.17 – Відмова у створенні відгуку без підтвердженої покупки

За допомогою інструменту Postman додатково виконано перевірку REST API: імітація запитів з некоректним CVV, доступ до /orders без авторизації (401), доступ до POST /games під роллю USER (403).

Таблиця 2.16 – Фрагмент контрольного списку результатів тестування web-сайту

№	Перевірка	Очікуваний результат	Статус
1	Реєстрація ного user	201, роль USER	Ок
2	Логін admin/admin123	Доступ до admin.html	Ок
3	Фільтр каталогу за жанром	Відфільрований список	Ок
4	CVV з 2 цифрами на	HTML5-валідація,	ок

	payment.html	форма не відправляється	
5	Оплата тестовою карткою	статус paid, ключі в order	Ок
6	Відгук без покупки	403	Ок
7	Видалення гри (admin	204	ок

У результаті проведених випробувань встановлено, що web-сайт «KeyGames» функціонує стабільно та повністю відповідає вимогам технічного завдання. Взаємодія між web-сторінками та серверною логікою забезпечує цілісність даних та комфортну роботу користувача. Проєкт готовий до демонстрації на захисті кваліфікаційної роботи.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з розгортання вебсервісу

Для забезпечення роботи розробленого інтернет-магазину цифрових ключів «KeyGames» у навчальному або демонстраційному середовищі виконується процедура локального розміщення програмних компонентів на персональному комп'ютері. Це робиться покроково. Шляхом використання вбудованого сервера Spring Boot[1] та реляційної бази даних PostgreSQL[3] вдається побудувати відмовостійку конфігурацію «клієнт у браузері — сервер на порту 8080 — БД game_store» без окремого хостингу. Процес запуску вимагає чіткого налаштування підключення до БД та перевірки наявності JDK 17. Результат має бути стабільним.

Мінімальні вимоги до апаратного та програмного забезпечення робочого місця наведено в таблиці 3.1.

Таблиця 3.1 – Мінімальні вимоги до робочого місця

Компонент	Вимога
Ос	Windows 10/11 (64-bit) або Linux
Оперативна пам’ять	8 ГБ (рекомендовано 16 ГБ)
Вільний місце на дискові	2 ГБ
JDK	17 або новіше
PostgreSQL	14 або новіше
Maven	3.8+ (або вбудований wrapper)
Браузер	Chrome, Firefox, Edge (актуальна версія)

Під час підготовки до розміщення серверної частини web-сайту «KeyGames» необхідно виконати наступну послідовність дій:

– За допомогою інсталятора з офіційного сайту postgresql.org[3] встановити

PostgreSQL[3] на локальний комп'ютер і запам'ятати пароль суперкористувача postgres.

– Шляхом виконання скрипта create_database.sql створити користувача KeyGames, базу даних game_store та видати необхідні права на схему public.

– Через утиліту psql або pgAdmin переконатися, що підключення KeyGames / game_store успішне. Успішне підключення до бази даних game_store показано на рисунку 3.1.

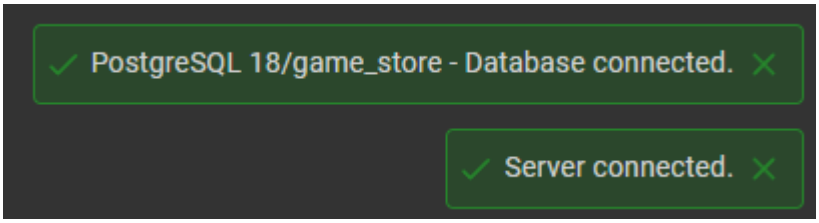


Рисунок 3.1 – Підключення до бази даних PostgreSQL проєкту KeyGames

Важливим етапом виступає конфігурація файлу src/main/resources/application.yml, де прописуються параметри доступу до PostgreSQL. Безпека понад усе. Для роботи сервера необхідно перевірити:

– spring.datasource.url — jdbc:postgresql://localhost:5432/game_store;

– spring.datasource.username — KeyGames;

– spring.datasource.password — пароль, заданий при створенні користувача (у навчальному варіанті: 1234567890);

– server.port — 8080;

– app.upload.dir — uploads (каталог обкладинок ігор).

Після збереження налаштувань виконується збірка Maven-проєкту. Це дозволяє перевірити коректність залежностей Spring Boot 3.3.1[1].

Для розгортання клієнтської та серверної частини в одному процесі використовується команда: mvn spring-boot:run.

Альтернативно — запуск класу GameStoreApplication з IntelliJ IDEA. Побудова виконується швидко. Послідовність кроків для запуску включає:

– Відкрити каталог проєкту у середовищі розробки або терміналі.

– Переконатися, що PostgreSQL запущений як служба ОС.

– Виконати mvn spring-boot:run і дочекатися рядка Started GameStoreApplication у консолі.

У браузері відкрити адресу http://localhost:8080/.

Успішний запуск серверної частини Spring Boot ілюструє рисунок 3.2.

Рисунок 3.2 – Успішний запуск серверної частини Spring Boot

Відкриття головної сторінки після запуску сервера наведено на рис. 3.3.

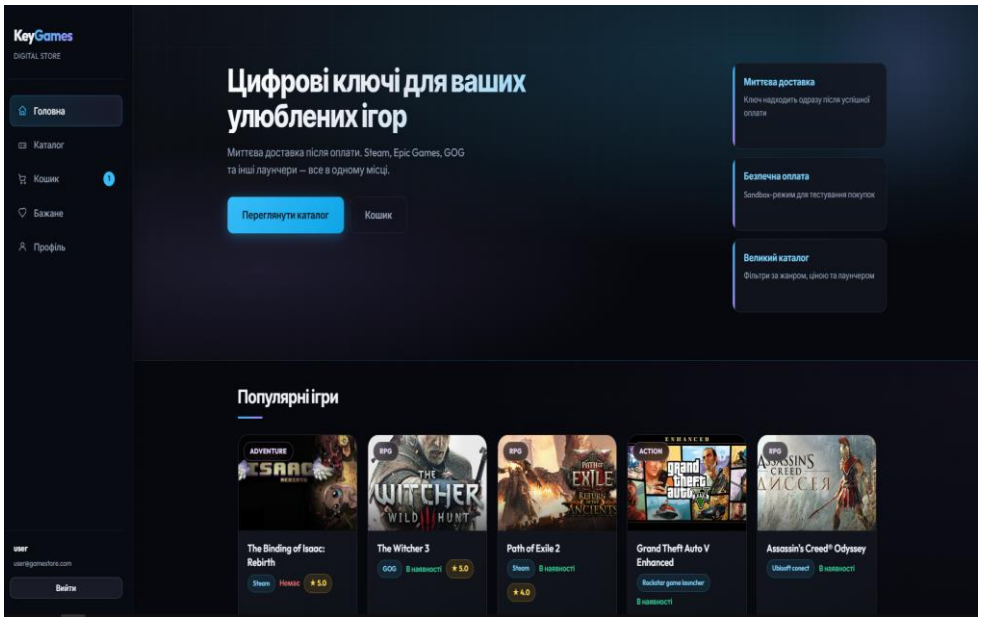


Рисунок 3.3 – Головна сторінка web-сайту KeyGames у браузері

Після завершення всіх етапів розміщення проводиться фінальна перевірка інтеграції, яка включає тестування авторизації та відкриття каталогу на локальному сервері. Усі посилання мають працювати. При першому запуску компонент DataInitializer автоматично створює тестові облікові записи, які представлено у таблиці 3.2.

Таблиця 3.2 – Тестові користувачі

Логін	пароль	роль
user	user123	USER
admin	admin123	ADMIN

Завдяки локальному розгортанню web-сайт «KeyGames» стає доступним для демонстрації на захисті кваліфікаційної роботи в режимі реального часу, що повністю відповідає цілям навчального проєкту.

3.2 Інструкція з обслуговування та наповнення вебсервісу

Для підтримки актуальності каталогу ігор та стабільної роботи інтернет-магазину «KeyGames» передбачено спеціалізовані інструменти керування контентом через адміністративну панель admin.html. Це спрощує роботу оператора магазину. Шляхом використання інтуїтивно зрозумілого інтерфейсу адміністратор може оперативно додавати нові ігри, змінювати ціни, завантажувати обкладинки та контролювати залишок stock. Процес наповнення каталогу є циклічним. Кожна операція супроводжується валідацією даних на сервері Spring Boot для запобігання появі некоректної інформації на web-сторінках. Результат відображається миттєво.

Під час підготовки до додавання нової позиції у каталог необхідно виконати наступну покрокову процедуру:

1. Авторизація — вхід під обліковим записом admin / admin123 через сторінку login.html.
2. Перехід до панелі — у бічному меню (layout.js) обрати посилання «Адмін-панель» (admin.html).

За допомогою форми входу, зображеної на рисунку 3.4, вводиться логін та пароль адміністратора. Після успішної автентифікації у навігаційному меню з'являються посилання на admin.html та orders.html.

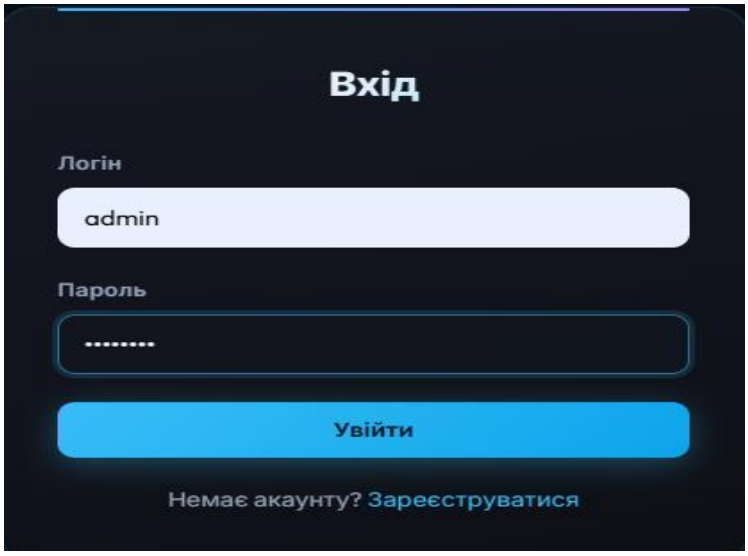


Рисунок 3.4 – Вікно авторизації адміністратора web-сайту KeyGames

3. Перегляд каталогу — на сторінці admin.html відображається таблиця існуючих ігор із можливістю редагування та видалення.

