НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ТЕПЛОЕНЕРГЕТИЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА АВТОМАТИЗАЦІЇ ПРОЕКТУВАННЯ ЕНЕРГЕТИЧНИХ ПРОЦЕСІВ І СИСТЕМ

КУРСОВА РОБОТА

З кредитного модулю:

Основи web-програмування-2

на тему: **онлайн магазин для покупки аккаунтів та ключів для ігор**

Студента ІІ курсу, групи ТІ-92
Черноусова Дениса Ігоровича

Спеціальність 121 – Інженерія програмного забезпечення
Освітня програма Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій

Керівник доцент, к.т.н., Титенко С. В.

Національна оцінка_____

Кількість балів: _____ оцінка: ECTS_____

Члени комісії: _____ _____
(вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(вчене звання, науковий ступінь, прізвище та ініціали)

Засвідчую, що у цій курсовій роботі немає запозичень праць інших авторів без відповідних посилань

Студент _____ Д. І. Черноусов

КИЇВ 2021

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ТЕПЛОЕНЕРГЕТИЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА АВТОМАТИЗАЦІЇ ПРОЕКТУВАННЯ ЕНЕРГЕТИЧНИХ ПРОЦЕСІВ І СИСТЕМ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до курсової роботи на тему:

онлайн магазин для покупки аккаунтів та ключів для ігор

Спеціальність 121 – Інженерія програмного забезпечення
Освітня програма Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій

з кредитного модулю:
«Основи web-програмування-2. Курсова робота»

Виконав студент групи ТІ-92 _____ Д. І. Черноусов
(підпис, дата)

Керівник роботи, к.т.н, доц. _____ С. В. Титенко
(підпис, дата)

Київ 2021

ЗМІСТ

ВСТУП

Сьогодні комп’ютерні ігри - це такий самий продукт, як холодильник чи праска, або навіть одяг. Торгівля товарами, якими вже користувались нині досить поширена, адже можна купити товар, який всього-на-всього трохи гірший від нового за значно меншою ціною. З появою платформ для покупки ігор, таких як steam, покупка та продаж ігор стали звичайною справою, але у ігор також є своя ціна, іноді досить висока. Для людей, які через певні причини не бажали витрачати кошти на ігри, з’явилось нове рішення. Так як багато ігор після проводження в них певного часу просто “припадають пилом” у віртуальному сховищі, з’явився спосіб після використання віддати їх іншим людям за значно нижчою ціною – це сайти які є платформами для торгівлі ігровими ключами, аккаунтами та іншими даними, які дають доступ до ліцензійної гри продавця.

Принцип роботи таких магазинів дуже простий. Спочатку ви віддаєте їм ваші дані і згідно правил сайтів затверджуєте ціну, та частку доходу для сайту. Потім, коли хтось зайде на сайт і захоче купити ваш товар, отримають вигоду і покупець, бо він купив гру дешевше, і ви, бо отримали певні кошти за товар, який став вже непотребом, і сайт, бо він отримав частку від вартості.

Але не все так просто. Заходячи на такі сайти, користувач завжди ризикує, адже його можуть надурити і просто забрати кошти. Особливо підозріло виглядають сайти, що пропонують виграти ігри з певною ймовірністю, адже користувач не знає, яка вона насправді.

Сайт 4ourkeys – це чесний сайт для продажу та покупки ігор, бувших у використанні. Переваги цього сайту – це наявність відгуків, можливість як купити гру просто, так і виграти випадкову, яка буде чесною, адже немає сенсу робити нечесні випадковості на сайті, де можна просто купити гру, та хороший дизайн, що буде відрізняти його від нечесний сайтів, для яких не докладають багато зусиль.

1 ПОСТАНОВКА ЗАВДАННЯ

Завдання веб-системи полягає в тому, щоб надати користувачу доступ до зручного, добре оформленого, правильно та стабільно працюючого, чесного і безпечного інтернет-магазину. Для цього в першу чергу потрібно з’ясувати що ж таки інтернет-магазин. Якщо взяти визначення з сайту web-lighthouse.com, воно сформоване наступним чином: Інтернет-магазин це – електронний ресурс, сайт з певним каталогом, за допомогою якого відбувається прямий продаж товарів споживачеві, враховуючи доставку. При цьому розміщення інформації, замовлення товару і угода відбуваються прямо на сайті магазину. За цим визначенням, сайт повинен мати можливість вибору та покупки користувачами товарів, інформацію для споживача, яка є описом гри, доставку, яка у випадках з іграми безкоштовна та моментальна і угоду, яка формується при попередньому знайомленні користувача із правилами і після погодженні з ними, покупці товару.

За зручність та правильне оформлення відповідає дизайн сайту, який був створений на основі макету з веб-сайту figma. Зручність забезпечується завдяки схожості сайту на інші, подібні до нього і стандартному розташуванню блоків. Варто також зазначити, що ієрархія сторінок на сайті не є дуже складною, через що там дуже важко загубитись. Також сайт не перевантажений декораціями та непотрібними елементами, що дозволяє легко знаходити те, що потрібно. Оформлення сайту виконане по основним правилам дизайну, а саме: нормальний підбір кольорів, рівні відступи, правильно підібрана величина відступів для розмежування блоків та їх наповнення. Використані також іконки, що спрощують користування сайтом, роблячи його більш

інтуїтивно зрозумілим. Меню адаптоване під різні розміри екрану. Для форм є перевірка полів, що робить введення даних до них більш простою задачею для користувача.

Правильність та стабільність роботи веб-системи забезпечується наявністю бази даних та обробкою запитів сайту, коли користувач там щось робить. Завдяки обробці форм користувач може робити всі дії, доступні на сайті, не ризикуючи зламати систему. База даних дозволяє стабільно зберігати та редагувати великі об’єми інформації без ризику їх втрати. Завдяки базі даних сайт може зберігати різні статистичні дані, такі як кількість користувачів, загальна сума покупок або кількість покупок. Далеко не останню роль також відіграє наданий хостинг, потужності якого вистачає для нормальної роботи веб-системи, принаймні при невеликій кількості користувачів та інформації про них.

Чесність веб-системи забезпечується наявністю двох варіантів покупки товарів, що має збільшувати ймовірність того, що користувач, який вперше зайшов на сайт, не буде сприймати його як нечесний, бо якщо він сумнівається в запропонованому випадковому виборі гри, то може просто купити без ризику. Також реалізована система відгуків, завдяки якій новий користувач зможе почитати думки інших користувачів про ігри на сайті, справедливість ціни та про сайт в цілому. Якщо після перегляду цих можливостей користувач все ще буде мати сумніви, в меню є інформація про сайт та контакти, де можна детальніше ознайомитись з правилами та отримати відповіді на свої запитання.

Безпека – дуже важливий компонент будь-якого інтернет-магазину. Використання хешування паролів, post-запитів та наявність проміжної ланки між сайтом та базою даних, робить його достатньо безпечним для користування. Окрім звичайної реєстрації на сайт можна також увійти, маючи аккаунт на платформі steam. Важливим для безпеки також є те, що гості не можуть ні купувати товари, ні залишати коментарі, що має вберегти сайт від неприйнятних коментарів, бо хто захоче писати подібний коментар, якщо він вже зареєструвався і у сайту є його дані.

Завдяки реалізації цих компонентів, сайт має виконувати свої функції повноцінно, тобто, як вже було описано, бути нормальним інтернет-магазином, а також

збільшити ту невелику кількість сайтів, які чесно торгують іграми, бувшими в використанні. Таким чином задача веб-системи – виконувати всі реалізовані в ній функції, а розробників – забезпечити її підтримку за допомогою певної роботи з клієнтами, забезпечення нормального хостингу і можливо навіть наглядом за сайтом і виявлення користувачів, які використовують його не за призначенням

2 ЗАГАЛЬНА АРХІТЕКТУРА ВЕБ-СИСТЕМИ

2.1 Загальний принцип роботи

Веб система складається з чотирьох основних блоків – власне сайту, де розташовані сторінки з певним наповненням, дизайном та частковою адаптивністю та певні елементи інтерактивності, такі як кейси, що можна описати як frontend, бази даних, написаної на mysql і розміщеної на phpMyAdmin, що містить інформацію про користувачів, товари, покупки, і дві проміжні ланки між базою даних та сайтом. Перша з них написана на javascript і відповідає за дані користувача, профіль, тобто баланс, покупки та відгуки і за відкриття кейсів(отримання з набору ігор різної ціни однієї з попередньою оплатою певної суми, встановленої сайтом за можливість виграти одну з ігор даного набору). Цю частину можна назвати backend1. За все інше: розміщення ігор на сайті, коректне виведення їх кількості та вартості, відображення зображень, дат, статистичних даних, а також основну взаємодію з базою даних відповідає частина написана на php з використанням фреймворку Laravel. Цю частину можна назвати як backend2. В кінцевому результаті отримуємо схему, зображену на рисунку 2.1.1

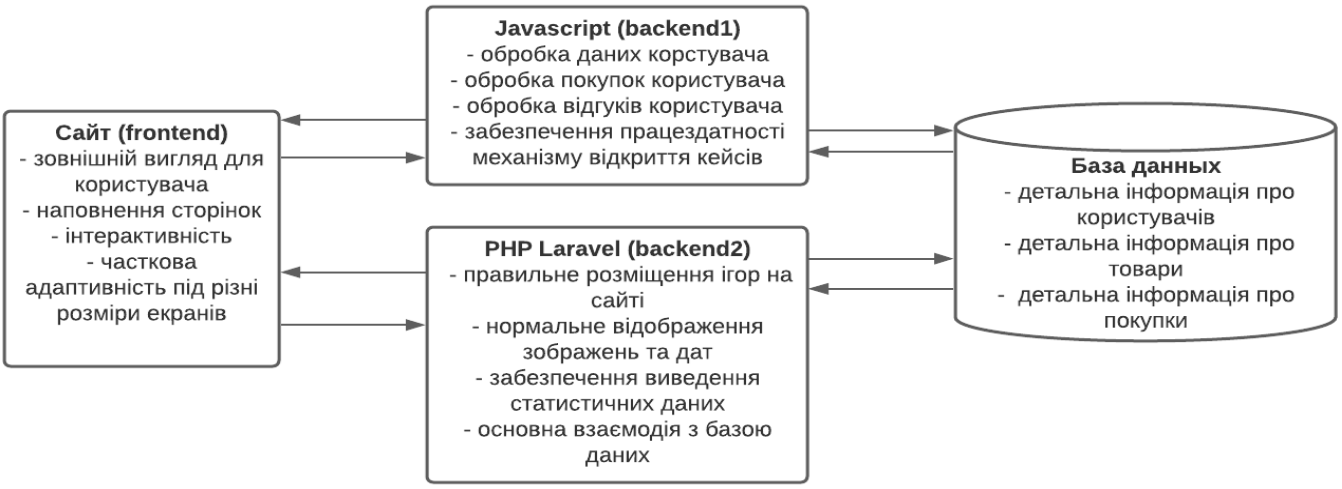

Рисунок 2.1.1 – Загальна схема роботи веб-системи

2.2 Загальний опис сайту

На сайті користувачу доступні наступні сторінки: головна, інформаційна сторінка для гри, реєстрація та вхід, профіль користувача, інформаційні вкладки в меню головної сторінки, такі як зворотній зв’язок(контакти), гарантії та інформація про проект. Схема навігації по сайту зображена на рисунку 2.2.1.

