

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРНЕТ-МАГАЗИНУ

ЦИФРОВИХ ТОВАРІВ

SOFTWARE IMPLEMENTATION OF AN ONLINE DIGITAL GOODS STORE

Виконав: студент(ка) 4 курсу, групи КНТ-218

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Риндін П.Ю.

(прізвище та ініціали)

Керівник Колпакова Т.О.

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Риндіна Павла Юрійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація інтернет-магазину цифрових товарів

Software Implementation of an Online Digital Goods Store

керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “ 27 ” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 07 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Колпакова Т.О., доцент		
Нормоконтролер	Липовець М.В., асистент		

7. Дата видачі завдання “04” квітня 2022 року.

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Риндін П.Ю.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Колпакова Т.О.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 95 с., 35 рис., 1 табл., 3 дод., 15 джерел.

МАГАЗИН ІГОР, МЕНЕДЖМЕНТ ЦИФРОВИХ ТОВАРІВ, АДМІНІСТРУВАННЯ ДОСТУПУ, ВЕБЗАСТОСУНОК, ХМАРНЕ СХОВИЩЕ, MYSQL, FLASK RESTFULL, АВТОТЕСТИНГ.

Об'єкт дослідження – процес адміністрування вебсервісу для продажу цифрових товарів. Предмет дослідження – програмне забезпечення продажу цифрових товарів.

Мета роботи – розробка програмного забезпечення адміністрування та продажу цифрових товарів.

Матеріали, методи та технічні засоби: мова програмування Python, система бази даних MySQL, інфраструктура хмарного зберігання AWS, сервіс безперервної інтеграції Travis CI.

Результати. Розроблено програмне забезпечення купівлі та продажу цифрових товарів з можливістю адміністрування асортименту та налаштування прав доступу клієнтів. Реалізована база даних та архітектура хмарного сховища. Налаштована система юніт автотестування за допомогою вебсервісу безперервної інтеграції.

Висновки. Реалізований вебсервіс надає можливість автентифіковуватись та реєструватись, виконувати пошук та фільтрацію товарів, залишати коментарі, переглядати, видаляти та модифікувати екземпляри продуктів, додавати та видаляти їх з кошика, оформлювати покупку, налаштовувати права користувачів, переглядати та модифікувати їх профілі.

Галузь використання. Адміністрування процесу покупки та продажу ліцензійних копій комп'ютерних ігор.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 95 pages, 35 figures, 1 table, 3 appendixes, 15 sources.

GAME STORE, DIGITAL GOODS MANAGEMENT, ACCESS ADMINISTRATION, WEBSITE, CLOUD STORAGE, MYSQL, FLASK RESTFULL, AUTOTESTING.

The object of research is the process of administrating of the web service for the sale of digital goods. The subject of research – software implementation of an online digital goods store.

The purpose of the work is to develop software for the administration and sale of digital goods with the possibility of purchasing.

Materials, methods and technical means: Python programming language, MySQL database system, AWS cloud storage infrastructure, Travis CI continuous integration service.

Results. Software for buying and selling digital goods with the ability to administer the range and configure access rights for customers was developed. Database and cloud storage architecture was implemented. A system of self-testing units was configured with the help of a web service of continuous integration.

Conclusions. Implemented web service provides the ability to authenticate and register for a web application session by logging in to gain access rights, configure search and filter products, leave comments, view, delete and modify instances of products, add and remove them from the cart, make purchases , configure user rights, view and modify their profiles.

Field of usage. Administrating the process of buying and selling licensed copies of computer games.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	7
Вступ.....	8
1 Аналіз предметної області	10
1.1 Вимоги та функціонал інтернет-магазину.....	10
1.2 Інтернет-магазини з продажу копій цифрових ігор	11
1.3 Висновки за розділом 1	14
2 Розробка архітектури програми.....	15
2.1 Функціональні вимоги до програмного забезпечення	15
2.2 Вибір засобів розробки програмного продукту.....	22
2.3 Проєктування бази даних	28
2.4 Висновки за розділом 2	31
3 Розробка програми.....	32
3.1 Алгоритм функціонування програмного забезпечення.....	32
3.2 Структура реалізованого програмного забезпечення.....	34
3.3 Опис прийнятих під час реалізації рішень	36
3.4 Висновки за розділом 3	40
4 Експлуатація та тестування розробленої програми	41
4.1 Призначення програми.....	41
4.2 Умови виконання програми.....	41
4.3 Виконання програми	41
4.4 Тестування розробленої програми	47
4.5 Висновки за розділом 4.....	48
Висновки.....	49
Перелік джерел посилання	50
Додаток А Технічне завдання.....	52
Додаток Б Текст програми.....	58
Додаток В Слайди презентації	88