Через інтерактивну таблицю, зображену на рисунку 3.5, адміністратор може оцінити поточний асортимент та переконатися у відсутності дублікатів перед створенням нової картки гри.

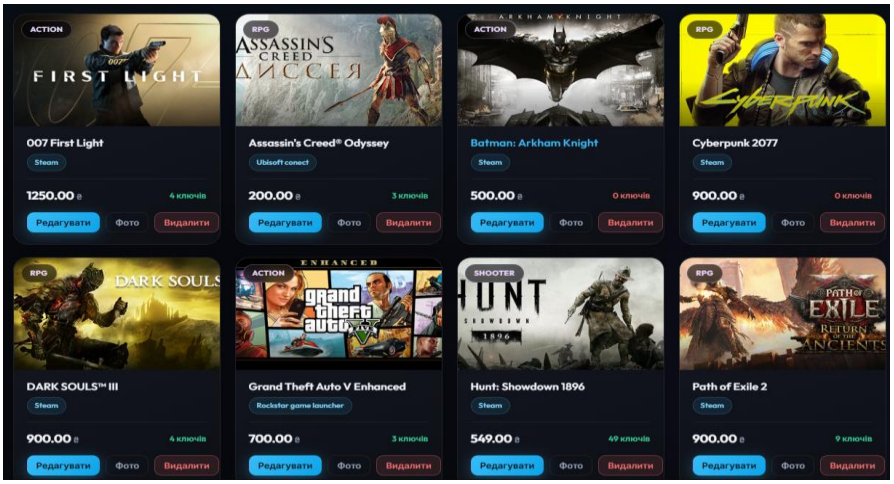


Рисунок 3.5 – Інтерфейс таблиці ігор у адміністративній панелі

4. Додавання гри — заповнення форми: назва, опис, жанр, платформа (Steam, Epic…), ціна, залишок stock.

У відповідні поля вносяться текстові дані та числові параметри. Форма має

вбудовану перевірку обов'язкових полів на стороні клієнта та сервера.

5. Завантаження обкладинки — через <input type="file"> файл передається на сервер методом POST /games/{id}/image у каталог uploads/games/.

6. Збереження — натискання «Зберегти» викликає POST або PUT /games; дані фіксуються у таблиці games PostgreSQL. Форма створення нової позиції в каталозі показана на рисунку 3.6.

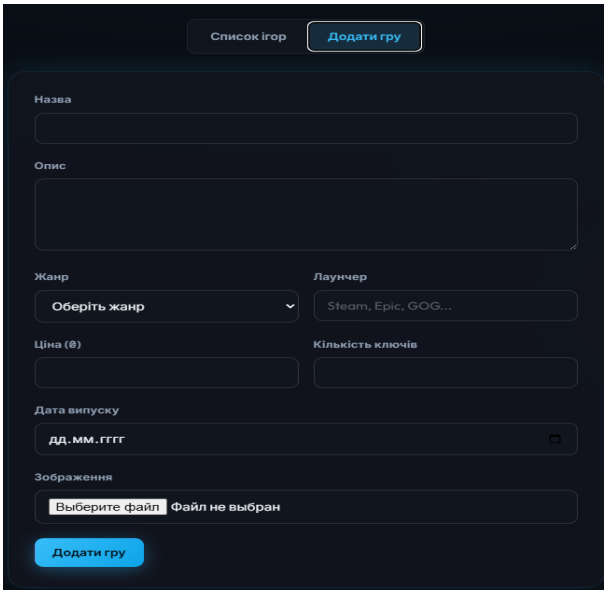


Рисунок 3.6 – Форма створення нової позиції в каталозі ігор

Завантаження обкладинки гри наведено на рисунку 3.7.

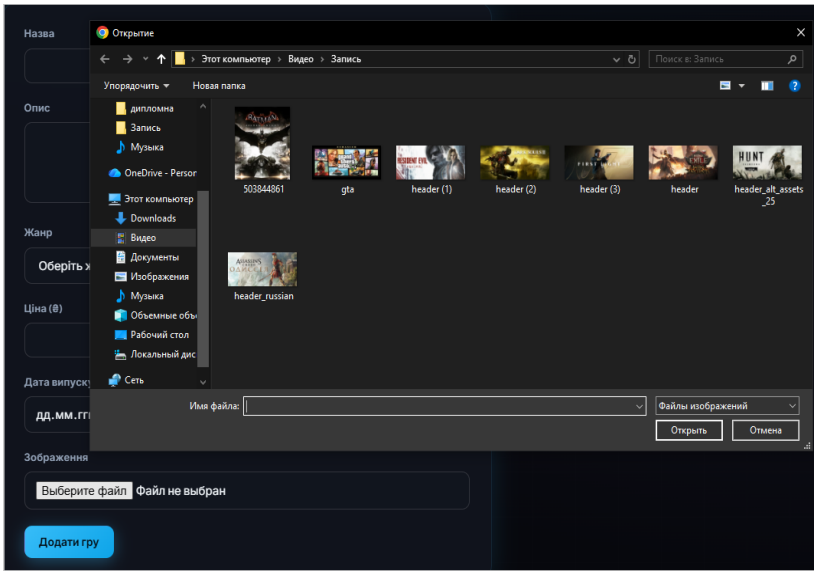


Рисунок 3.7 – Модуль завантаження обкладинки гри

Шляхом виконання контрольного сценарію перевіряється повний цикл покупки на вебсайті:

– Зареєструвати нового користувача або увійти як user / user123.

– Відкрити каталог, додати гру в кошик (cart.js, localStorage).

– Оформити замовлення на checkout.html — статус pending_payment.

– На payment.html провести оплату: картка 4242 4242 4242 4242, термін 12/30, CVV 123 (3 цифри).

– У profile.html перевірити статус paid та наявність ліцензійних ключів.

– Залишити відгук у модальному вікні гри.

– Увійти як admin, переглянути orders.html.

Результати контрольних перевірок наведено в таблиці 2.16, Подальше обслуговування системи полягає у регулярному моніторингу логів Spring Boot та своєчасному оновленні залишків stock і ключів через описаний вище інтерфейс адміністратора.

3.3 Інструкція з популяризації та підтримки вебсервісу

Після розміщення інтернет-магазину «KeyGames» у мережі Інтернет вебсервіс доступний за адресою https://key-games.vercel.app. Frontend обслуговується платформою Vercel[9], backend (Spring Boot) і база даних PostgreSQL[3] — на Railway[10]. Запити від браузера до API маршрутизуються через проксі /api/* у файлі vercel.json, що забезпечує коректну роботу без помилок CORS.

У межах стратегії залучення відвідувачів доцільно оптимізувати HTML-сторінки під пошукові системи: семантичні теги HTML5[4] (<title>, <meta name="description">, заголовки <h1>–<h3>), атрибут alt у зображеннях обкладинок, файл robots.txt (обмеження індексації login.html та admin.html). Для навчального проєкту достатньо базової SEO-розмітки; у production можна підключити Google Search Console та sitemap.xml.

Для просування бренду «KeyGames» можна використовувати соціальні мережі (Instagram, Telegram, Discord) з посиланнями на https://key-

games.vercel.app/games.html та UTM-мітками для обліку переходів.

Щоденна робота покупця (роль USER):

– перегляд каталогу на games.html без обов'язкової реєстрації;

– реєстрація та вхід через login.html;

– оформлення замовлення на checkout.html, sandbox-оплату на payment.html, отримання ключів у profile.html;

– wishlist і відгуки після авторизації.

Кошик зберігається в localStorage браузера; wishlist і замовлення — на сервері в PostgreSQL[3].

Робота адміністратора (роль ADMIN) — через admin.html та orders.html: CRUD ігор, завантаження обкладинок, перегляд замовлень, контроль stock і ключів у game_keys.

Технічна підтримка в хмарному середовищі:

– моніторинг деплоїв у панелях Vercel Dashboard та Railway Dashboard;

– резервне копіювання БД PostgreSQL[3] (pg_dump або вбудовані засоби Railway[10]);

– оновлення змінних середовища (URL БД, паролі) без публікації секретів у репозиторії;

– після змін у Java-коді — redeploy backend на Railway; frontend на Vercel[9]оновлюється автоматично при push у Git.

Для подальшого розвитку передбачено інтеграцію реального платіжного шлюзу (LiqPay, Stripe), OAuth2/JWT замість Basic Auth, email-розсилку ключів — без повної перебудови архітектури Spring Boot.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини даного дипломного проєкту є проведення економічних розрахунків, спрямованих на визначення економічної ефективності розробки KeyGames, прийняття рішення про подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки.

Об'єкт розробки є KeyGames

Розрахунок вартості розробки виконується в декілька етапів:

– описати технологічний процес розробки із зазначенням трудомісткості кожної операції;

– визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;

– обчислити витрати на електроенергію;

– нарахувати суму амортизаційних відрахувань;

– визначити суму накладних витрат;

– скласти кошторис та визначити собівартість робіт;

– розрахувати ціну робіт;

– визначити економічну ефективність та термін окупності.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

В цьому підрозділі розглянемо основні етапи технологічного процесу для розробки KeyGames. Для визначення загальної тривалості проведення робіт доцільно дані витрат часу по окремих операціях технологічного процесу звести у таблицю.