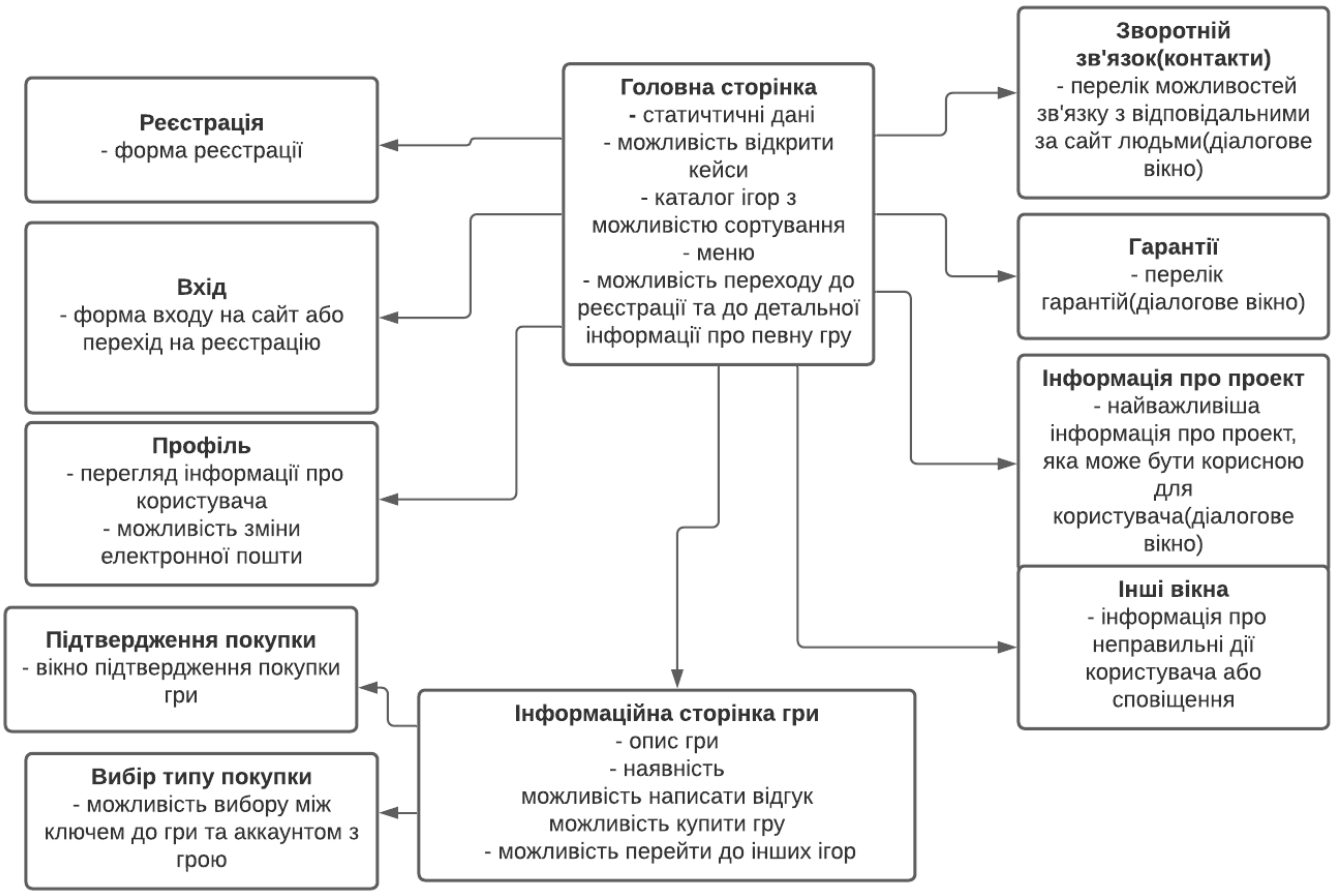

Рисунок 2.2.1 – Навігація по сайту та інформаційні вікна

Головна сторінка містить різні статистичні дані, такі як перелік ігор, які нещодавно виграли інші користувачі сайту, загальна кількість відкритих кейсів(наборів ігор з яких можна виграти тільки одну), загальну кількість користувачів на сайті, а також кількість користувачів, що користуються послугами сайту на даний момент. Потім йде частина з відкриттям кейсів, для яких можна обрати ключі, залежно від яких буде підбиратися відповідний набір ігор. Кейси також можна відкривати в демо-режимі, щоб пересвідчитись в чесності заданої на сайті випадковості. При використанні демо-режиму, гроші не спишуться з аккаунту, але й гру користувач не отримає. Далі йде каталог ігор, які можна просто купити. Там реалізована можливість сортування по таким параметрам, як платформи, де були перший раз куплені ігри та тип ігор, яких доступно два. Це ліцензійний ключ, який необхідний для отримання ліцензійної гри або аккаунт з

грою, або декількома іграми. Також основна інформація про ігри, така як назва, ціна та наявність виведена на передпоказ в комірках каталогу. Якщо ж ігор недостатньо на поточній сторінці, можна переглянути ще.

Інформаційна сторінка гри містить її опис, картинку, ціну та можливість написати коментар. Ці джерела інформації дадуть користувачу або упевнитися у своєму виборі, або передумати купляти гру і не писати негативний відгук. Якщо користувач захоче купити гру, він зможе обрати ключ або аккаунт з грою, якщо вони є в наявності, після чого йому потрібно буде підтвердити покупку і з його балансу буде списана вартість цієї гри.

Реєстрація на сайті та вхід створені максимально просто. Треба всього ввести пароль, пошту та ім’я. Поля перевіряються. Якщо користувач вже зареєстрований і помилково натиснув на кнопку реєстрації, він може одразу перейти до входу на сайт і навпаки, що є досить зручною функцією.

Профіль користувача зберігає його покупки, баланс та дані. Баланс можна поповнити, зайшовши в профіль. Також можна поставити для себе картинку, яка буде відображатися в коментарях та профілі. При необхідності, можна змінити пошту.

Інформаційні вікна, які є в меню надають користувачу можливість ознайомитися з правилами сайту, з проектом та розробниками, а при виникненні запитань, він може звернутися до контактної інформації і скориставшись одним із запропонованих способів зв’язку, отримати необхідну інформацію.

3 ОПИС ЗАДАЧІ ІНДИВІДУАЛЬНОГО ВНЕСКУ

3.1 RESTful API

Основною індивідуальною задачею було розробити програмний інтерфейс для сайту 4keys. Потрібно було забезпечити сайт такими запитами, через які можна отримати інформацію про один, декілька, за певним принципом, або всі об’єкти бази даних та їхні зв’язки з іншими таблицями. Також необхідно було створити функціонал для панелі адміністратора, тобто, запити, що дозволяють додавати, оновлювати та видаляти інформацію з бази даних.

Для відповідей API програми на запити сайту було вибрано текстовий формат JSON. Він базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Він використовується переважно для передачі структурованої інформації через мережу, а саме за допомогою процесу серіалізації.

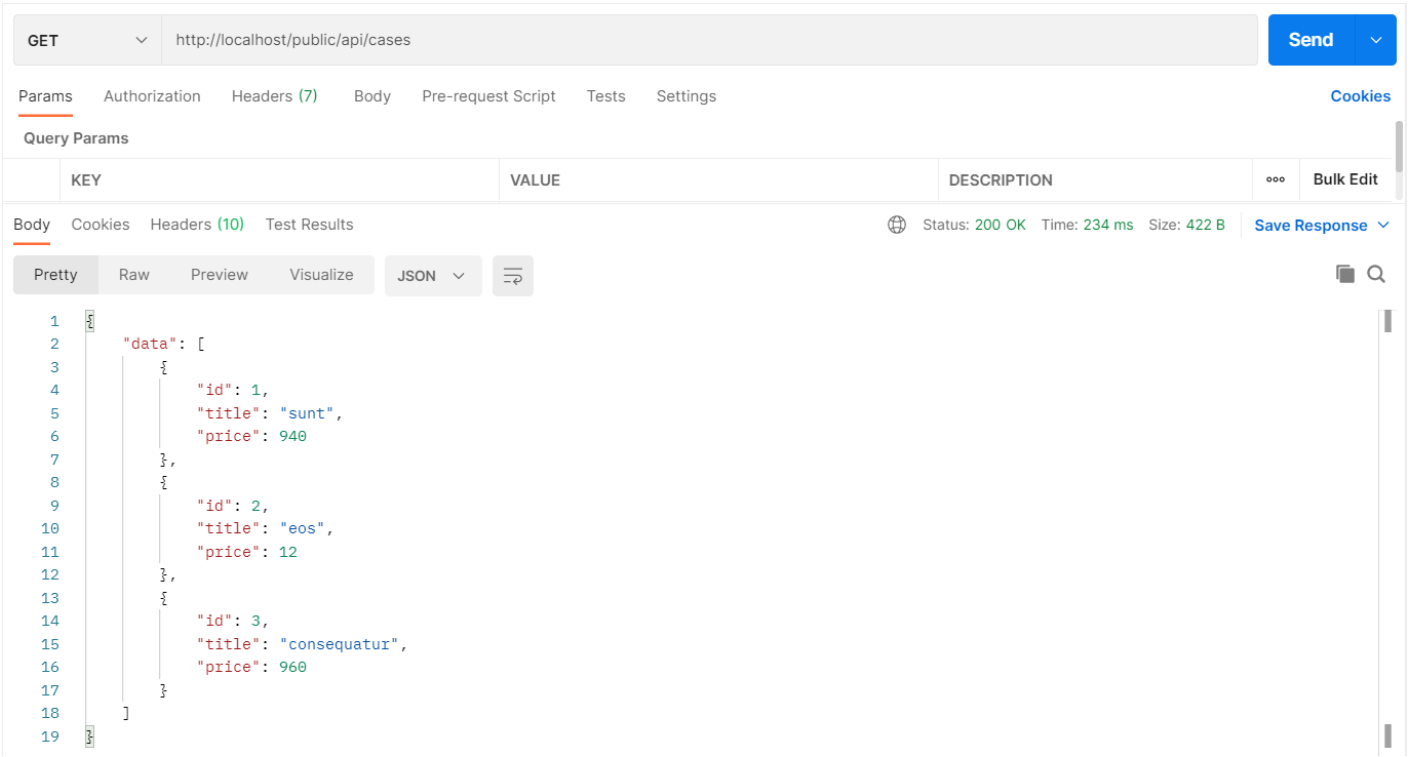

Рисунок 3.1 – json у відповідь на GET запити для отримання інформації про усі кейси

4 ОПИС ЗАСОБІВ ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ВНЕСКУ

4.1 Laravel & PHP

Для створення програмного інтерфейсу сайту використовувався PHP-фреймворк Laravel. Це програмне забезпечення було встановлене на Open Server, локальний веб-сервер для Windows і програмне середовище, в яке вмонтовано мову програмування PHP, phpMyAdmin та безліч інших необхідних програм для веб програмування.