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
AWS	– Amazon Web Services;
CI	– Continious Integration;
CSS	– Cascading Style Sheets;
HTML	– HyperText Markup Language;
JSON	– JavaScript Object Notation;
REST	– Representational State Transfer;
SQL	– Structured Query Language;
WSGI	– Web Server Gateway Interface;
БД	– база даних;
СКБД	– система керування базами даних.

ВСТУП

У сучасному світі бізнес активно йде в онлайн простір. Для підприємців-початківців і продавців вже з досвідом онлайн маркетинг та арбітраж є відмінним способом вийти на необхідні обсяги продажу. Тож розробка та просування приватних торгових майданчиків є трендом ІТ вебіндустрії. Інтернет-магазин дає можливість презентувати товар користувачам у мережі, а вони легко можуть його оглянути, прицінитись та придбати. У свою чергу продавець реалізує можливість укладання угоди про продаж, чим і відрізняється від звичайного оголошення та переходить у рамки електронної комерції. Угода або електронний договір укладається дистанційно, тільки на підставі опису товару та без можливого фізичного доступу до нього на момент покупки. Тим самим існує комфортна можливість продажу не тільки цифрових товарів.

Алгоритм отримання та оформлення товару з боку покупця включає пошук та підключення до потрібного вебресурсу за необхідними характеристиками, вибір конфігурації та налаштувань конкретного продукту, оформлення замовлення з подальшим підтвердженням за допомогою інтегрованої системи або інших сервісів. Звичайна покупка здійснюється за допомогою сервісів онлайн банкінгу та інтернет грошових переводів, або оплатою при отриманні у відділенні пошти чи доставки кур'єром. При продажі цілком або умовно цифрових товарів продавець уникає цілої низки складних логістичних схем, але повинен надати гарантії доступу та повернення товару. Алгоритм отримання та реалізації повинен бути вказаний самим продавцем, це може бути: зберігання ключа-копії на стороні інтернет-магазину у бібліотеці користувача, відправка інтернет-поштою, чи доставка іншим можливим способом.

Суть інтернет-магазину полягає у наданні самореалізованої продукції або послуг, де постачальник часто має прямий зв'язок з виробником. Але це не завжди так, велика частина інтернет сайтів з продажу профілюються як

маркетплейси або магазини з вільним асортиментом.

Вони мають змогу реалізувати товар різних продавців, беручи з кожного лоту певний відсоток від загальної ціни товару. Цей варіант найбільше підходить реселлерам та офіційним агрегаторам фірмових товарів та послуг.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вимоги та функціонал інтернет-магазину

Важливим показником при організації онлайн продажів є правильне формулювання запитів з боку покупця. В залежності від потрібних товарів структура та стратегія просування інтернет магазину можуть бути різними. У світі великий успіх має продаж цифрових ліцензійних копій різного роду програмного забезпечення. Наприклад, комп'ютерних ігор, системних утиліт, програм офісного та виробничого типу. У контексті цієї розробки було обрано область продажу ліцензійних ключів комп'ютерних ігор.

Специфіка товару передбачає певне оформлення і репрезентацію магазину і певні функції для полегшення і поліпшення досвіду використання покупцем та персоналом. Щодо функціоналу, інтернет-магазин – це система з розподіленими обов'язками та групами користувачів. Кожна з груп має свої права та повноваження вчиняти певні дії. Визначення користувача в потрібну групу відбувається під час автентифікації в систему, коли виходить група скидається. У свою чергу, кожен авторизований користувач має можливість переглядати та змінювати дані у своєму профілі, повністю взаємодіяти з продуктами в магазині, залишати оцінки до них та оформляти у кошик для подальшої реєстрованої покупки.