Таблиця 4.1 – Середній час виконання робіт по обслуговуванню та стадії (операції) технологічного процесу

№	Назва операції (стадії)	Виконавець	Час, год
1	Планування та аналіз	Кер. проєкту (Pm)	10
		Інженер (l1)	8
2	Розробка технічного завдання	Кер. проєкту (Pm)	8
		Інженер (l1)	5
3	Дизайн інтерфейсу	Інженер (l1)	18
		Інженер (l2)	20
4	Розробка функціоналу	Інженер (l2)	45
5	Тестування та відладка	Тестувальник	10
6	Документування	Інженер (l1)	4
7	Розгортання та підтримка	Інженер (l1)	25
8	Управління проєктом	Кер. проєкту (Pm)	27
	Разом		180

Сумарний час виконання операцій технологічного процесу становить 180 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

У даному підрозділі проводиться аналіз і розрахунок витрат, пов'язаних з оплатою праці та відрахуваннями на соціальні заходи, що необхідні для розробки KeyGames.

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та діяльності підприємства.

Основна заробітна плата розраховується за формулою:

$$З_{осн.}=Т_с*К_г \quad (4.1)$$

де:$Т_с$ – тарифна ставка, грн. (приймаємо для керівника проєкту (Pm) – 500 грн./год, інженера (l2) – 300 грн./год, інженера (l1) – 150 грн./год, тестувальника – 120 грн./год);

$К_г$ – кількість відпрацьованих годин.

Отже, основна заробітна плата для:

– Керівника проєкту (Pm): $З_{осн}$ = 45 * 500 = 22 500.

– Інженера (l2): $З_{осн}$ = 65* 300 = 19 500

– Інженера (l1): $З_{осн}$ =60 * 150 = 9000

– Тестувальника: $З_{осн}$ = 10 * 120 = 1 200.

Сумарна основна заробітна плата становить:

$З_{осн}$ = 22 500 + 19 500 + 9000 + 1 200 = 52 200

Додаткова заробітна плата становить 10% від суми основної заробітної плати:

$$З_{дод.}=З_{осн.}* К_{допл.} \quad (4.2)$$

де $К_{допл}$ – коефіцієнт додаткових виплат.

Отже, додаткова заробітна плата за категоріями:

– Керівник проєкту: 3 дод = 22 500 * 0 , 1 = 2 250 грн.

– Інженер (l2): 3 дод = 19 500 * 0 , 1 = 1 950 грн.

– Інженер (l1): 3 дод = 9000 * 0 , 1 = 900 грн.

– Тестувальник: 3 дод = 1 200 * 0 , 1 = 120 грн.

Загальна додаткова заробітна плата:

3дод=2250+1950+900+120=5 220 грн.

Загальні витрати на оплату праці ($В_{о.п.}$) визначаються за формулою:

$$Во.п.=Зосн.+Здод. \quad (4.3)$$

$В_{о.п}$=52 200+5 220=57 420 грн.

Єдиний соціальний внесок (ЄСВ – 22%) визначається за формулою:

$$Вєсв=Во.п*0,22 \quad (4.4)$$

$В_{єсв}$=57 420*0,22=12 632рн.

Проведені розрахунки витрат на оплату праці наведено у таблиці 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	ЄСВ, грн.	Всього витрати на оплату праці, грн. 6 = 3+4+5
		Тарифна ставка, грн.	К-сть годин	Фактично нарах. зарплати, грн.			
1	Кер. проєкту (Pm)	500	45	22 500	2 250	5 445	30 195
2	Інженера (l2)	300	65	19 500	1 950	4 719	26 169
3	Інженера (l1)	150	60	9000	900	2 178	12 078
4	Тестувальник	120	10	1 200	120	290	1 610
Разом				52200	5 220	12632	70 052

Отже, загальні витрати на оплату праці становлять 70 052 грн.

4.3 Розрахунок витрат на електроенергію

Розрахуємо вартість електроенергії. Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_в=W*T*S \quad (4.5)$$

де W - необхідна потужність, кВт (комп'ютер – 0,82 кВт/год);

Т - кількість годин роботи обладнання (180 год);

S - вартість кіловат-години електроенергії (приймаємо 15,94 грн).

$З_{ек}$ =0,82*180*15,94=2353 грн.

Витрати на електроенергію становлять 2 353 грн.

4.4 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення.

Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Амортизація на них нараховується лише в випадку, якщо мінімально допустимі строки їх корисного використання – 2 роки. Для визначення амортизаційних відрахувань застосовуємо формулу:

$$А = \frac{Б_В * Н_А}{100\%} * T, \quad (4.6)$$

де: А – амортизаційні відрахування за звітний період, грн.;

$Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн. (вартість ПК – 55 000 грн);

$Н_А$ – норма амортизації, 0,04;

Т – кількість годин роботи обладнання (180 год)

$$А = \frac{55\ \ 000{,}00 * 0{,}04}{150} * 180 = \frac{2\ \ 200}{150} * 180 = 14\ \ 6667 * 180 = 2640грн$$

Сума амортизаційних відрахувань становить 2 640 грн.

4.5 Обчислення накладних витрат

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20–60% від суми основної та додаткової заробітної плати працівників.

$$Н_в = В_{о.п.} * 0,4 \qquad (4.7)$$

де: $Н_В$ – накладні витрати.

$$Н_В = 57420 * 0,4 = 22\ 968 \text{ грн.}$$

4.6 Складання кошторису витрат та визначення собівартості KeyGames

Для складення кошторису витрат та визначення собівартості, результати проведених вище розрахунків зведемо у таблиці 4.3.

Таблиця 4.3 – Кошторис витрат KeyGames

№	Зміст витрат	Сума, грн	%
1.	Витрати на оплату праці	70.052	71.3
2.	Витрати Електроенергія	2 353	2.4
3.	Амортизаційні відрахування	2 640	2.7
4.	Накладні витрати	22 968	23.4
5.	Собівартість	98 013	100

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$Св = Во.п + Вс.з + Зе + А + Нв \quad (4.8)$$

Св=70 052+2 353+2 640+22 968=98 013 грн.

Отже, собівартість дорівнює 98 013 грн.

4.7 Розрахунок ціни вебсервісу «KeyGames»

Розрахунок ціни науково-дослідної роботи включає в себе урахування різноманітних факторів, таких як рівень рентабельності, собівартість та податкова ставка.

Ціну робіт можна визначити за формулою:

$$Ц = Св * (1+Р_{рен}) * (1+ПДВ), \quad (4.9)$$

де: Св – собівартість; $Р_{рен}$ – рівень рентабельності (30%); ПДВ – ставка податку на додану вартість (20%).

Ц=98 013*(1+0,3)*(1+0,2) = 152 900

4.8 Визначення економічної ефективності і терміну окупності

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ) і термін окупності (Ток).

$$ЧТВ = -Св + \sum_{i=1}^{t} \frac{Г_П}{(1+i)^t}, \quad 4.10)$$

де: Св – собівартість розробки; $Г_П$ – грошовий потік за t-ий рік, і i – дисконтна ставка (10% = 0,1).

$$ЧТВ = -98\ \ 013 + \frac{54\ \ 887}{(1+0,1)^1} + \frac{54\ \ 887}{(1+0,1)^2} + \frac{54\ \ 887}{(1+0,1)^3} = 38\ \ 513$$

Якщо ЧТВ ≥ 0, то проект може бути рекомендований до впровадження.

$$T_{ОК} = T_{ПВ} + \frac{Н_В}{Г_{ПР}} \quad (4.11)$$

де: $Т_{ПВ}$ – період до повного відшкодування витрат, років; $Н_В$ – невідшкодовані витрати на початок року, грн.; $Г_{ПР}$ – грошовий потік на початок року, грн.

$$Т_{ок} = 2 + \frac{2754}{54\ \ 887} = 2{,}1 \ \ р.$$

Всі дані внесемо в зведену таблицю 4.4.

Таблиця 4.4 – Техніко-економічні показники розробки вебсервісу «KeyGames»

№ п/п	Показник	Значення
1.	Собівартість, грн.	98 013
2.	Плановий прибуток або грошовий потік, грн.	54 887
3.	Ціна, грн.	152 900
4.	Чиста теперішня вартість, грн.	38 513
5.	Термін окупності, рік	2,1

Прибутковість проєкту та термін окупності свідчать про його фінансову ефективність та здатність повернути капітальні вкладення протягом 2,1 року. Отже, на основі отриманих показників можна зробити висновок, що розробка вебсервісу «KeyGames» є доцільною з економічної точки зору.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

5.1 Навчання з питань охорони праці

Організація навчання з питань охорони праці регламентується Законом України «Про охорону праці», Порядком проведення навчання і перевірки знань з питань охорони праці (Наказ Мінсоцполітики № 2464 від 21.11.2014) та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин».

Мета навчання — формування у виконавця кваліфікаційної роботи стабільних навичок безпечної поведінки на робочому місці програміста: дотримання режиму праці за ПК, правил електробезпеки при підключенні монітора та системного блока, профілактики зорового та статичного навантаження, а також базових знань з пожежної безпеки в приміщенні лабораторії або кабінету.

Види навчання, які проходить студент групи КН-421 Шевяков Антон.