Хоча Laravel призначений для розробки веб-додатків відповідно до шаблону model–view–controller (MVC), можливо також використовувати його для розробки програмного інтерфейсу. В такому випадку view набуде значення шаблону JSON, що ми відсилаємо на «дружній» нам веб-застосунок.

Laravel був вибраний невипадково. Серед його переваг можна виділити:

- широке ком’юніті;
- хороший рівень безпеки баз даних;
- легкість розширення функціоналу;
- зручність налаштування та користування;
- наявність постійних оновлень;
- доступна та детальна документація

Усі перелічені можливості цього сервісу є лише поверхневим описом фреймворку. Конкретно кажучи, цей фреймворк дозволить якісно та безпечно отримати запити через рути (routes) та обробити відповіді через контролери. Взаємодія з базою даних відбувається в основному за рахунок моделей. Також для тестування роботи api знадобилося створення фабрик – механізм штучного заповнення бази даних.

Один з небагатьох мінусів Laravel є документація, яка повільно оновлюється. Через це потрібно спеціально встановлювати чуть старіші версії, аби не було потреби вивчати найновіший функціонал методом «тику» та дослідженням системних файлів фреймворку.

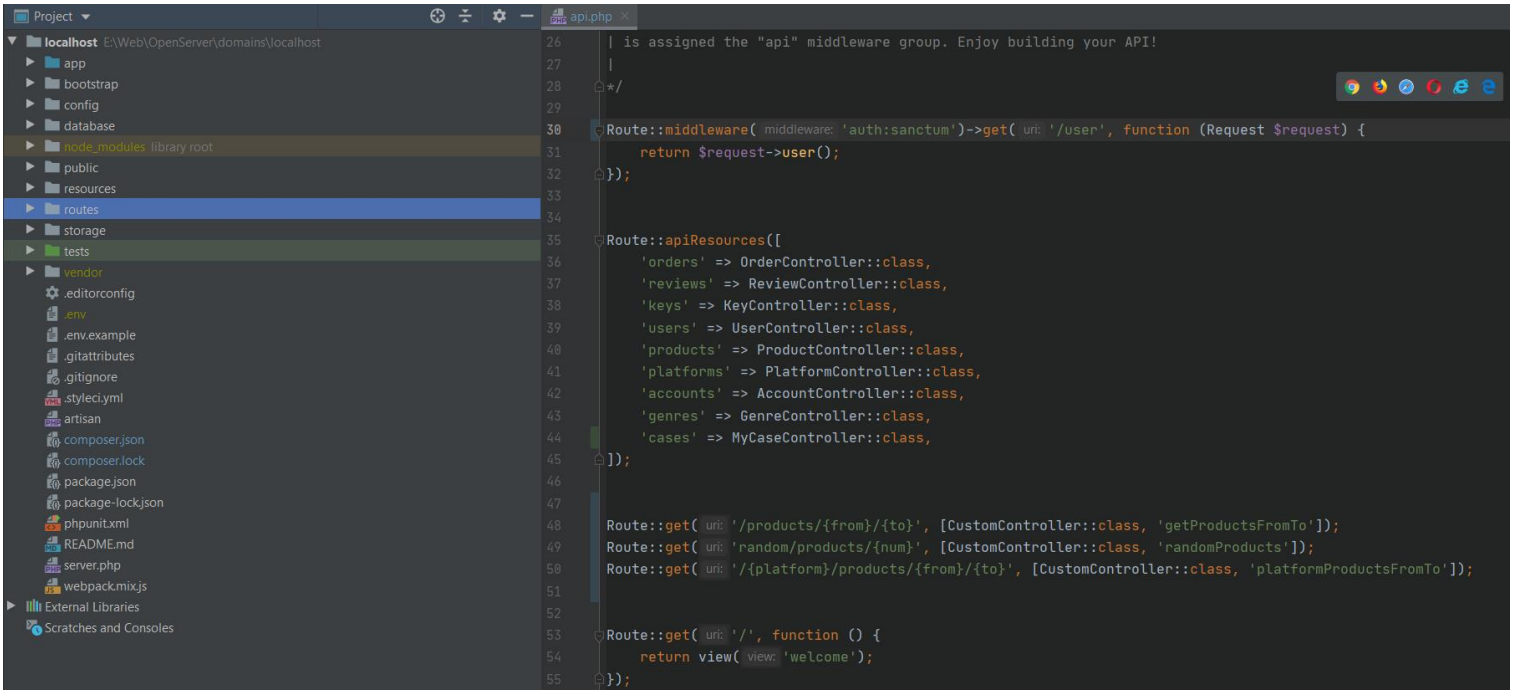

Рисунок 4.1.1 – каталоги проекту на Laravel(зліва), усі можливі рути (справа)

4.2 PhpStorm

PhpStorm - це інтегроване середовище розробки для PHP HTML і CSS з можливостями аналізу коду на льоту, запобігання помилок у сирцевому коді і автоматизованими засобами рефакторинга для PHP. Автодоповнення коду в PhpStorm підтримує специфікацію PHP 7.0/7.4, включаючи генератори, простори імен, замикання, типажі і синтаксис коротких масивів. Хоча в цьому середовищі розробки існує повноцінний SQL-редактор з можливістю редагування отриманих результатів запитів, було використано для роботи з базою даних саме phpMyAdmin. Це середовище можна побачити на рисунку 4.1.1

4.3 Postman

Postman – це набір інструментів для тестування API. За допомогою нього можна надсилати необхідні нам GET, POST, PUT, DELETE запити і перевіряти коректність відповіді серверу. На виході можна отримати файли в різних текстових форматах таких як JSON, XML, HTML.

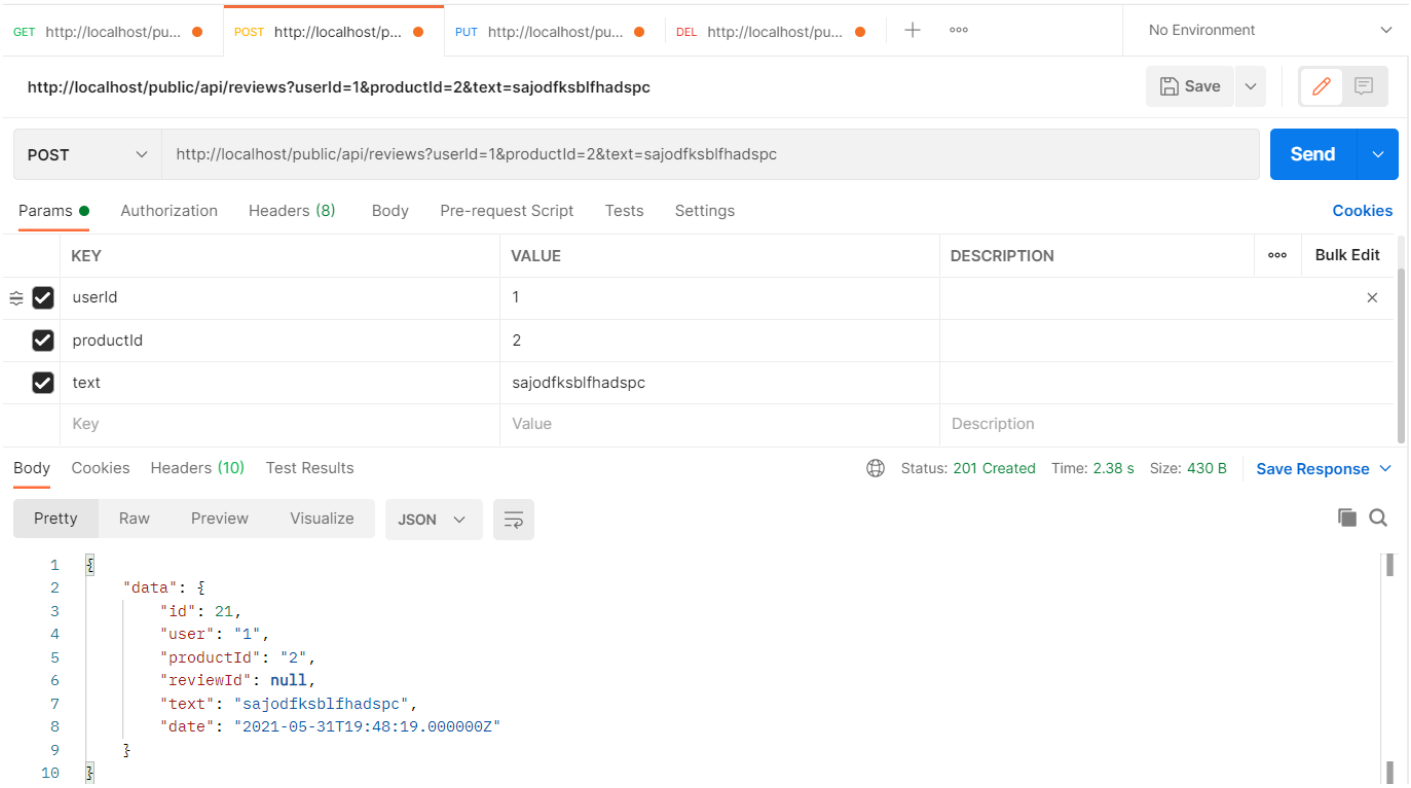

Рисунок 4.3.1 – основний функціонал програми

На рисунку 4.3.1 можна побачити надсилання запиту методом POST на локальний сервер, щодо створення рядку в таблиці reviews (коментарі). Стовпці – це ключі (KEY) в них ми вставляємо певні дані (VALUE). Заповнюючи ці необхідні дані, можна помітити, що посилання, в на тому рівні, де назва методу, теж відповідно змінюється.

Також, окрім різних текстових форматів, можна вибрати зручний вид подання відповідної інформації. Для виведення JSON на рисунку 4.3.1 використано подання “Pretty”, що додає до табуляцію, яка відсутня у сирому “Raw” реальному вигляді JSON. Вибравши вид “Preview”, буде видано у вигляді веб-браузера, при цьому необхідно, аби застосунок відповідав HTML кодом на запити.

5 ОПИС ПРОЦЕСУ ВИКОНАННЯ ТА РЕЗУЛЬТАТІВ ІНДИВІДУАЛЬНОГО ВНЕСКУ

5.1 Міграції та фабрики

Перш ніж переходити до створення програмного інтерфейсу сайту, потрібно було створити та заповнити бази даних. За для безпеки основного сайту, всі тести над програмним забезпеченням проводилися на локальному сайті, тож потрібно було створити та неодноразово перероблювати локальну базу даних, заповнювати даними.

Для реалізації вище описаної задачі були використані міграції та фабрики. Спершу були створенні моделі та міграції за допомогою консолі. Далі описані усі стовбці таблиці конкретної моделі. При цьому таблиць відповідно моделей і міграцій всього 10.