На відміну від авторизованих користувачів, неавторизований покупець не може бути встановлений у системі, тому має обмежений, мінімально необхідний функціонал для оформлення та купівлі ігор анонімно. Керівну роль мають адміністратори та менеджери. Адміністратор є основною роллю в інфраструктурі сервісу, його профіль створюється автоматично при розгортанні та запуску системи через важливість у працездатності більшості функціоналу сайту. Адміністратор має можливість переглядати та змінювати всі дані на сторінках користувачів та товарів, додавати та видаляти ресурси вручну.

Головною особливістю цієї групи користувачів є можливість надавати

ролі іншим учасникам. Роль менеджера успадковує всі права адміністратора, крім можливості призначати ролі користувачам, та в основному використовується для налаштування роботи магазину.

У магазинах з великим асортиментом потрібен алгоритм пошуку та сортування товарів. Реалізується система пошуку за назвами разом із сортуванням за ключовими словами. Для більш точного пошуку по продуктах, вводяться додаткові теги і категорії, що дають можливість вводити пошукові запити тонкого налаштування.

1.2 Інтернет-магазини з продажу копій цифрових ігор

На ринку представлено багато різноманітних реалізацій інтернет-магазинів із продажу цифрових копій комп'ютерних ігор.

Gog.com [1] (рис. 1.1) – сервіс цифрової дистрибуції ігор. Продукти можна купувати та завантажувати онлайн, і всі вони надаються без захисту. Цінова політика сервісу спочатку передбачала дві категорії ігор – 5\$ і 10\$. Але зі стартом продажів нових ігор, а пізніше і за підтримки регіональних цін, цінових категорій з'явилося набагато більше. Знижки регулярно пропонуються на окремі ігри або добірки у вихідні та будні.

Користувачам не потрібно завантажувати будь-які клієнти або програми для завантаження чи оновлення ігор, але є додатковий менеджер завантажень, який дозволяє автоматизувати завантаження та перевіряти установники завантажених ігор. Всі ігри можна зберігати на будь-який носій та перезавантажувати необмежену кількість разів. Завантажені ігри не прив'язані до профілю користувача на GOG.com, але за законом регулюються користувальницькою угодою.

Щоб ігри правильно працювали на останніх версіях операційних систем та пристроїв, деякі ігри були оновлені та поставляються з відкритими емуляторами та сумісністю. Іноді гра містить додаткові оновлення, зроблені сторонніми розробниками.

Крім самої гри, користувач також отримує додаткові матеріали, які додаються до гри та розміщуються на сторінці товару. Цей матеріал зазвичай включає саундтрек до гри, фони, аватари та керівництва. GOG також пропонує повну багатомовну підтримку клієнтів та 30-денну гарантію повернення грошей. А для користувачів, чия регіональна ціна вища, ніж ціна в США, існує програма "справедливої ціни", в рамках якої користувачі отримують "зайву" суму назад як кредит для подальших покупок [2].



Рисунок 1.1 – сервіс цифрової дистрибуції ігор Gog.com

Humble Bundle [3] (рис. 1.2) — це цифровий магазин відеоігор, який виник із оригінальної пропозиції Humble Bundle, колекцій ігор, які продаються за ціною, визначеною покупцем, і частина ціни йде на благодійність, а решта розподіляється між розробниками гри. Humble Bundle продовжує пропонувати ці обмежені за часом пакети, але розширився, включивши в нього більший і більш стійкий магазин. Початкові пакети, як правило, являли собою колекції незалежно розроблених ігор з підтримкою платформ наданих без керування цифровими правами.