У процесі підготовки до виконання кваліфікаційної роботи студент проходить такі види навчання з питань охорони праці:

– навчання в ЗВО (дисципліна «Безпека життєдіяльності та охорона праці») загальні принципи ОП, класифікація небезпечних факторів, електробезпека, перша допомога; форма — лекції та практичні заняття протягом навчання;

– вступний інструктаж ознайомлення з правилами поведінки в навчальному корпусі, порядком евакуації, розташуванням вогнегасника та аптечки; проводиться до початку роботи в комп’ютерній лабораторії;

– первинний інструктаж на робочому місці безпечна робота з ПК, монітором і периферією; заборона самовільного ремонту кабелів; режим перерв; тривалість не менше 2 год;

– повторний інструктаж оновлення знань після канікул або зміни умов у лабораторії; проводиться щорічно або за наказом;

– цільовий інструктаж робота з мережею 220 В, підключення ДБЖ, безпечне використання хмарних сервісів; проводиться перед розгортанням на Railway

Програма навчання для робочого місця розробника вебзастосунку включає:

1. Загальні вимоги — не залишати увімкненим обладнання без нагляду;
2. не допускати потрапляння рідини на клавіатуру, системний блок або блок живлення; не перевантажувати мережеві подовжувачі.
3. Ергономіка — правильне положення на кріслі, відстань до монітора 50–70 см, перерви 10–15 хв через кожну годину роботи за ПК (ДСанПіН 3.3.2-007-98).
4. Електробезпека — використання лише справних кабелів і розеток із заземленням;
5. заборона дотику до відкритих контактів блоку живлення ПК під напругою.
6. Пожежна безпека — заборона паління; знання місця розташування вогнегасника ОУ-2; дії при задимленні або перегріві обладнання.
7. Інформаційна безпека — не публікувати паролі PostgreSQL[3] та ключі Railway[10] у репозиторії; використовувати HTTPS для production-демонстрації.

Перевірка знань з питань охорони праці проводиться відповідальним за навчання у навчальному закладі (комісія або викладач дисципліни БЖД) у формі усного опитування або тестування. Результати фіксуються в журналі інструктажів. До самостійної роботи над дипломним проєктом «KeyGames» студент допускається після успішного проходження вступного та первинного інструктажів.

5.2 Механічна дія струму

Під час розробки та тестування вебзастосунку «KeyGames» основним джерелом небезпеки ураження електричним струмом є мережа змінного струму 220 В, 50 Гц, якою живляться персональний комп’ютер, монітор, периферія та, за наявності, джерело безперебійного живлення (ДБЖ). Приміщення лабораторії або кабінету за ступенем електронебезпеки належить до категорії без підвищеної небезпеки (НПАОП 0.00-1.28-10).

Механічна дія струму — це фізіологічний ефект, при якому проходження електричного струму через м’язову тканину викликає судоми, стиснення грудної

клітки та nevoluntary скорочення м’язів кистей. У результаті людина може не здатна самостійно розімкнути захоплення пошкодженого кабеля або металевого корпусу обладнання, що знаходиться під напругою. Тривала судомна дія струму призводить до механічних ушкоджень тканин, переломів кісток (у разі падіння) та асфіксії через параліч дихальних м’язів.

Типові травмонебезпечні місця на робочому місці розробника:

– пошкоджена ізоляція кабеля живлення монітора або системного блока;

– відкриті контакти розетки або подовжувача при відсутності заземлення;

– корпус ПК або корпус монітора під напругою дотику при обриві проводу PE (захисного заземлення);

– дотик до внутрішніх частин ПК (блок живлення) під час некваліфікованого ремонту.

Робоче місце розробника «KeyGames» живиться від однофазної мережі змінного струму напругою 220 В, частотою 50 Гц, з заземленою нейтраллю (режим TN-C-S / TN-S). Персональний комп’ютер підключається через триполюсну розетку із заземлювальним контактом.

Граничні значення сили струму та наслідки механічної дії наведено в таблиці 5.1

Таблиця 5.1 – Граничні значення сили струму та наслідки механічної дії

Сила струму, мА	Наслідок
0,6–1,5	Порог відчуття
5–7	Болючі судоми
10–15	Стійкі судоми; неможливість самостійно відпустити провідник
20–30	Порушення дихання
50–100	Фібриляція серця, зупинка дихання

Отримане значення перевищує небезпечний поріг (50–100 мА) у 2–4 рази і

може призвести до зупинки серця та дихання.

Висновок: умови експлуатації ПК без належного заземлення та цілісності кабелів є неприпустимими.

Захисні заходи на робочому місці розробника «KeyGames»:

– використання розеток із справним заземленням (опір заземлювача — не більше 4 Ом);

– застосування мережевих фільтрів із захистом від перенапруг;

– заборона роботи з пошкодженими кабелями та самовільного відкривання блоку живлення ПК;

– встановлення ДБЖ для безпечного завершення роботи PostgreSQL при відключенні електроенергії;

– роздільне прокладення кабелів живлення та сигнальних кабелів;

– наявність аптечки та знання правил першої допомоги при ураженні струмом (виклик «103», відключення джерела напруги, перевірка дихання та пульсу).

Дотримання зазначених заходів забезпечує прийнятний рівень електробезпеки під час виконання кваліфікаційної роботи до зупинки серця та дихання.

ВИСНОВКИ

У кваліфікаційній роботі виконано розробку вебсервісу дистрибуції комп'ютерних ігор «KeyGames».

За результатами першого розділу проведено аналіз платформ Steam, Epic Games Store, GOG та itch.io; сформульовано технічне завдання з вимогами до функціоналу, надійності та умов експлуатації.

У другому розділі спроєктовано клієнт-серверну архітектуру на Spring Boot 3.3.1[1] і PostgreSQL[3]; описано варіанти використання, інтерфейс, систему класів, структуру бази даних (7 таблиць) та REST API[6]. Реалізовано каталог, кошик, оформлення замовлень, sandbox-оплату з видачею ключів, wishlist, відгуки та адміністративну панель. Тестування підтвердило працездатність основних сценаріїв.

Третій розділ містить інструкції з локального розміщення та запуску вебсервісу, обслуговування та наповнення каталогу, популяризації та технічної підтримки в production-середовищі.

У четвертому розділ виконано техніко-економічні розрахунки розробки «KeyGames»: визначено трудомісткість технологічного процесу (180 год, табл. 4.1), собівартість, ціну робіт та показники економічної ефективності.

У п’ятому розділі проаналізовано небезпечні та шкідливі фактори при локальній розробці та демонстрації KeyGames; визначено вимоги до робочого місця, електро- та пожежної безпеки та екологічні заходи. Умови праці визнано безпечними.

Напрями подальшого вдосконалення: інтеграція реального платіжного шлюзу (LiqPay, Stripe); OAuth2/JWT замість Basic Auth; Docker-контейнеризація; email-розсилка ключів; мобільна адаптація PWA.

ПЕРЕЛІК ПОСИЛАНЬ

1) Spring Boot Reference Documentation : вебсайт. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата звернення: 14.05.2024).
2) Spring Security Reference : вебсайт. URL: https://docs.spring.io/spring-security/reference/ (дата звернення: 14.05.2024).
3) PostgreSQL Documentation : вебсайт. URL: https://www.postgresql.org/docs/ (дата звернення: 15.05.2024).
4) MDN Web Docs — HTML : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 15.05.2024).
5) MDN Web Docs — JavaScript : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 16.05.2024).
6) Fielding R. Architectural Styles and the Design of Network-based Software Architectures : дис. … — 2000. — REST.
7) Bloch J. Effective Java. 3rd ed. Boston : Addison-Wesley, 2018. 416 p.
8) Мартін Р. Чистий код : створення, аналіз і рефакторинг. Львів : Фабула, 2019. 448 с.
9) Vercel Documentation : вебсайт. URL: https://vercel.com/docs (дата звернення: 20.05.2024).
10) Railway Documentation : вебсайт. URL: https://docs.railway.app/ (дата звернення: 20.05.2024).
11) HTTP/1.1 Semantics and Content : вебсайт. URL: https://datatracker.ietf.org/doc/html/rfc9110 (дата звернення: 21.05.2024).
12) Веб-технології та розробка веб-застосунків / О.В. Шевченко, М.А. Іванов, С.П. Кравченко та ін. Київ : Політехніка, 2023. 412 с.
13) ДСанПіН 3.3.2-007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин.
14) НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації

електронно-обчислювальних машин.

15) Наказ Мінсоцполітики № 2464 від 21.11.2014. Порядок проведення навчання і перевірки знань з питань охорони праці.

16) ДСН 3.3.6.037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку.

17) НАПБ А.01.001-2014. Правила пожежної безпеки в Україні.

18) Закон України «Про відходи електричного та електронного обладнання» від 05.03.2022 № 2112-IX.

ДОДАТКИ

Додаток А. Лістинг програмного коду головної HTML-сторінки «KeyGames»

```
<!DOCTYPE html>
<html lang="uk">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>KeyGames — ігровий магазин</title>
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&dis
play=swap" rel="stylesheet">
<link rel="stylesheet" href="/css/style.css?v=33">
</head>
<body class="app-body">
<aside id="sidebar" class="sidebar"></aside>
<div class="main-wrapper">
<main>
<section class="epic-hero">
<div class="epic-hero-layout">
<div class="epic-hero-inner">
<h1>Отримуйте ключі для улюблених ігор миттєво</h1> <p>Купуйте цифрові ключі
миттєво. Steam, Epic Games, GOG та інші лаунчери — все в одному місці.</p>
<div class="hero-actions">
<a href="/games.html" class="btn btn-large btn-primary">Переглянути
каталог</a>
<a href="/cart.html" id="hero-cart-link" class="btn btn-large btn-
ghost">Кошик</a>
</div>
```

```
</div>
<aside class="hero-features" aria-label="Переваги сервісу">
<div class="feature-pill">
<h3>Миттєва доставка</h3>
<p>Ключі з'являються одразу після оплати</p>
</div>
<div class="feature-pill">
<h3>Безпечна оплата</h3>
<p>Sandbox-оплата для тестування покупок</p>
</div>
<div class="feature-pill">
<h3>Профіль клієнта</h3>
<p>Історія та ключі, статуси та параметри</p>
</div>
</aside>
</div>
</section>
<section class="page-section">
<h2 class="section-title">Популярні ігри</h2>
<div id="games-grid" class="games-grid home-games-grid"><div
class="loading">Завантаження...</div></div>
</section>
</main>
<footer class="app-footer"><p>&copy; 2025 KeyGames. Всі права
захищені.</p></footer>
</div>
<div id="game-modal" class="modal">
<div class="modal-content">
<span class="close">&times;</span>
<div id="modal-body"></div>
</div>
</div>
<script src="/js/config.js?v=37"></script>
<script src="/js/api.js?v=37"></script>
```

```
    <script src="/js/cart.js"></script>
<script src="/js/layout.js"></script>
<script src="/js/auth.js"></script>
<script src="/js/reviews.js?v=1"></script>
<script src="/js/main.js?v=22"></script>
</body>
</html>
```