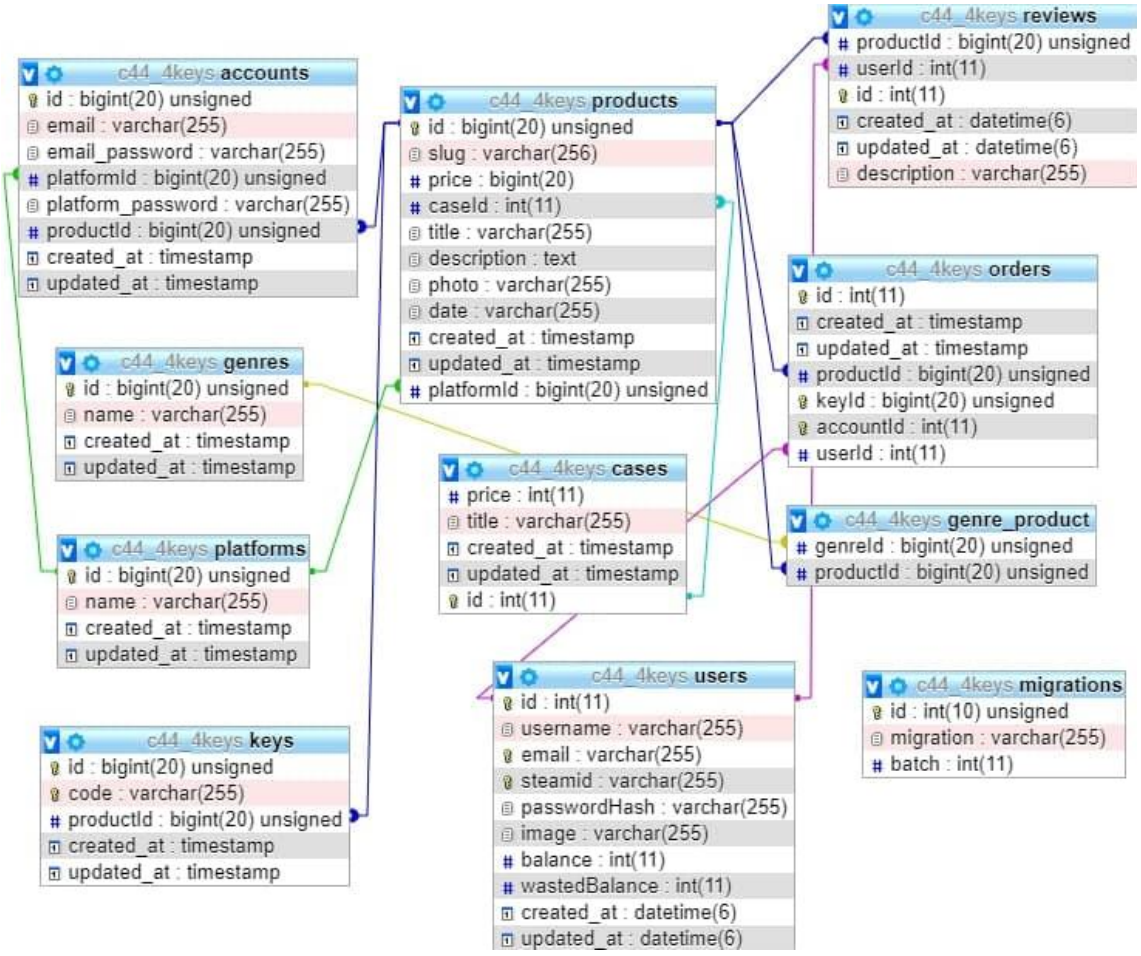

Рисунок 5.1.1 – усі таблиці бази даних

На рисунку 5.1.1 можна помітити таблицю migrations. Вона створена для перенесення інших таблиць, що знаходяться в проекті на локальному сервері. Тож в задачу входило створити файли міграції відповідно до таблиць. Найчастіше серед усіх консольних команд використовувалася ‘php artisan migrate:fresh’ – видаляє стару та встановлює нову базу даних на основі міграцій на локальному сервері.

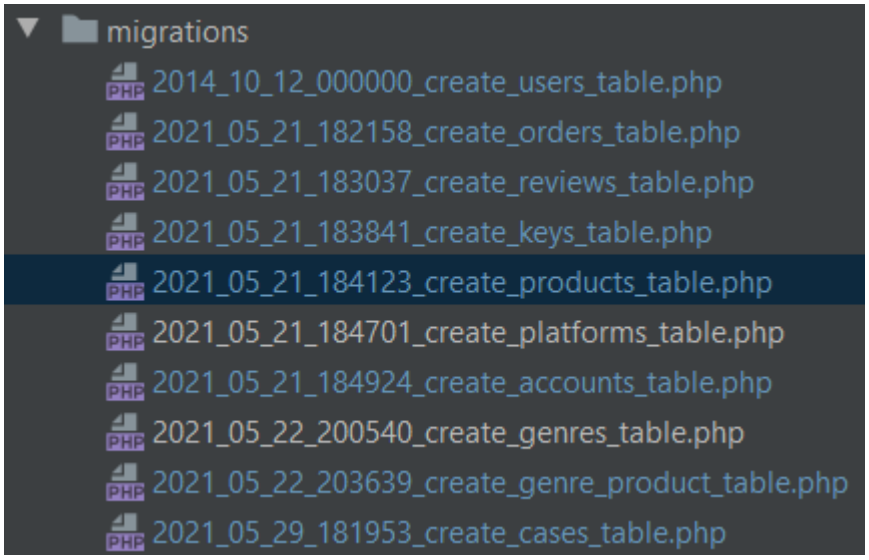

Рисунок 5.1.2 – усі міграції в проекті

Як раніше було сказано, для заповнення баз даних використовуються фабрики, відповідні рядки заповнюються випадковими даними через фейкер або звязуться з уже існуючими рядками інших таблиць, аби дотримуватися логіки проекту. За допомогою консолі були створені заводи та сіддер, що запускає роботу цих заводів також у сідері прописане встановлення зв’язків багато до багатьох у звязній таблиці. Запускаємо сідер через команду в консолі ‘php artisan db:seed’. За допомогою цієї системи була початково заповнена база даних серверу, наприкла, ключами, який в базі налічюється 2000 штук (див. рисунок 5.1.3). Подібні процедури були зроблені з іншими таблицями, що були імпортовані з локальної бази даних на сервер за допомогою панелі керування phpMyAdmin.

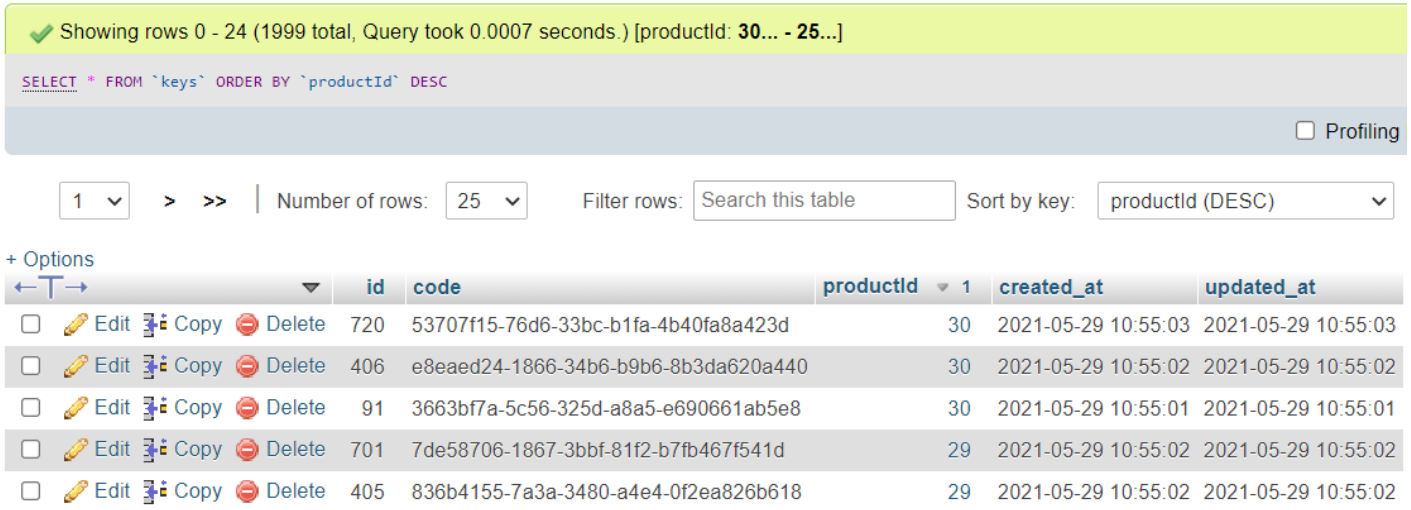

Рисунок 5.1.3 – 2000 різних ключів для ігор в базі даних

5.2 Зв’язок між таблицями

Наступним етапом стало встановлення взаємозв’язків, між новоствореними базами даних. Реалізовувалося це все через моделі у фреймворці Laravel. Зв’язок встановлювався згідно вище згаданому рисунку 5.1.1. Для зручного використання фронтендом цього програмного інтерфейсу були змінені деякі системні файли та був змінений класичний підхід задання параметрів.

Наприклад, візьмемо до уваги таблицю products. Вона пов’язана з таблицями keys, accounts та reviews зв’язком один до багатьох, тобто у одного продукту може бути багато ключів або коментарів. Багато до одного описаний через відношення products до platforms. Для зв’язку багато до багатьох потребувалося створити зв’язну таблицю genre_product, що дозволяє продукту мати декілька жанрів, і на оборот, жанрам – декілька продуктів. Зв’язок один до одного можна помітити між таблицями orders та keys або accounts, тобто в ордері про купівлі може бути тільки унікальний ключ або акаунт продукту. Взаємозв’язки також будувалися на перед тим продуману бізнес логіку.

Крім того, в моделях були прописані поля даних на основі полів у відповідних таблицях, що в подальшому знадобилося в створені нових моделей через запити методом POST.

5.3 Безпека

Аби не можна було отримати дані через програмний інтерфейс третіми особами, було прийняте рішення встановити пароль у вигляді хедера на проміжне програмне забезпечення (middleware). Основна його ціль у фреймворці – це фільтрація та первина обробка HTTP запитів. Тож, послугами API може користуватися.

Також важливо було донести сайту певні помилки у зрозумілому для нього вигляді. Некоректні запити вертають JSON відповідь та код помилки, при цьому для кожного – своя.