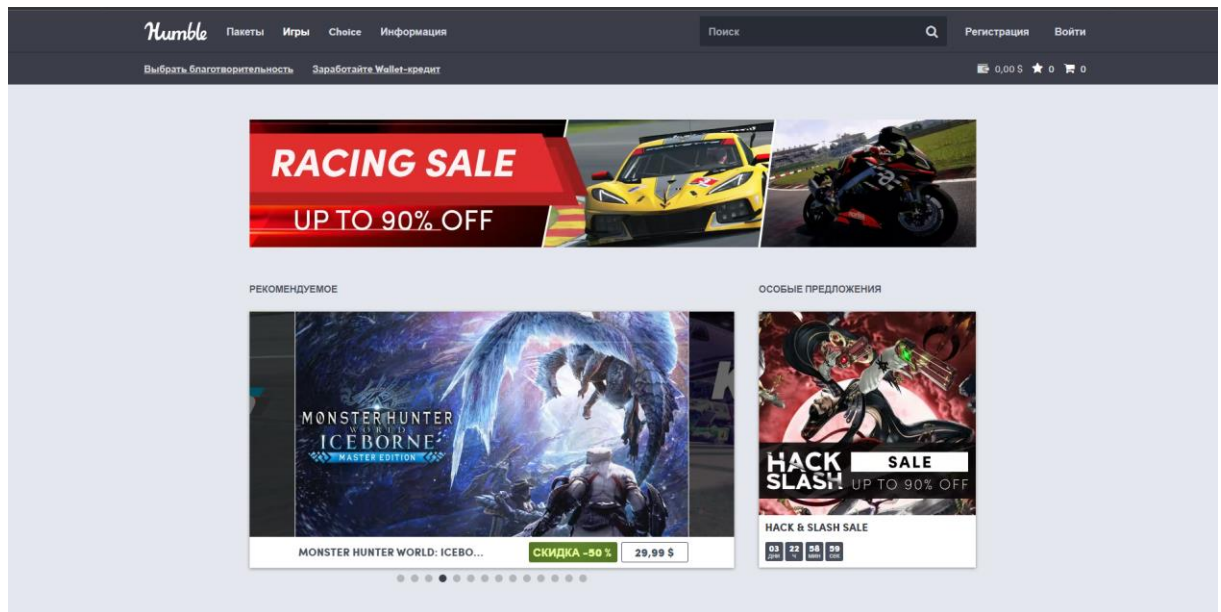


Рисунок 1.2 – Цифровой магазин відеоігор Humble Bundle

Green Man Gaming [4] (рис. 1.3) — британський роздрібний продавець, розповсюджувач і видавець відеоігор. Він має багатоплатформний каталог із понад 9000 ігор від понад 1350 видавців, які продають ігри в 195 країнах із використанням понад 140 платіжних систем і підтримують 18 валют. 90% його доходу отримується за межами Великобританії. Видавничий підрозділ Green Man Gaming дозволяє незалежним студіям відеоігор вибирати комплекс послуг для публікації.