Додаток Б. Лістинг програмного коду клієнтського модуля api.js

```
const RAILWAY_API = 'https://keygames-production.up.railway.app';
function resolveApiBase() {
if (typeof window === 'undefined') return '';
const host = window.location.hostname;
if (host === 'localhost' || host === '127.0.0.1' ||
host.endsWith('.railway.app') || host.endsWith('.vercel.app')) {
return '';
}
const configured = window.KEYGAMES_API_URL;
if (configured && String(configured).trim()) {
return String(configured).replace(/\/$/, '');
}
return RAILWAY_API;
}
const API_BASE = resolveApiBase();
function resolveAssetUrl(url) {
if (!url) return '';
if (url.startsWith('http://') || url.startsWith('https://')) return url;
if (url.startsWith('/')) return API_BASE + url;
return url;
}
function getAuthToken() { return localStorage.getItem('authToken'); }
function getCurrentUser() {
const s = localStorage.getItem('currentUser');
return s ? JSON.parse(s) : null;
}
function userRole(user) {
if (!user?.role) return null;
const role = user.role;
if (typeof role === 'string') return role.toUpperCase();
if (typeof role === 'object' && role.name) return
String(role.name).toUpperCase();
```

```
return null;
}
function isAdminUser(user) {
return userRole(user) === 'ADMIN';
}
function redirectAfterAuth(fallback = '/') {
const redirect = new URLSearchParams(location.search).get('redirect');
location.href = redirect && redirect.startsWith('/') ? redirect : fallback;
}
function requireLogin(redirect = '/') {
if (getCurrentUser()) return true;
location.href = '/login.html?redirect=' + encodeURIComponent(redirect);
return false;
}
function parseApiError(errorBody, status) {
if (!errorBody || typeof errorBody !== 'object') return `HTTP ${status}`;
if (errorBody.detail) return errorBody.detail;
if (errorBody.errors) return Object.entries(errorBody.errors).map(([k, v]) =>
`${k}: ${v}`).join('; ');
if (errorBody.message) return errorBody.message;
return `HTTP ${status}`;
}
function normalizeApiPath(url) {
if (!url || !url.startsWith('/')) return url;
if (url.startsWith('/api/')) return url;
if (url === '/games' || url.startsWith('/games/') ||
url.startsWith('/games?')) {
return '/api' + url;
}
if (url === '/orders' || url.startsWith('/orders/') ||
url.startsWith('/orders?')) {
return '/api' + url;
}
return url;
```

```
}
async function apiRequest(url, options = {}) {
const method = (options.method || 'GET').toUpperCase();
const headers = { Accept: 'application/json', ...options.headers };
const apiPath = normalizeApiPath(url);
const isPublicAuth = apiPath.startsWith('/api/auth/');
const token = getAuthToken();
if (token && !isPublicAuth) headers['Authorization'] = token;
if (options.body != null && method !== 'GET' && method !== 'HEAD') {
headers['Content-Type'] = 'application/json';
}
const response = await fetch(API_BASE + apiPath, { ...options, headers });
if (!response.ok) {
const error = await response.json().catch(() => ({}));
throw new Error(parseApiError(error, response.status));
}
if (response.status === 204) return null;
return response.json();
}
const authAPI = {
register: (username, email, password) => apiRequest('/api/auth/register', {
method: 'POST', body: JSON.stringify({ username, email, password }) }),
login: (username, password) => apiRequest('/api/auth/login', { method:
'POST', body: JSON.stringify({ username, password }) })
};
const gamesAPI = {
listGenres: () => apiRequest('/api/games/genres'),
listGames: (skip = 0, limit = 100, title = null, genre = null, minPrice =
null, maxPrice = null) => {
let url = `/api/games?skip=${skip}&limit=${limit}`;
if (title) url += `&title=${encodeURIComponent(title)}`;
if (genre) url += `&genre=${encodeURIComponent(genre)}`;
if (minPrice != null && minPrice !== '') url += `&minPrice=${minPrice}`;
if (maxPrice != null && maxPrice !== '') url += `&maxPrice=${maxPrice}`;
```

```
return apiRequest(url);
},
listPopular: (limit = 5) => apiRequest(`/api/games/popular?limit=${limit}`),
getGame: (id) => apiRequest(`/api/games/${id}`),
createGame: (d) => apiRequest('/api/games', { method: 'POST', body:
JSON.stringify(d) }),
updateGame: (id, d) => apiRequest(`/api/games/${id}`, { method: 'PUT', body:
JSON.stringify(d) }),
deleteGame: (id) => apiRequest(`/api/games/${id}`, { method: 'DELETE' }),
async uploadGameImage(id, file) {
const headers = {};
const token = getAuthToken();
if (token) headers['Authorization'] = token;
const fd = new FormData();
fd.append('file', file);
const r = await fetch(API_BASE + normalizeApiPath(`/api/games/${id}/image`),
{ method: 'POST', headers, body: fd });
if (!r.ok) throw new Error(parseApiError(await r.json().catch(() => ({})),
r.status));
return r.json();
}
};
const wishlistAPI = {
listWishlist: () => apiRequest('/api/wishlist'),
isInWishlist: (gameId) => apiRequest(`/api/wishlist/game/${gameId}/exists`),
addToWishlist: (gameId) => apiRequest(`/api/wishlist/game/${gameId}`, {
method: 'POST' }),
removeFromWishlist: (gameId) => apiRequest(`/api/wishlist/game/${gameId}`, {
method: 'DELETE' })
};
const paymentsAPI = {
sandboxPay: (orderId, cardNumber, expiry, cvv) =>
apiRequest('/api/payments/sandbox', {
method: 'POST', body: JSON.stringify({ orderId: Number(orderId), cardNumber,
```

```
expiry, cvv })
})
};
const ordersAPI = {
listOrders: (skip = 0, limit = 100, mine = false) => {
let url = `/api/orders?skip=${skip}&limit=${limit}`;
if (mine) url += '&mine=true';
return apiRequest(url);
},
getOrder: (id) => apiRequest(`/api/orders/${id}`),
createOrder: (d) => apiRequest('/api/orders', { method: 'POST', body:
JSON.stringify(d) }),
deleteOrder: (id) => apiRequest(`/api/orders/${id}`, { method: 'DELETE' })
};
window.GAME_GENRES = [];
async function loadGenres() {
try { window.GAME_GENRES = await gamesAPI.listGenres(); }
catch { window.GAME_GENRES =
['Action','Adventure','RPG','Strategy','Sports','Racing','Shooter','Puzzle','
Horror','Simulation','Indie','MMORPG']; }
}
function renderGenreOptions(selected = '', all = true) {
let h = all ? '<option value="">Всі жанри</option>' : '<option value=""
disabled selected>Оберіть жанр</option>';
return h + (window.GAME_GENRES || []).map(g => `<option
value="${escapeHtml(g)}"${g === selected ? ' selected' :
''}>${escapeHtml(g)}</option>`).join('');
}
function renderGameImage(game) {
return game.imageUrl ? `<img
src="${escapeHtml(resolveAssetUrl(game.imageUrl))}"
alt="${escapeHtml(game.title || '')}" class="game-card-img">` : '🎮';
}
function gameLauncher(game) {
```

```
return game?.launcher || game?.platform || '';
}
function renderRatingBadge(game) {
if (!game?.reviewCount) return '';
const avg = game.avgRating != null ? Number(game.avgRating).toFixed(1) : '—';
return `<span class="rating-badge" title="${game.reviewCount} відгуків">★
${avg}</span>`;
}
function escapeHtml(text) {
if (!text) return '';
const d = document.createElement('div');
d.textContent = text;
return d.innerHTML;
}
```

Лістинг С.2 – cart.js

```
const CART_KEY = 'keygames_cart';
function getCart() {
try {
return JSON.parse(localStorage.getItem(CART_KEY) || '[]').map(i => ({ ...i,
quantity: 1 }));
} catch {
return [];
}
}
function saveCart(cart) {
localStorage.setItem(CART_KEY, JSON.stringify(cart));
if (typeof renderSidebar === 'function') renderSidebar();
}
function getCartCount() {
return getCart().length;
}
function addToCart(game) {
const cart = getCart();
const ex = cart.find(i => i.gameId === game.id);
```

```
if (ex) {
ex.price = game.price;
ex.title = game.title;
ex.imageUrl = game.imageUrl;
ex.stock = game.stock;
ex.genre = game.genre;
ex.launcher = gameLauncher(game);
saveCart(cart);
return false;
}
cart.push({
gameId: game.id,
title: game.title,
price: game.price,
imageUrl: game.imageUrl,
genre: game.genre,
launcher: gameLauncher(game),
stock: game.stock,
quantity: 1
});
saveCart(cart);
return true;
}
function removeFromCart(gameId) {
saveCart(getCart().filter(i => i.gameId !== gameId));
}
function clearCart() {
localStorage.removeItem(CART_KEY);
if (typeof renderSidebar === 'function') renderSidebar();
}
Function getCartTotal() {
return getCart().reduce((s, i) => s + parseFloat(i.price), 0);
}
```