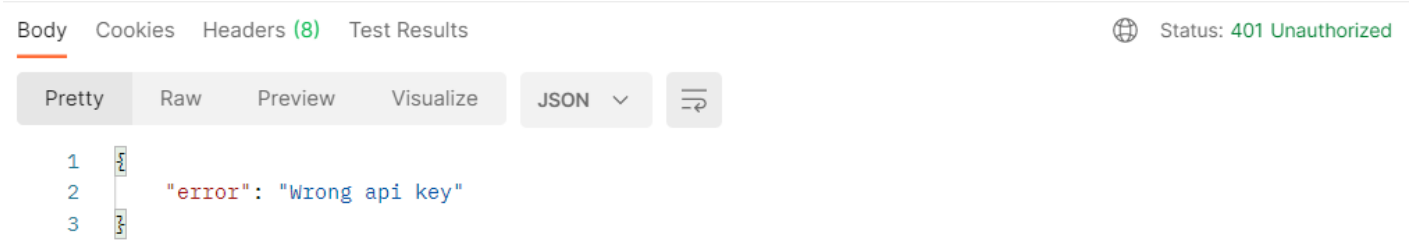

Рисунок 5.3.1 – не введений чи неправильно ведений api ключ. Помилка 401

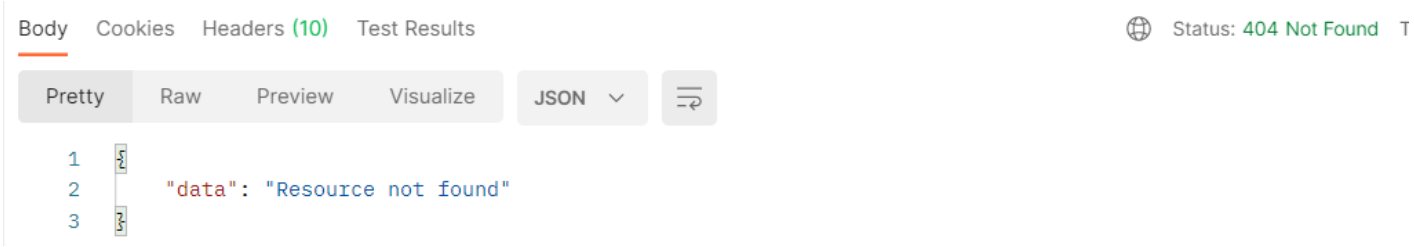

Рисунок 5.3.2 – продукт, що сайт пробує знайти, не існує. Помилка 404

Для створення й оновлення даних використовувалися механізми валідації в Laravel. Були створені класи типу Request відповідно до типу моделі для встановлення правил запиту для методів POST та PUT. Загалом поля отримували такі правила, які важливі для звичайної та бізнес логіки:

- Деякі поля являються головними для рядку, тож вони повинні бути заповненні;
- Деякі поля мають містити унікальні дані;

- Деякі поля з числами не можуть бути від’ємними;
- Деякі поля посилаються на інші, тож вони мають обов’язково існувати
- Поля не можуть посилатися на вже проданий товар;
- Та інші…

При недотримані правил запитів програмний інтерфейс відповідає відповідно.

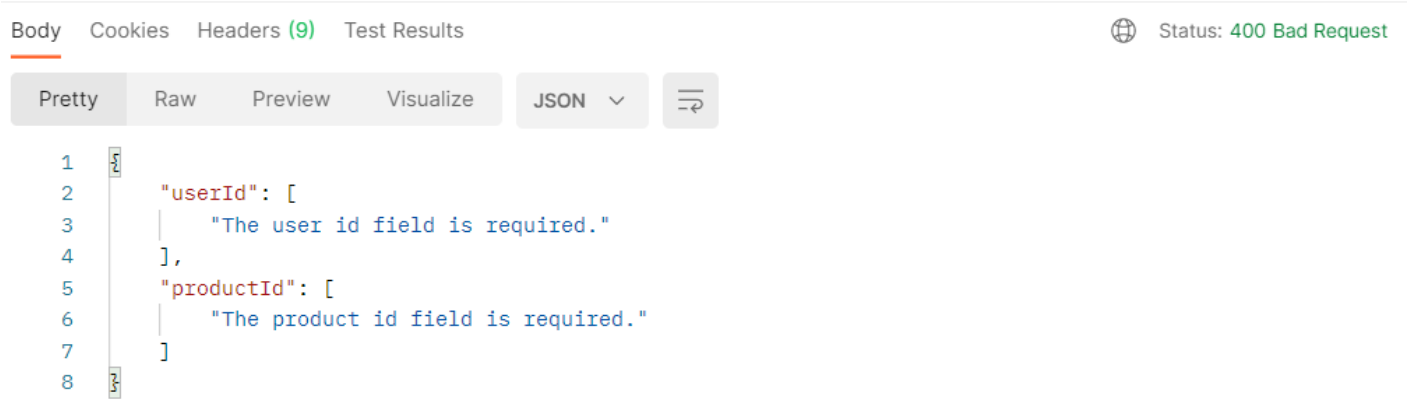

Рисунок 5.3.3 – неправильно введений запит методом POST. Помилка 400

5.4 Запити і відповіді

Усі можливі рути, які відповідають запитам, прописані в одному файлі. Кожен з них посилається на індивідуальний ресурсний арі контролер, тобто для таблиці products існує свій контролер в якому прописані стандартні операції : index, store, show, update, delete. Для наочності, продемонструємо взаємодію з products через таблицю 5.4.1,

Метод	Запит	Операція	Призначення
GET	/products	index	Виведення усіх рядків таблиці
POST	/products{query}	store	Створення нового рядка у таблиці
GET	/products/{index}	show	Виведення рядка за індексом
PUT	/products/{index}{query}	update	Оновлення рядка за індексом
DELETE	/products/{index}	delete	Видалення рядка за індексом

Таблиця 5.4.1 – класичні запити для products

Ці запити актуальні й для інших таблиць. Також слід згадати й інші, не ресурсні запити, які були винесені в окремий контролер. До них входять виведення : випадкових продуктів з визначеною кількістю, продуктів у певній кількості, порядку та виведення певної кількості продуктів, що належать єдиній платформі.

Важливо було видавати інформацію з таблиці в практичному вигляді. За допомогою класів типу Resource можна налаштувати вигляд JSON, а саме добавити необхідні дані про об’єкт посилаючись на інші таблиці, або не виводити певні стовпці взагалі. Тож, наприклад, для моделі Product відповідь складали усі можливі дані з її таблиці, більше того, користуючись зовнішніми ключами до класу ResourceProduct були добавлені його жанри, платформа, кейс та доступний товар. Доступний товар – це кількість ключів та акаунтів, що не знаходяться в ордерах, тобто не куплені.


```
{
    "data": {
        "id": 1,
        "slug": "pfannerstill",
        "price": 4764,
        "title": "voluptatem",
        "description": "Non optio non magnam tenetur consequuntur quibusdam. Voluptas et et ipsa itaque
        "photo": "https://drive.google.com/file/d/1juZJVpc-6URWc3Di86qCGhZkxEBMS5_n/view?usp=sharing",
        "date": "28.10.2003",
        "genres": [
            {
                "id": 1,
                "name": "johns"
            },
            {
                "id": 2,
                "name": "brown"
            },
            {
                "id": 4,
                "name": "baumbach"
            }
        ],
```


Рисунок 5.4.1 – не повний JSON продукту, у якому дані про його жанри й не тільки

6 ОПИС РЕЗУЛЬТАТІВ ЗАГАЛЬНОЇ ВЕБ-СИСТЕМИ

6.1 Головна сторінка

При переході на сайт користувач потрапляє на головну сторінку, на якій він може перейти до реєстрації, отримати важливу інформацію в меню, переглянути ігри, що отримали інші користувачі, статистичну інформацію, отримати за певну плату гру з набору, запропонованого за ці гроші та переглянути каталог ігор з можливістю покупки.

Перейти до реєстрації можна, натиснувши відповідну кнопку у правому верхньому куті сайту. Зовнішній вигляд частини меню, що відповідає за аутентифікацію користувача представлений на рисунку 6.1.1.

Рисунок 6.1.1 – меню реєстрації сайту

На випадок, якщо користувач не помітить цього меню, або не захоче реєструватися, веб-система запропонує йому це зробити, коли він захоче зробити на сайті певні дії, такі як покупка гри або додавання коментаря.

Лівіше від цього меню є інформаційне меню сайту, де можна прочитати про сайт і перейти на головну сторінку. Вигляд цього меню зображений на рисунку 6.1.2.

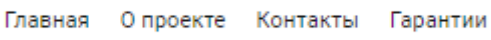

Рисунок 6.1.2 – інформаційне меню сайту

Під меню перше, що можна побачити – це перелік товарів, нещодавно куплених користувачами сайту, а також інформацію про загальну кількість покупок випадкових ігор та користувачів загалом та онлайн. Це все має показати користувачу, що сайт чесний

і має велику кількість постійних користувачів. Вигляд цієї частини головної сторінки зображений на рисунку 6.1.3.

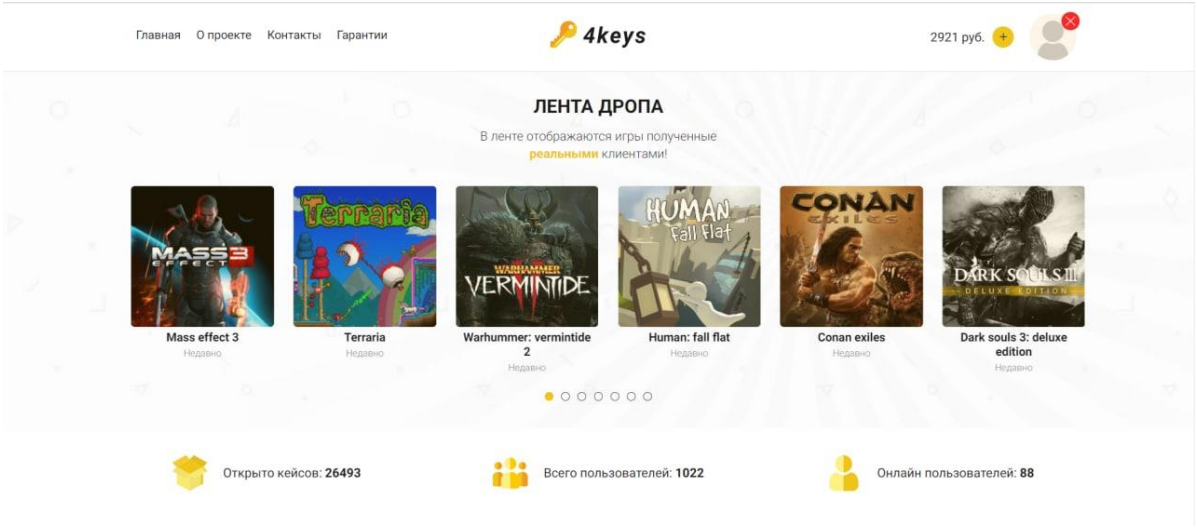

Рисунок 6.1.3 – статистична частина головної сторінки

Далі йде частина сайту, де користувач може отримати випадкову гру з набору. Є 2 варіанти роботи цієї функції сайту. Перший - це демо-режим, що не списує з аккаунту кошти, але і користувач товар не отримує. Це зроблено перш за все для оцінки шансів та пересвідчення в чесності функції. Щоб скористатися даною функцією, треба обрати ключ із запропонованих і натиснути кнопку запуску випадкового генератора. Отримана гра з’явиться після закінчення роботи функції у вигляді діалогового вікна. Зараз працює лише бета-версія. Зовнішній вигляд цієї частини головної сторінки представлений на рисунку 6.1.4

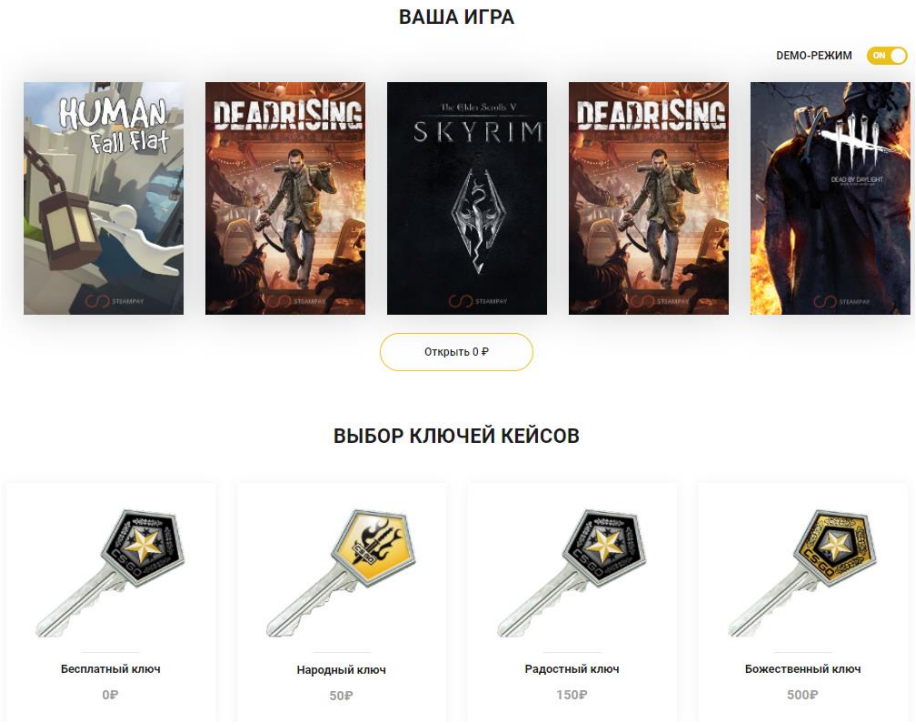

Рисунок 6.1.4 – інтерфейс відкриття кейсів

Наступна частина – це каталог ігор. Ігри можна сортувати по офіційній платформі, на якій вони раніше були куплені та за наявністю ключа або аккаунту. Якщо не одна з ігор не відповідає інтересам користувача, він може загрузити інші ігри з каталогу, натиснувши на відповідну кнопку. Для кожної гри є передпоказ, що містить картинку з назвою гри, ціну та наявність ключів та акаунтів для цієї гри. При наведенні курсору на гру, відобразиться її назва. Це зроблено на випадок, якщо картинка недостатньо чітко показує назву. Передпокази ігор до наведення курсору і після зображені на рисунках 6.1.5 та 6.1.6 відповідно.

Рисунок 6.1.5 – передпоказ гри

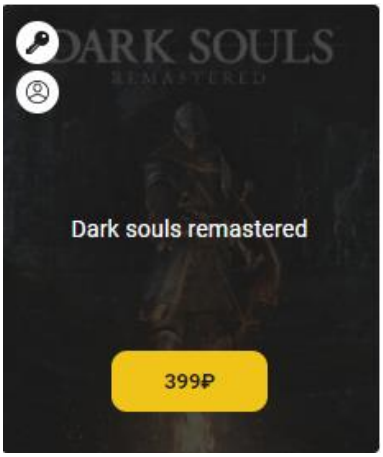

Рисунок 6.1.6 – результат наведення курсору на блок передпоказу

Загальний вигляд каталогу представлений на рисунку 6.1.7, а на рисунку 6.1.8 зображена кнопка завантаження додаткових ігор на екран.

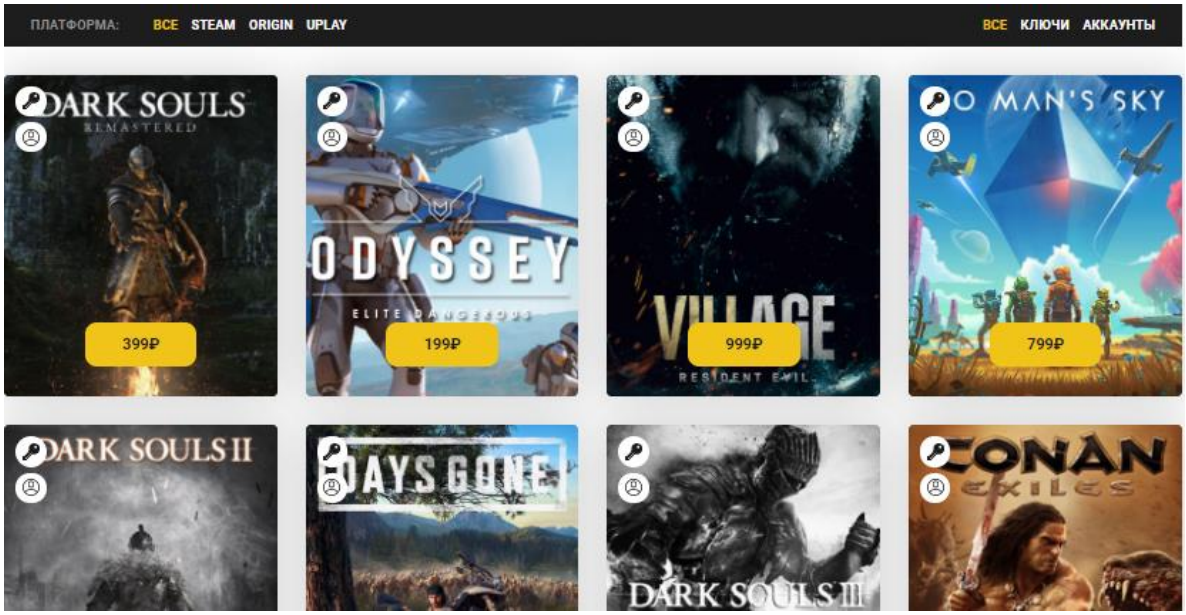

Рисунок 6.1.7 – каталог ігор

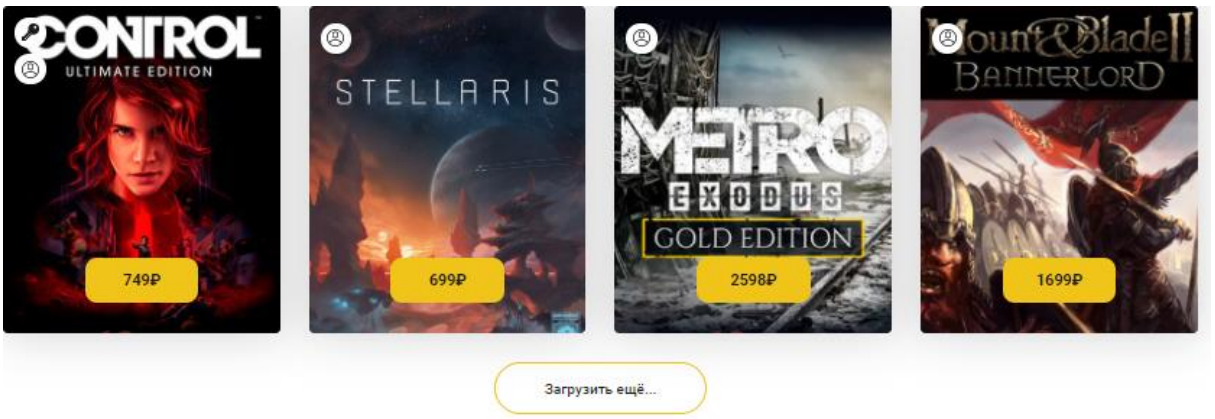

Рисунок 6.1.8 – кінець каталогу та можливість завантаження додаткових ігор

Останньою частиною головної сторінки є нижнє меню, яке повторює верхнє, але в іншому стилі. Воно створене для того, щоб користувачу не було потрібно вертатись назад для доступу до меню, а також на випадок, якщо користувач не помітив одразу верхнє меню, то зміг скористатись нижнім. Вигляд нижнього меню зображений на рисунку 6.1.9.

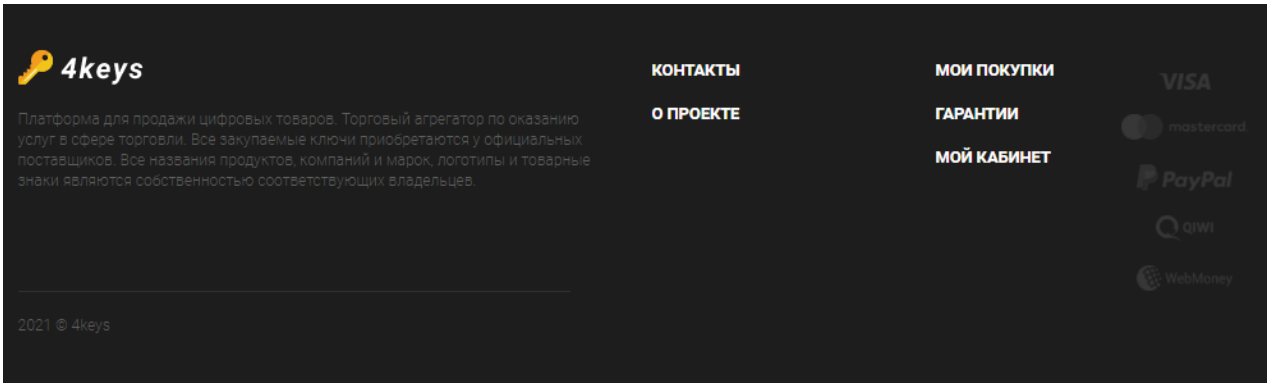

Рисунок 6.1.9 – нижнє меню головної сторінки

Таким чином, головна сторінка є найбільш наповненою сторінкою сайту, що зв’язує його компоненти та показує користувачу всі можливості веб-системи.

6.2 Інформаційні сторінки та діалогові вікна

На сайті наявні інформаційні діалогові вікна та сторінки, що містять інформацію про проект, контактну інформацію, гарантії та інформацію про ігри. З діалогових вікон можна вийти, натиснувши кнопку-хрестик в правому верхньому куті або натиснувши в будь-яку точку поза межами вікна. З інформаційних сторінок вийти можна через меню.

При виникненні питань стосовно сайту або певних неприємностей, з меню можна перейти до контактної інформації. Ці дані представлені у діалоговому вікні, що зображене на рисунку 6.2.1.

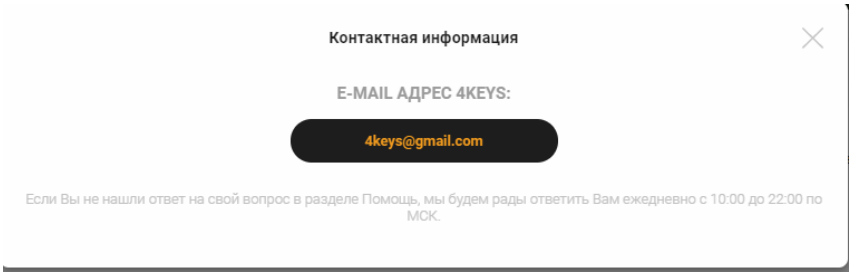

Рисунок 6.2.1 – Діалогове вікно контактної інформації

Для отримання детальної інформації про сайт, користувач може ознайомитись з інформацією про проект та гарантії, що міститься в діалогових вікнах, які з’являються при натисканні на відповідні пункти меню. В першому вікні розміщена інформація про мету сайту та напрям розробки, а у другому – правила сайту. Діалогові вікна про сайт та гарантії зображені на рисунках 6.2.2 та 6.2.3 відповідно.