Лістинг С.3 – checkout.js

```
let checkoutItems = [];
document.addEventListener('DOMContentLoaded', async () => {
const user = getCurrentUser();
if (!user) {
location.href = '/login.html?redirect=' + encodeURIComponent('/checkout.html'
+ location.search);
return;
}
try {
const params = new URLSearchParams(location.search);
const gameId = params.get('gameId');
if (gameId) {
checkoutItems = [{ game: await gamesAPI.getGame(gameId), quantity: 1 }];
} else {
const cart = getCart();
if (!cart.length) {
showError('Кошик порожній');
return;
}
checkoutItems = await Promise.all(
cart.map(async c => ({ game: await gamesAPI.getGame(c.gameId), quantity: 1
}))
);
const n = document.getElementById('customer-name');
const e = document.getElementById('customer-email');
if (n) n.value = user.username || '';
if (e) e.value = user.email || '';

renderSummary();
document.getElementById('checkout-form')?.addEventListener('submit',
handleCheckout);
} catch (err) {
showError(err.message || 'Не вдалося завантажити замовлення');
}
```

```
});
function renderSummary() {
const el = document.getElementById('order-summary');
if (!el) return;
let total = 0;
const rows = checkoutItems.map(({ game }) => {
total += parseFloat(game.price);
return `<div class="summary-
row"><span>${escapeHtml(game.title)}</span><span>${formatPrice(game.price)}
₴</span></div>`;
}).join('');
el.innerHTML = rows + `<div class="summary-row summary-
total"><span><strong>До
сплати</strong></span><span><strong>${formatPrice(total)}
₴</strong></span></div>`;
}
async function handleCheckout(e) {
e.preventDefault();
document.getElementById('error-message').style.display = 'none';
if (!checkoutItems.length) {
showError('Кошик порожній');
return;
}
for (const { game, quantity } of checkoutItems) {
if (quantity > game.stock) {
showError(`Недостатньо "${game.title}"`);
return;
}
const btn = e.target.querySelector('button[type="submit"]');
if (btn) btn.disabled = true;
try {
const order = await ordersAPI.createOrder({
customerName: document.getElementById('customer-name').value.trim(),
customerEmail: document.getElementById('customer-email').value.trim(),
```

```
items: checkoutItems.map(({ game, quantity }) => ({ gameId: game.id, quantity
}))
});
if (!order?.id) {
throw new Error('Сервер не повернув номер замовлення');
}
location.href = `/payment.html?orderId=${order.id}`;
} catch (err) {
showError(err.message);
if (btn) btn.disabled = false;
}
function showError(msg) {
const el = document.getElementById('error-message');
if (el) {
el.textContent = msg;
el.style.display = 'block';
}
function formatPrice(p) { return parseFloat(p).toFixed(2); }
```

Лістинг С.4 – payment.js

```
document.addEventListener('DOMContentLoaded', async () => {
const user = getCurrentUser();
if (!user) {
location.href = '/login.html?redirect=' +
encodeURIComponent(location.pathname + location.search);
return;
const orderId = new URLSearchParams(location.search).get('orderId');
if (!orderId) {
location.href = '/profile.html';
return;
}
try {
const order = await ordersAPI.getOrder(orderId);
if (order.status === 'paid') {
document.getElementById('payment-form')?.remove();
```

```
document.getElementById('payment-summary').innerHTML = `
<p>Замовлення #${order.id}</p>
<p><span class="order-status paid">Оплачено</span></p>
<a href="/profile.html" class="btn btn-primary btn-block" style="margin-
top:16px">До профілю</a>`;
return;
}
if (order.status !== 'pending_payment') {
document.getElementById('payment-form')?.remove();
document.getElementById('payment-summary').innerHTML = `<div class="error-
message">Це замовлення недоступне для оплати</div>`;
return;
}
const total = order.items.reduce((s, i) => s + parseFloat(i.unitPrice) *
i.quantity, 0);
document.getElementById('payment-summary').innerHTML = `
<p>Замовлення #${order.id}</p>
<p><span class="order-status pending_payment">Очікує оплати</span></p>
<p style="margin-top:12px"><strong>${total.toFixed(2)} ₴</strong></p>
<div class="sandbox-hint">Тестова картка: <code>4242 4242 4242
4242</code></div>`;
} catch (e) {
document.getElementById('payment-form')?.remove();
document.getElementById('payment-summary').innerHTML = `<div class="error-
message">${escapeHtml(e.message)}</div>`;
setupCardNumberInput();
setupExpiryInput();
setupCvvInput();
document.getElementById('payment-form')?.addEventListener('submit', async e
=> {
e.preventDefault();
try {
const r = await paymentsAPI.sandboxPay(orderId,
document.getElementById('card-number').value,
```

```
document.getElementById('card-expiry').value,
document.getElementById('card-cvv').value);
if (r.success) {
clearCart();
showPaymentSuccess();
setTimeout(() => location.href = '/profile.html', 2200);
} else {
document.getElementById('payment-error').textContent = r.message;
document.getElementById('payment-error').style.display = 'block';
}
} catch (err) {
document.getElementById('payment-error').textContent = err.message;
document.getElementById('payment-error').style.display = 'block';
}
});
});
function setupCardNumberInput() {
const el = document.getElementById('card-number');
if (!el) return;
el.setAttribute('maxlength', '19');
el.setAttribute('inputmode', 'numeric');
el.addEventListener('input', () => {
const digits = el.value.replace(/\D/g, '').slice(0, 16);
el.value = digits.match(/.{1,4}/g)?.join(' ') || '';
});
function showPaymentSuccess() {
document.querySelector('.success-overlay')?.remove();
const overlay = document.createElement('div');
overlay.className = 'success-overlay';
overlay.innerHTML = `
<div class="success-overlay-card">
<div class="success-overlay-icon">✓</div>
<h2 class="success-overlay-title">Оплата успішна!</h2>
<p class="success-overlay-text">Ключі доступні у вашому профілі</p>
```

```
</div>`;
document.body.appendChild(overlay);
}
function setupExpiryInput() {
const el = document.getElementById('card-expiry');
if (!el) return;
el.setAttribute('maxlength', '5');
el.setAttribute('inputmode', 'numeric');
el.addEventListener('input', () => {
const digits = el.value.replace(/\D/g, '').slice(0, 4);
el.value = digits.length > 2 ? `${digits.slice(0, 2)}/${digits.slice(2)}` :
digits;
});
}
function setupCvvInput() {
const el = document.getElementById('card-cvv');
if (!el) return;
el.setAttribute('maxlength', '3');
el.setAttribute('inputmode', 'numeric');
el.addEventListener('input', () => {
el.value = el.value.replace(/\D/g, '').slice(0, 3);
});
}
```

Додаток В. Лістинг програмного коду серверних класів обробки замовлень (Java)

```
package com.example.gamestore.controller;
import java.math.BigDecimal;
import java.net.URI;
import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
import com.example.gamestore.dto.GameRequest;
import com.example.gamestore.dto.GameResponse;
import com.example.gamestore.dto.GameUpdateRequest;
import com.example.gamestore.model.GameGenres;
import com.example.gamestore.service.GameService;
import jakarta.validation.Valid;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Positive;
import jakarta.validation.constraints.PositiveOrZero;
@Validated
@RestController
@RequestMapping({"/api/games", "/games"})
public class GameController {
```

```
private final GameService gameService;
public GameController(GameService gameService) {
this.gameService = gameService;
}
@GetMapping("/genres")
public List<String> listGenres() {
return GameGenres.ALL;
}
@GetMapping("/popular")
public List<GameResponse> listPopularGames(
@RequestParam(defaultValue = "5") @Positive @Max(20) int limit
) {
return gameService.listPopularGames(limit);
}
@GetMapping
public List<GameResponse> listGames(
@RequestParam(defaultValue = "0") @PositiveOrZero int skip,
@RequestParam(defaultValue = "100") @Positive @Max(1000) int limit,
@RequestParam(required = false) String title,
@RequestParam(required = false) String genre,
@RequestParam(required = false) BigDecimal minPrice,
@RequestParam(required = false) BigDecimal maxPrice
) {
return gameService.listGames(skip, limit, title, genre, minPrice, maxPrice);

@GetMapping("/{id}")
public GameResponse getGame(@PathVariable Long id) {
return gameService.getGame(id);
}
@PostMapping(value = {"", "/"})
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<GameResponse> createGame(@Valid @RequestBody
GameRequest request) {
return
```

```
ResponseEntity.status(HttpStatus.CREATED).body(gameService.createGame(request
));
}
@PostMapping("/bulk")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<List<GameResponse>> createGames(@Valid @RequestBody
List<GameRequest> requests) {
return
ResponseEntity.status(HttpStatus.CREATED).body(gameService.createGames(reques
ts));
}
@PutMapping("/{id}")
@PreAuthorize("hasRole('ADMIN')")
public GameResponse updateGame(@PathVariable Long id, @Valid @RequestBody
GameUpdateRequest request) {
return gameService.updateGame(id, request);
}
@GetMapping("/{id}/image")
public ResponseEntity<Void> redirectGameImage(@PathVariable Long id) {
GameResponse game = gameService.getGame(id);
if (game.imageUrl() == null || game.imageUrl().isBlank()) {
return ResponseEntity.notFound().build();
}
return
ResponseEntity.status(HttpStatus.FOUND).location(URI.create(game.imageUrl()))
.build();
}
@PostMapping("/{id}/image")
@PreAuthorize("hasRole('ADMIN')")
public GameResponse uploadGameImage(@PathVariable Long id,
@RequestParam("file") MultipartFile file) throws java.io.IOException {
return gameService.uploadGameImage(id, file);
@DeleteMapping("/{id}")
@PreAuthorize("hasRole('ADMIN')")
```

```
public ResponseEntity<Void> deleteGame(@PathVariable Long id) {
gameService.deleteGame(id);
return ResponseEntity.noContent().build();
}
}
```