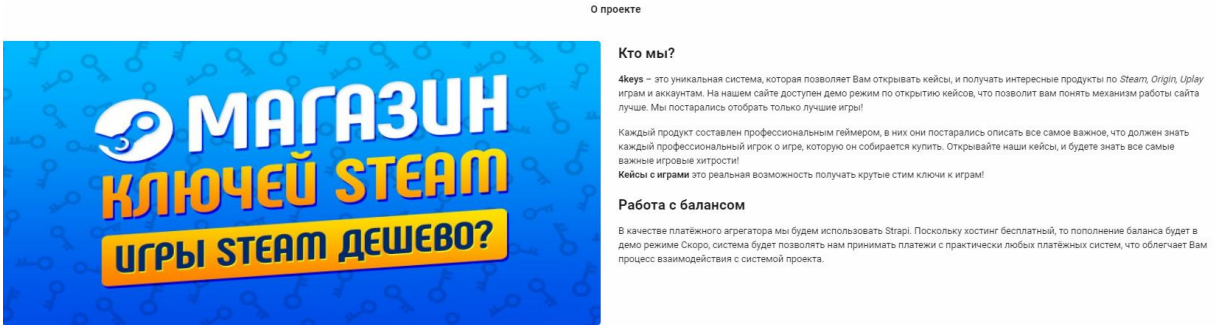

Рисунок 6.2.2 – Діалогове вікно з інформацією про проект

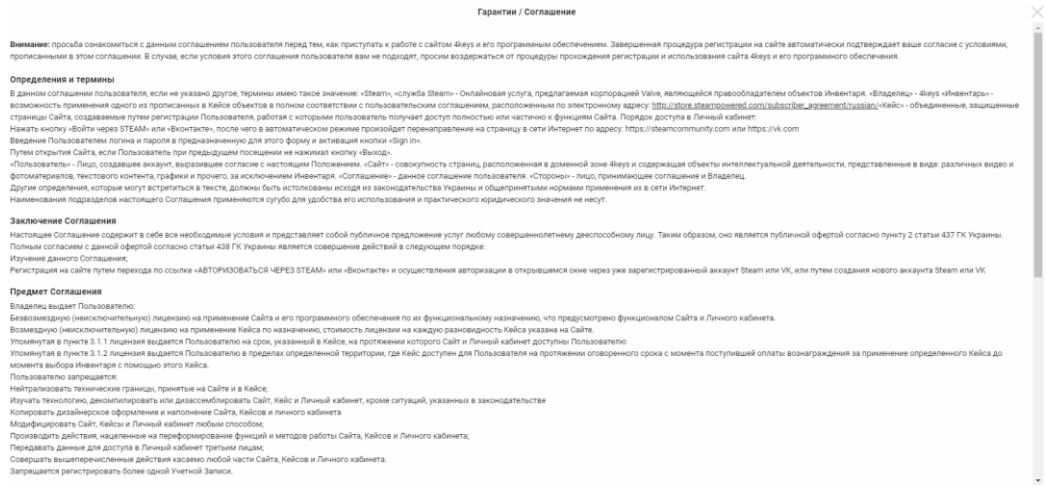

Рисунок 6.2.3 – Діалогове вікно правил сайту

Натиснувши на гру в каталозі, можна перейти до інформаційної сторінки цієї гри. Там крім тієї інформації, що видно в каталозі є ще дата виходу, жанр, опис, коментарі, інші ігри. Вигляд цієї сторінки та поле коментарів зображено на рисунках 6.2.4 та 6.2.5 відповідно. До гри можна залишити лише один коментар.

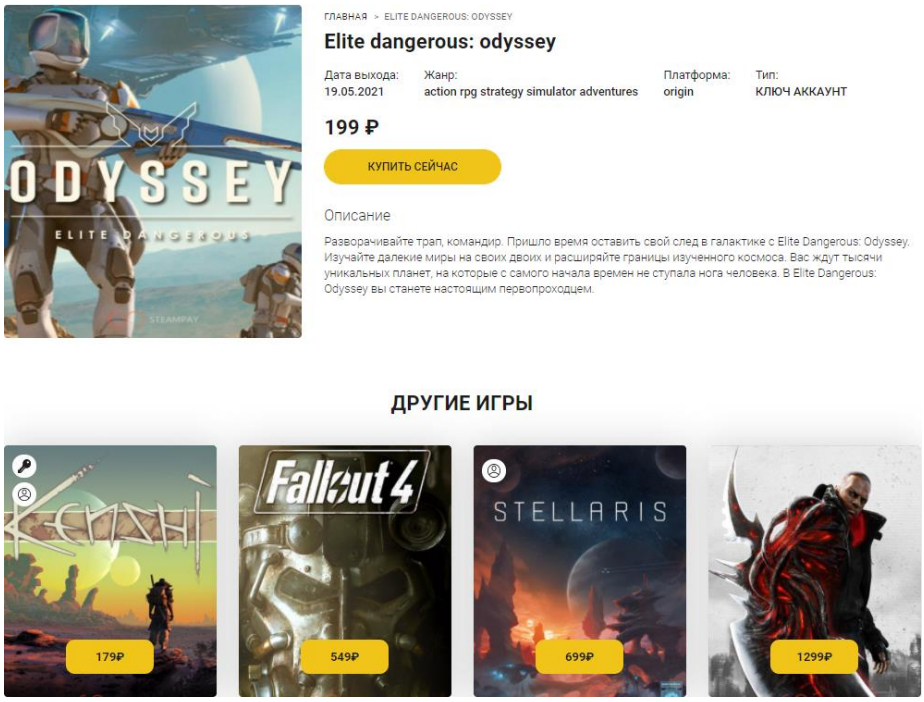

Рисунок 6.2.4 – Вигляд інформаційної сторінки про гру

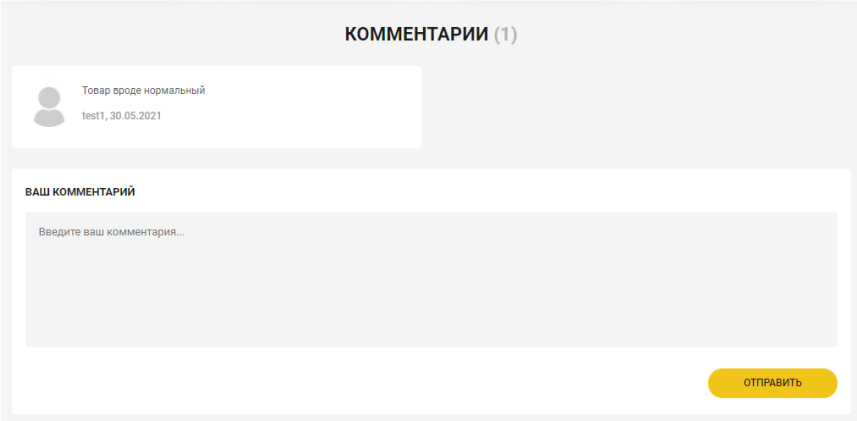

Рисунок 6.2.5 – Поле для коментарів

Покупка гри доступна тільки для зареєстрованих користувачів. При натисканні на кнопку покупки, можна вибрати ключ або аккаунт. Вікно вибору зображене на рисунку 6.2.6.

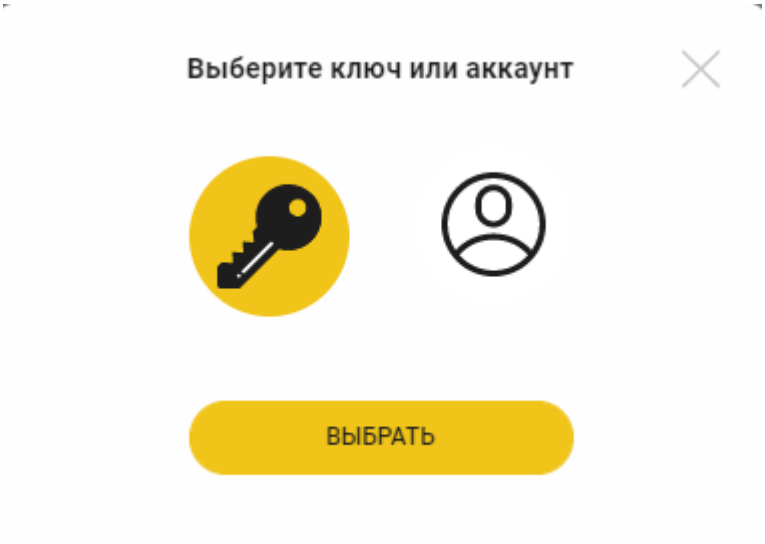

Рисунок 6.2.6 – Вікно вибору варіанту покупки

В цьому інформаційному вікні також є велика кількість повідомлень, наведення зображень яких є недоцільним. Повідомлення можуть бути про необхідність реєстрації перед покупкою, недостатню кількість коштів на акаунті, підтвердження покупки, спробу заповнити коментар, не купивши товар, помилкове заповнення поля коментарю, повідомлення про відміну та успішність покупки.

6.3 Аккаунт

Як вже було показано, на сайті присутня авторизація користувача. Можна зареєструватись, увійти за допомогою сайту або аккаунту платформи steam. Форми реєстрації та входу взаємопов’язані. Це означає, що з однієї можна перейти на іншу при помилковому виборі. Ці форми зображені на рисунках 6.3.1 та 6.3.2 відповідно.

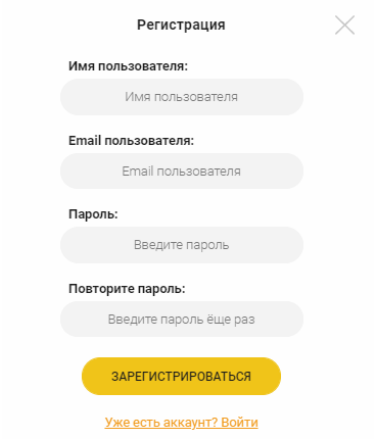

Рисунок 6.3.1 – Вигляд форми реєстрації

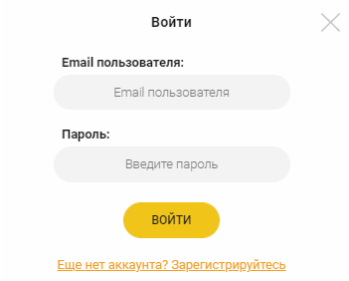

Рисунок 6.3.2 – Вигляд форми входу

Для входу за допомогою аккаунту steam відкривається нове вікно, де потрібно підтвердити вхід.

Після успішної реєстрації можна перейти в профіль користувача, натиснувши на картинку, що з’явилась замість кнопок реєстрації та входу або на відповідні пункти нижнього меню, а також поповнити баланс, натиснувши на кнопку-плюс біля балансу. Вигляд меню для користувача показаний на рисунку 6.3.3.

Рисунок 6.3.3 – Меню користувача після реєстрації

В кабінеті користувача є його особисті дані, такі як ім’я та пошта, яку він може змінити, баланс, покупки та зображення(якщо вхід був здійснений з платформи steam). Вигляд цієї сторінки зображений на рисунку 6.3.4.

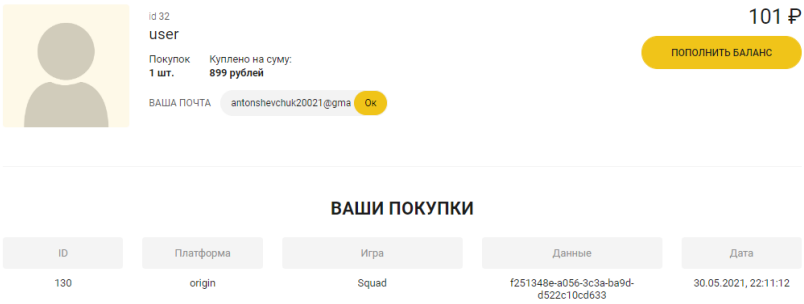

Рисунок 6.3.4 – Особистий кабінет користувача

Кнопка поповнення балансу виконує ту саму функцію, що й кнопка-плюс в верхньому меню. Для демонстрації, оплата не працює і можна поповнити свій баланс просто вказавши суму та натиснувши кнопку-підтвердження. Форма поповнення балансу зображена на рисунку 6.3.5.

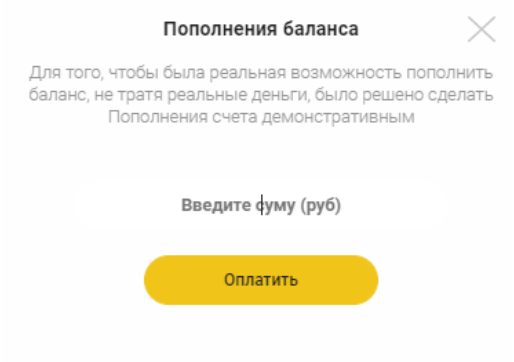

Рисунок 6.3.6 – Форма поповнення балансу

Профіль користувача містить тільки найнеобхідніші дані та не перевантажений непотрібними даними, що значно спрощує користування сайтом.

ВИСНОВКИ

В ході виконання курсової роботи було отримано інформацію про принципи роботи інтернет-магазинів, запити в них та обробку інформації. Була створена власна база даних для проекту, яка через відсутність точної моделі проекту, змінювалась багато разів. Були отримані додаткові знання про можливості RESTful api, javascript та php. Отримано досвід роботи з немалими базами даних в phpMyAdmin. Також було знайдено спосіб швидкої генерації записів в базу даних за допомогою додаткового програмного забезпечення. Був нагаданий синтаксис mysql, який знадобився для цієї роботи, але не в повній мірі, через те, що деякий функціонал, як, наприклад, перевірка значень, був перенесений на сервер. Також досить невдало була розподілена робота команди, через що хтось один робив основну частину роботи, а інші тільки допомагали. Певні навички були отримані і в пошуку помилок на сайті, що дозволяло швидко знаходити ситуації, в яких сайт функціонує неправильно. Дуже корисним було написання веб-системи від початку до кінця, що дало змогу попрацювати з кожним етапом і спостерігати, як ці етапи впливають на проект в цілому.

Тим не менш, хороший інтернет-магазин потребує набагато більше затрат ніж цей проект, але все одно кінцевий вигляд сайту є досить непоганим і основний функціонал в тій чи іншій мірі реалізований. Повертаючись до бази даних сайту, можна також сказати, що вона потребувала більше затрат часу та точного плану розробки, щоб мати хорошу архітектуру, з якою не виникало би пустих комірок або яка хоча б мінімізувала б їх кількість. Також було використано небагато типів даних, що можливо і добре, але не так адаптивно. [8]

Загалом, сайт вийшов працездатним, що є найважливішим і має непоганий вигляд.

ЛІТЕРАТУРА

1. https://laravel.com/docs/8.x
2. https://metanit.com/sql/mysql/2.4.php
3. https://dev.mysql.com/doc/
4. https://stackoverflow.com/
5. Abeysinghe S. RESTful PHP Web Services / Samisa Abeysinghe., 2008. – 222 c. – (Kindle Edition).
6. Haafiz Waheed-ud-din Ahmad. Building RESTful Web Services with PHP 7 / Haafiz Waheed-ud-din Ahmad., 2017. – 244 c. – (1st edition).
7. Lorna Jane Mitchell. PHP Web Services: APIs for the Modern Web / Lorna Jane Mitchell., 2016. – 180 c. – (2nd Edition).
8. Stowe M. Undisturbed REST: a Guide to Designing the Perfect API / Michael Stowe., 2015. – 200 c. – (First-Look Edition).
9. https://www.toptal.com/laravel/restful-laravel-api-tutorial

Додаток

Життєвий цикл таблиці products, моделі Pr

2021_05_21_184123_create_products_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateProductsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('slug')->unique();
            $table->bigInteger('price');
            $table->string('title')->unique();
            $table->text('description');
            $table->string('photo');
            $table->bigInteger('caseId')->nullable();
            $table->bigInteger('platformId')->nullable();
            $table->string('date');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     * @return void
     */
```

```
    public function down()
    {
        Schema::dropIfExists('products');
    }
}
```

ProductFactory.php

```
<?php
namespace Database\Factories;

use App\Models\MyCase;
use App\Models\Platform;
use App\Models\Product;
use Illuminate\Database\Eloquent\Factories\Factory;

class ProductFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = Product::class;
    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'slug'=> $this->faker->unique()->domainWord,
            'price' => rand(100, 5000),
            'title' => $this->faker->unique()->word(),
            'description' => $this->faker->text(),
            'caseId' => MyCase::all()->random(),
            'photo' => 'https://drive.google.com/file/d/1juZJVpc-6URWc3Di86qCGhZkxEBMS5_n/view?usp=sharing',
            'platformId' => Platform::all()->random(),
            'date' => $this->faker->date('d.m.Y'),
```

```
        ];
    }
}
```

Product.php

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    protected $fillable = [
        'id', 'slug', 'price', 'title', 'description', 'photo', 'platformId', 'caseId',
        'date', 'created_at', 'updated_at'
    ];

    public function reviews()
    {
        return $this->hasMany(Review::class);
    }

    public function platform()
    {
        return $this->belongsTo(Platform::class, 'platformId');
    }
    public function accounts()
    {
        return $this->hasMany(Account::class );
    }
    public function case()
    {
        return $this->belongsTo(MyCase::class,'caseId' );
    }
    public function keys()
    {
```

```
        return $this->hasMany(Key::class);
    }
    public function genres()
    {
        return $this->belongsToMany(Genre::class);
    }
}
```

ProductController.php

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Http\Requests\ProductRequest;
use App\Http\Resources\ProductResource;
use App\Models\Genre;
use App\Models\Genre_Product;
use App\Models\Product;
use http\Env\Response;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Str;

class ProductController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Resources\Json\AnonymousResourceCollection
     */
    public function index()
    {
        return ProductResource::collection(Product::all());
    }
    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return ProductResource
```

```
 */
public function store(ProductRequest $request)
{

    $product = new Product($request->all());
    if($product)$product->id = Product::all()->last()->id + 1;
    $product->created_at = now()->timestamp;

    $product->genres()->attach($request->genres);
    $product->save();

    return new ProductResource($product);
}

/**
 * Display the specified resource.
 *
 * @param $slug
 * @return ProductResource
 */
public function show($slug)
{
    if (is_numeric($slug)) return new ProductResource(Product::all()->find($slug));
    return new ProductResource(Product::all()->where('slug', $slug)->first());
}
/**
 * Update the specified resource in storage.
 *
 * @param ProductRequest $request
 * @param Product $product
 * @return ProductResource
 */
public function update(ProductRequest $request, Product $product)
{
    $product->genres()->detach();
    $product->genres()->attach($request->genres);

    $product->update($request->validated());
```

```
        return new ProductResource($product);
    }
    /**
     * Remove the specified resource from storage.
     *
     * @param Product $product
     *                                                                                                  @return
\Illuminate\Contracts\Foundation\Application|\Illuminate\Contracts\Routing\ResponseFactory|\Illuminate\Http\Response
     */
    public function destroy(Product $product)
    {
        $product->genres()->detach();
        $product->delete();
        return response(null, 204);
    }
}
```

ProductRequest.php

```
<?php

namespace App\Http\Requests;

use App\Models\Product;
use Illuminate\Contracts\Validation\Factory as ValidationFactory;
use Illuminate\Contracts\Validation\Validator;
use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Http\Exceptions\HttpResponseException;
use Illuminate\Support\Str;
use Illuminate\Validation\Rule;

class ProductRequest extends FormRequest
{

    protected function createDefaultValidator(ValidationFactory $factory)
    {
        $this->slug = Str::slug($this->title, '-');
        return parent::createDefaultValidator($factory);
    }
```

```
/**
 * Determine if the user is authorized to make this request.
 *
 * @return bool
 */
public function authorize()
{
    return true;
}

/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */

public function rules()
{
    return [

        'id' => ['gt:0', 'integer',Rule::unique('products')->ignore($this->product)],

        'slug' => ['required', 'regex:/^[a-z0-9]+(?:-[a-z0-9]+)*$/i',
            Rule::unique('products')->ignore($this->product)],

        'price' => ['required', 'integer', 'gt:0'],
        'title' => ['required', 'string', Rule::unique('products')->ignore($this->product)],

        'description' => ['required', 'string'],
        'photo' => ['required', 'string'],
        'platformId' => ['integer', 'exists:platforms,id'],
        'caseId' => ['integer', 'exists:cases,id'],

        'genres.*' => 'distinct',
        'genres' => ['array', 'exists:genres,id',
            Rule::unique('genre_product', 'genreId')->where(function ($query) {
                $query->where('productId', $this->product);
            })
        ],
```

```
            'date' => ['required', 'date_format:d.m.Y' /*,'before_or_equal:today'*/],
        ];
    }
}
```

ProductResource.php

```
<?php

namespace App\Http\Resources;

use App\Models\Account;
use App\Models\Key;
use App\Models\MyCase;
use App\Models\Order;
use App\Models\Platform;
use App\Models\Product;
use Illuminate\Http\Resources\Json\JsonResource;
use Illuminate\Support\Str;

class ProductResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param \Illuminate\Http\Request $request
     * @return array
     */
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'slug' => $this->slug,
            'price' => $this->price,
            'title' => $this->title,
            'description' => $this->description,
            'photo' => $this->photo,
            'date' => $this->date,
            'genres' => GenreResource::collection($this->genres),
            'platform' => new PlatformResource($this->platform),
```

```
            'case' => new MyCaseResource($this->case),
            'type' => [
                'keys' => $this->keys->whereNotIn('id', Order::all()->pluck('keyId'))->count(),
                'accounts' => $this->accounts->whereNotIn('id', Order::all()->pluck('accountId'))->count(),
            ],
        ];
    }
}
```