Лістинг D.2 – OrderController.java

```
package com.example.gamestore.controller;
import java.util.List;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import com.example.gamestore.dto.OrderRequest;
import com.example.gamestore.dto.OrderResponse;
import com.example.gamestore.service.OrderService;
import jakarta.validation.Valid;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Positive;
import jakarta.validation.constraints.PositiveOrZero;
@Validated
@RestController
@RequestMapping({"/api/orders", "/orders"})
public class OrderController {
private final OrderService orderService;
public OrderController(OrderService orderService) {
```

```
this.orderService = orderService;
}
@GetMapping
public ResponseEntity<List<OrderResponse>> listOrders(
Authentication auth,
@RequestParam(name = "skip", defaultValue = "0") @PositiveOrZero int skip,
@RequestParam(name = "limit", defaultValue = "100") @Positive @Max(1000) int
limit,
@RequestParam(name = "mine", defaultValue = "false") boolean mine
) {
return ResponseEntity.ok(orderService.listOrders(skip, limit, auth.getName(),
isAdmin(auth), mine));
}
@GetMapping("/{id}")
public ResponseEntity<OrderResponse> getOrder(@PathVariable Long id,
Authentication auth) {
String username = auth != null ? auth.getName() : null;
return ResponseEntity.ok(orderService.getOrder(id, username, isAdmin(auth)));
}
@PostMapping
public ResponseEntity<OrderResponse> createOrder(
@Valid @RequestBody OrderRequest request,
Authentication auth
) {
String username = auth != null ? auth.getName() : null;
OrderResponse response = orderService.createOrder(request, username);
return ResponseEntity.status(HttpStatus.CREATED).body(response);
}
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteOrder(@PathVariable Long id, Authentication
auth) {
orderService.deleteOrder(id, auth.getName(), isAdmin(auth));
return ResponseEntity.noContent().build();
}
```

```
private boolean isAdmin(Authentication auth) {
if (auth == null) {
return false;
}
return auth.getAuthorities().stream()
.map(GrantedAuthority::getAuthority)
.anyMatch("ROLE_ADMIN"::equals);
}
```

Лістинг D.3 – OrderService.java

```
package com.example.gamestore.service;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.gamestore.dto.OrderItemRequest;
import com.example.gamestore.dto.OrderItemResponse;
import com.example.gamestore.dto.OrderRequest;
import com.example.gamestore.dto.OrderResponse;
import com.example.gamestore.model.Game;
import com.example.gamestore.model.Order;
import com.example.gamestore.model.OrderItem;
import com.example.gamestore.model.User;
import com.example.gamestore.repository.GameRepository;
import com.example.gamestore.repository.OrderRepository;
import jakarta.persistence.EntityNotFoundException;
@Service
public class OrderService {
private final OrderRepository orderRepository;
private final GameRepository gameRepository;
private final GameKeyService gameKeyService;
```

```
private final UserService userService;
public OrderService(OrderRepository orderRepository, GameRepository
gameRepository,
GameKeyService gameKeyService, UserService userService) {
this.orderRepository = orderRepository;
this.gameRepository = gameRepository;
this.gameKeyService = gameKeyService;
this.userService = userService;
}
@Transactional(readOnly = true)
public List<OrderResponse> listOrders(int skip, int limit, String username,
boolean isAdmin, boolean mineOnly) {
int page = skip / Math.max(limit, 1);
PageRequest pageable = PageRequest.of(page, limit,
Sort.by(Sort.Direction.DESC, "id"));
if (isAdmin && !mineOnly) {
return
orderRepository.findAll(pageable).stream().map(this::toResponse).toList();
}
User user = userService.findByUsername(username);
return orderRepository.findForUser(user.getId(), user.getEmail(), pageable)
.stream().map(this::toResponse).toList();
}
@Transactional(readOnly = true)
public OrderResponse getOrder(Long id, String username, boolean isAdmin) {
Order order = orderRepository.findById(id)
.orElseThrow(() -> new EntityNotFoundException("Order not found"));
if (isAdmin || canAccess(order, username)) {
return toResponse(order);
}
throw new AccessDeniedException("Немає доступу до замовлення");
}
@Transactional(readOnly = true)
public void assertCanPay(Long orderId, String username) {
```

```
Order order = orderRepository.findById(orderId)
.orElseThrow(() -> new EntityNotFoundException("Order not found"));
if (!canAccess(order, username)) {
throw new AccessDeniedException("Оплатити може лише покупець цього
замовлення");
}
if (!"pending_payment".equals(order.getStatus())) {
throw new IllegalArgumentException("Замовлення вже оплачене або недоступне
для оплати");
}
}
@Transactional
public OrderResponse createOrder(OrderRequest request, String username) {
if (username == null) {
throw new AccessDeniedException("Увійдіть, щоб оформити замовлення");
}
Order order = new Order();
order.setCustomerName(request.customerName());
order.setCustomerEmail(request.customerEmail());
order.setCreatedAt(LocalDateTime.now());
order.setStatus("pending_payment");
User user = userService.findByUsername(username);
order.setUserId(user.getId());
order.setCustomerEmail(user.getEmail());
List<OrderItem> items = new ArrayList<>();
for (OrderItemRequest ir : request.items()) {
Game game = gameRepository.findById(ir.gameId())
.orElseThrow(() -> new EntityNotFoundException("Game " + ir.gameId() + " not
found"));
if (Boolean.FALSE.equals(game.getIsActive()) || game.getStock() <
ir.quantity()) {
throw new IllegalArgumentException("Гра недоступна в потрібній кількості");
}
OrderItem oi = new OrderItem();
```

```
oi.setOrder(order);
oi.setGame(game);
oi.setQuantity(ir.quantity());
oi.setUnitPrice(game.getPrice());
items.add(oi);
}
order.setItems(items);
return toResponse(orderRepository.save(order));
}
@Transactional
public OrderResponse completePayment(Long orderId) {
Order order = orderRepository.findById(orderId).orElseThrow(() -> new
EntityNotFoundException("Order not found"));
if ("paid".equals(order.getStatus())) return toResponse(order);
if (!"pending_payment".equals(order.getStatus())) {
throw new IllegalArgumentException("Замовлення не можна оплатити");
}
for (OrderItem item : order.getItems()) {
Game game = item.getGame();
if (game.getStock() < item.getQuantity()) {
order.setStatus("failed");
orderRepository.save(order);
throw new IllegalArgumentException("Недостатньо товару: " + game.getTitle());
}
game.setStock(game.getStock() - item.getQuantity());
gameKeyService.assignKeys(item, item.getQuantity());
}
order.setStatus("paid");
return toResponse(orderRepository.save(order));
}
@Transactional
public void deleteOrder(Long id, String username, boolean isAdmin) {
Order order = orderRepository.findById(id)
.orElseThrow(() -> new EntityNotFoundException("Order not found"));
```

```
if (!isAdmin && !canAccess(order, username)) {
throw new AccessDeniedException("Можна видалити лише власні замовлення");
}
if ("paid".equals(order.getStatus())) {
for (OrderItem item : order.getItems()) {
gameKeyService.releaseKeys(item);
Game game = item.getGame();
game.setStock(game.getStock() + item.getQuantity());
}
}
orderRepository.delete(order);
}
@Transactional
public OrderResponse failPayment(Long orderId) {
Order order = orderRepository.findById(orderId).orElseThrow(() -> new
EntityNotFoundException("Order not found"));
if ("pending_payment".equals(order.getStatus())) {
order.setStatus("failed");
orderRepository.save(order);
}
return toResponse(order);
}
private boolean canAccess(Order order, String username) {
if (username == null) {
return false;
}
User user = userService.findByUsername(username);
if (order.getUserId() != null) {
return order.getUserId().equals(user.getId());
}
return order.getCustomerEmail() != null
&& order.getCustomerEmail().equalsIgnoreCase(user.getEmail());
}
private OrderResponse toResponse(Order order) {
```

```
List<OrderItemResponse> items = order.getItems().stream()
.map(i -> new OrderItemResponse(i.getId(), i.getGame().getId(),
i.getQuantity(), i.getUnitPrice(),
"paid".equals(order.getStatus()) ?
gameKeyService.getKeysForOrderItem(i.getId()) : List.of()))
.toList();
return new OrderResponse(order.getId(), order.getCustomerName(),
order.getCustomerEmail(),
order.getCreatedAt(), order.getStatus(), items);
```

Лістинг D.4 – PaymentService.java

```
package com.example.gamestore.service;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.example.gamestore.dto.OrderResponse;
import com.example.gamestore.dto.PaymentSandboxRequest;
import com.example.gamestore.dto.PaymentSandboxResponse;
@Service
public class PaymentService {
private static final String TEST_CARD = "4242424242424242";
private final OrderService orderService;
public PaymentService(OrderService orderService) {
this.orderService = orderService;
}
@Transactional
public PaymentSandboxResponse pay(PaymentSandboxRequest request, String
username) {
orderService.assertCanPay(request.orderId(), username);
String card = request.cardNumber().replaceAll("\\s+", "");
if (!TEST_CARD.equals(card)) {
OrderResponse failed = orderService.failPayment(request.orderId());
return new PaymentSandboxResponse(false, "Відхилено. Тестова картка: 4242
4242 4242 4242", failed);
}
OrderResponse paid = orderService.completePayment(request.orderId());
```

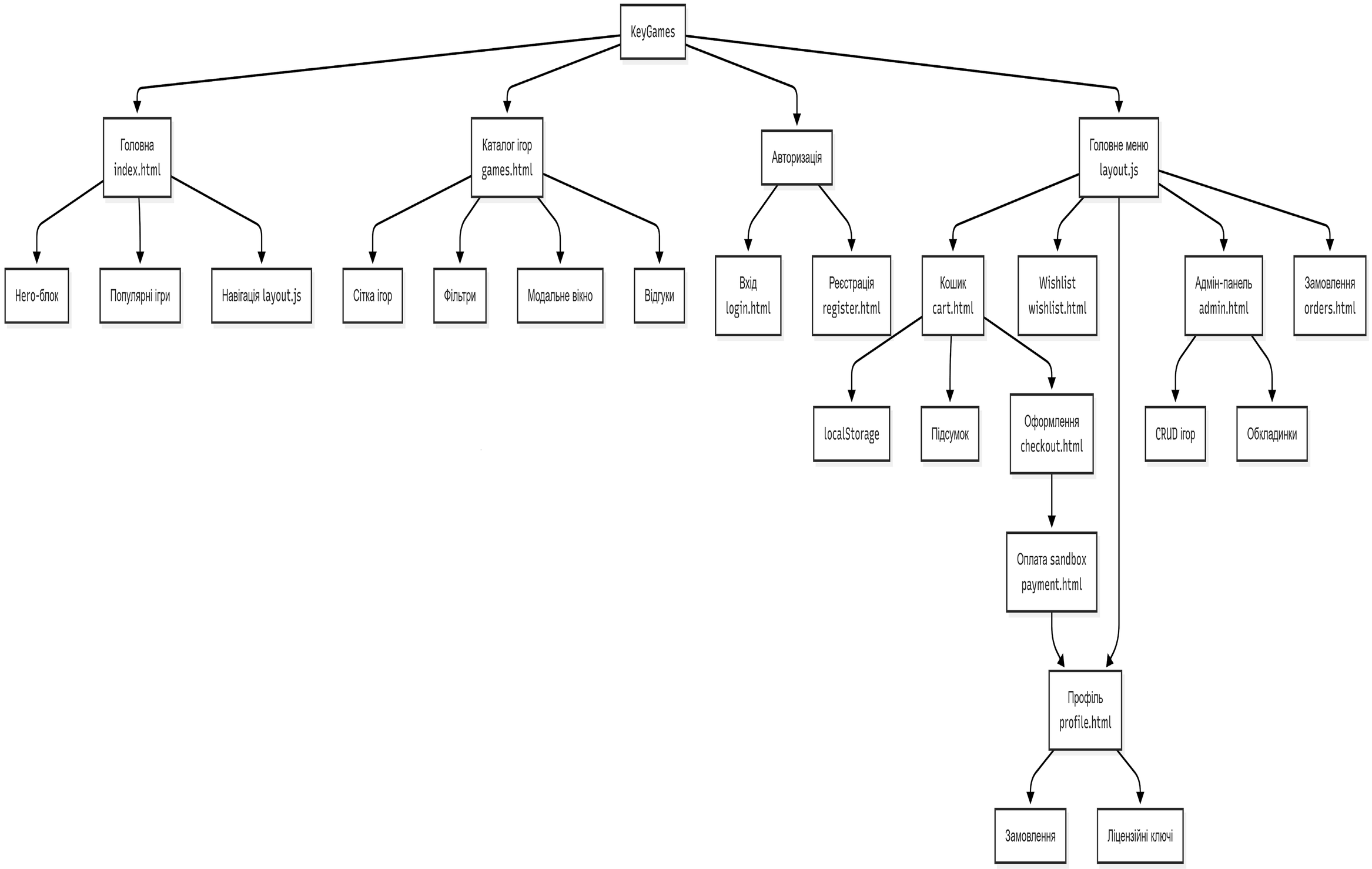
```
return new PaymentSandboxResponse(true, "Оплата успішна! Ключі доставлено.",
paid);
}
}
```

Додаток Д. Лістинг програмного коду скрипта бази даних PostgreSQL

```
CREATE USER "KeyGames" WITH PASSWORD '1234567890';
CREATE DATABASE game_store
WITH
OWNER = "KeyGames"
ENCODING = 'UTF8'
TABLESPACE = pg_default
CONNECTION LIMIT = -1;
GRANT ALL PRIVILEGES ON DATABASE game_store TO "KeyGames";
GRANT ALL ON SCHEMA public TO "KeyGames";
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON TABLES TO "KeyGames";
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON SEQUENCES TO
"KeyGames";
COMMENT ON DATABASE game_store IS 'KeyGames - база даних для магазину ігор';
```

2026.КВР.122.421.19.00.00 СС

Схема структурна вебсервісу



						2026.КВР.122.4.21.19.00.00 СС	
Изм.	Лист	№ докум.	Подп.	Дата	Розробка вебсервісу дистрибуції комп'ютерних ігор «KeyGames» Схема структурна вебсервісу	Лит.	Масса	Масштаб
Разраб.		Шевяков А.С.						
Пров.		Слободян Р.О.				Лист	Листов 1	
Т.контр.								
Реценз.						ВСП ТФК ТНТУ КН-421 м Тернопіль		
Н.контр.		Прийомак В.А.						
Утв.								

Копировал

Формат А1

Перв. примен.

Справ. №

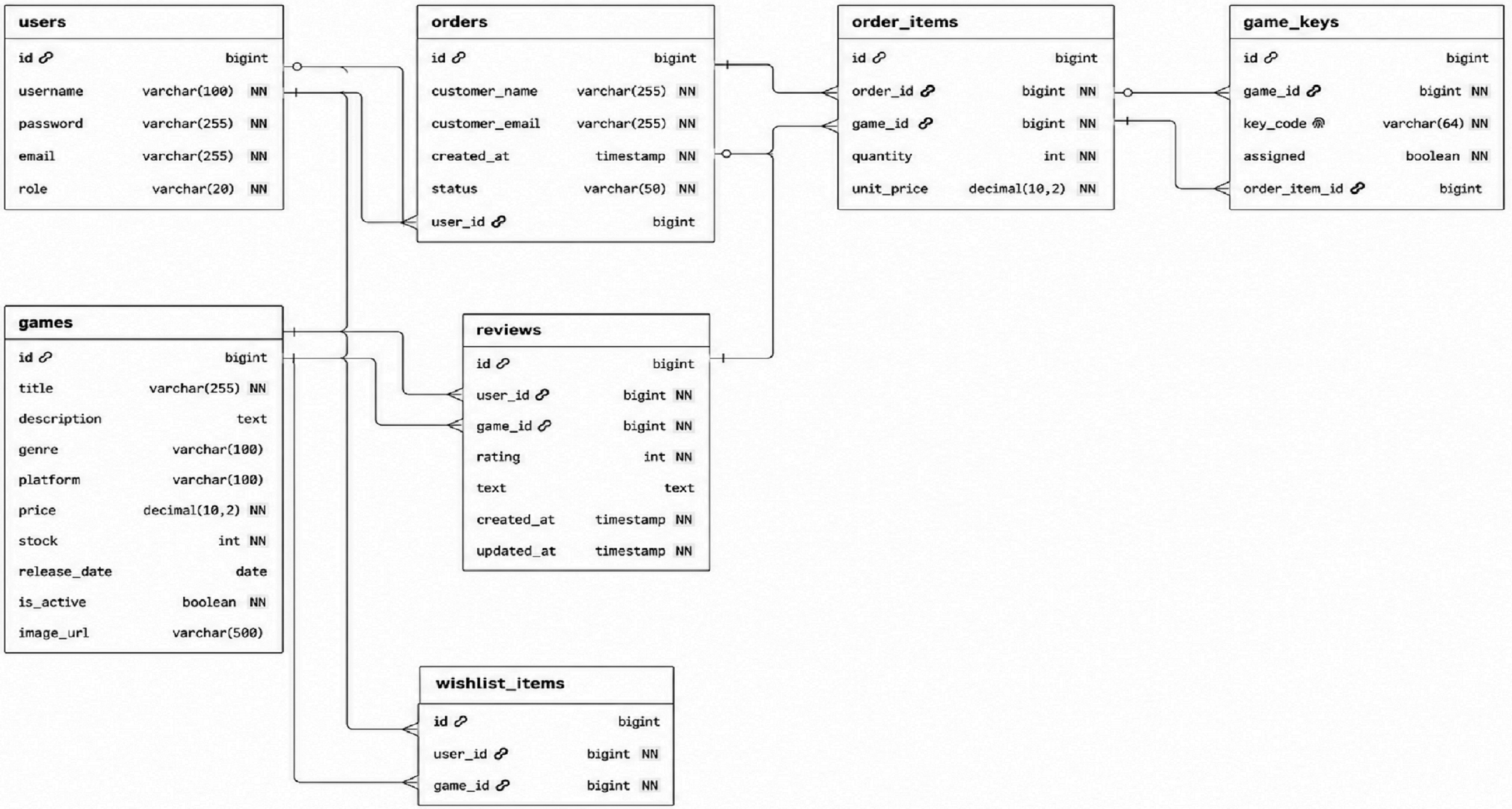
Подп. и дата

Инв. № подл. | Взам. инв. № | Инв. № дубл. | Подп. и дата

Таблиця техніко-економічний показників

№	Показник	Одиниця вимірювання	Значення
1	Середовище розробки	-	IntelliJ IDEA, Visual Studio Code
2	Тип інтерфейсу	-	Веб-інтерфейс
3	Frontend	-	HTML5, CSS3, JavaScript
4	Backend	-	Java 17 / Spring Boot 3.3.1
5	База даних	-	PostgreSQL
6	Розгортання	-	Vercel, Railway
7	Загальний розмір програми	МБ	20
8	Собівартість розолдки	ГРН	98 013
9	Плановий річний грошовий потік	ГРН	54 887
10	Ціна програмношо продукуту	ГРН	152 900
11	Чиста теперіішня вартість	ГРН	38 513
12	Термін окупності	рік	2.1

ER-діаграма бази даних



Изм.	Лист	№ докум.	Підп.	Дата	2026.КВР.122.421.19.00.00.БД		
Розроб.		Шевяков А.С.			Розробка вебсервісу дистрибуції комп'ютерних ігор «KeyGames»	Літ.	Маса	Масштаб
Пров.		Слободян Р.О.			ER-діаграма бази даних			
Т.контр.						Лист	Листів 1	
Резенз.						ВСП ТФК ТНТУ КН-421 м.Тернопіль		
Н.контр.		Приймак В.А.						
Утв.								

Копіював Формат А1

Перед. примен. | Справ. № | Підп. і дата | Інв. № дубл. | Взам. інв. № | Підп. і дата | Інв. № подл.

UML-діаграма варіантів використання