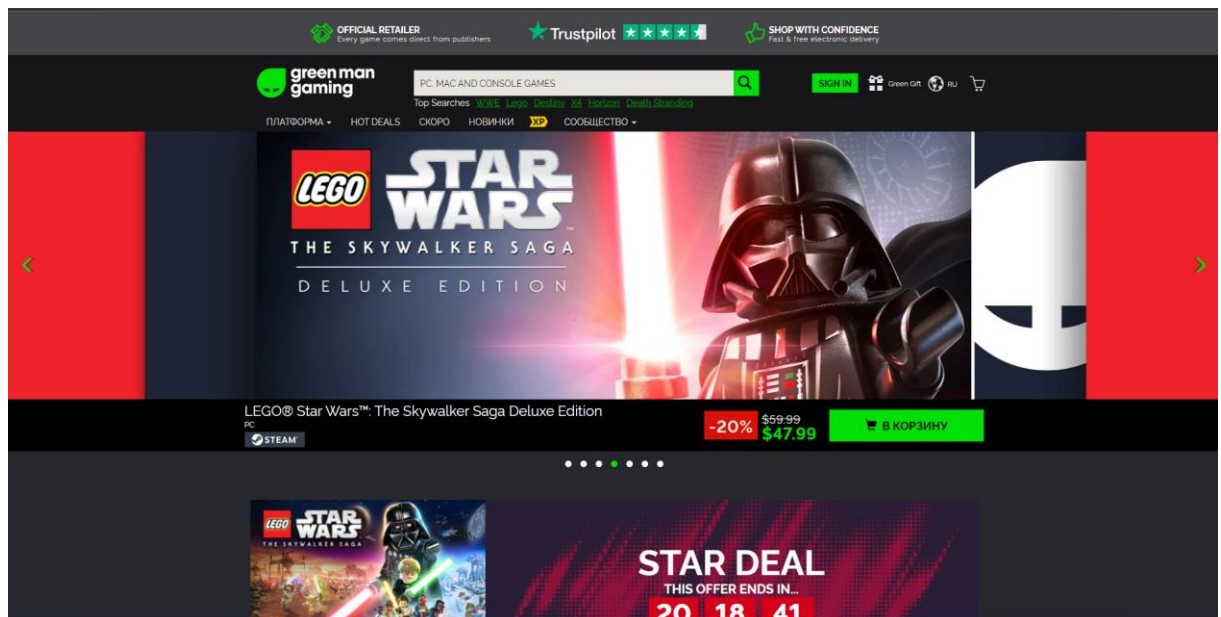


Рисунок 1.3 – Цифровой магазин відеоігор Green Man Gaming

1.3 Висновки за розділом 1

В даному розділі був проведений аналіз ринку та предметної області, який виявив підходи до реалізації поставленої мети та налаштування інтернет-магазину цифрових копій комп'ютерних ігор. Також були встановлені основні принципи проєктування та підтримки такого роду вебресурсів для подальшого покращення інтерфейсу користувача та адміністрування у власній програмній реалізації.

Завдання дипломної кваліфікаційної роботи полягають в наступному:

- аналіз предметної області;
- проєктування програмного забезпечення;
- реалізація програмного забезпечення.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Функціональні вимоги до програмного забезпечення

На основі аналізу аналогів та огляду загальноновживаних принципів побудови інтернет магазинів в проєкті було виділено чотири ключові ролі, адміністратор, менеджер, покупець та неавторизований користувач. Нижче наведено функціональні вимоги для кожної з цих ролей.

Сервіс повинен забезпечувати наступний функціонал адміністратора:

- відображення всієї колекції ігор у магазині;
- можливість пошуку ігор за назвою;
- фільтрація ігор за жанрами;
- фільтрація ігор за піджанрами;
- можливість реєстрації нового користувача;
- входження до системи за допомогою облікових даних;
- відображення особистих облікових даних;
- відображення облікових даних інших користувачів;
- можливість змінювати особисті облікові дані;
- можливість змінювати облікові дані інших користувачів;
- можливість змінювати ролі іншим користувачам;
- перегляд лоту гри;
- створення нового лоту;
- додавання інформації про лот;
- налаштування жанрів та піджанрів гри;
- додавання картинки та мініатюри до гри;
- зміна інформації про лот;
- видалення лоту;
- перегляд коментарів інших користувачів під грою;
- залишення своїх коментарів;
- видалення власних коментарів ;
- видалення коментарів інших користувачів;

- можливість відповідати на коментарі інших користувачів;
- можливість відновлювати коментарі.

Функціональні вимоги менеджера є наступними:

- відображення всієї колекції ігор у магазині;
- можливість пошуку ігор за назвою;
- фільтрація ігор за жанрами;
- фільтрація ігор за піджанрами;
- можливість реєстрації нового користувача;
- входження до системи за допомогою облікових даних;
- відображення особистих облікових даних;
- відображення облікових даних інших користувачів;
- можливість змінювати особисті облікові дані;
- можливість змінювати облікові дані інших користувачів;
- перегляд лоту гри;
- створення нового лоту;
- додавання інформації про лот;
- налаштування жанрів та піджанрів гри;
- додавання картинки та мініатюри до гри;
- зміна інформації про лот;
- видалення лоту;
- перегляд коментарів інших користувачів під грою;
- залишення своїх коментарів;
- видалення власних коментарів ;
- видалення коментарів інших користувачів;
- можливість відповідати на коментарі інших користувачів;
- можливість відновлювати коментарі.

Функціональні вимоги покупця включають:

- відображення всієї колекції ігор у магазині;
- можливість пошуку ігор за назвою;
- фільтрація ігор за жанрами;

- фільтрація ігор за піджанрами;
- можливість реєстрації нового користувача;
- входження до системи за допомогою облікових даних;
- відображення особистих облікових даних;
- відображення облікових даних інших користувачів;
- можливість змінювати особисті облікові дані;
- перегляд лоту гри;
- перегляд кошика покупок;
- додавання товарів до кошику покупок;
- видалення товарів з кошика покупок;
- оформлення та оплата покупки;
- перегляд історії своїх покупок;
- можливість оцінювати лот;
- перегляд коментарів інших користувачів під грою;
- залишення своїх коментарів;
- видалення власних коментарів;
- можливість відповідати на коментарі інших користувачів.

Неавторизований користувач не може бути встановлений у системі, тому має обмежений і буде мати мінімально необхідний функціонал для оформлення та купівлі ігор анонімно. Сценарій використання сервісу неавторизованим користувачем указано на рис 2.1.

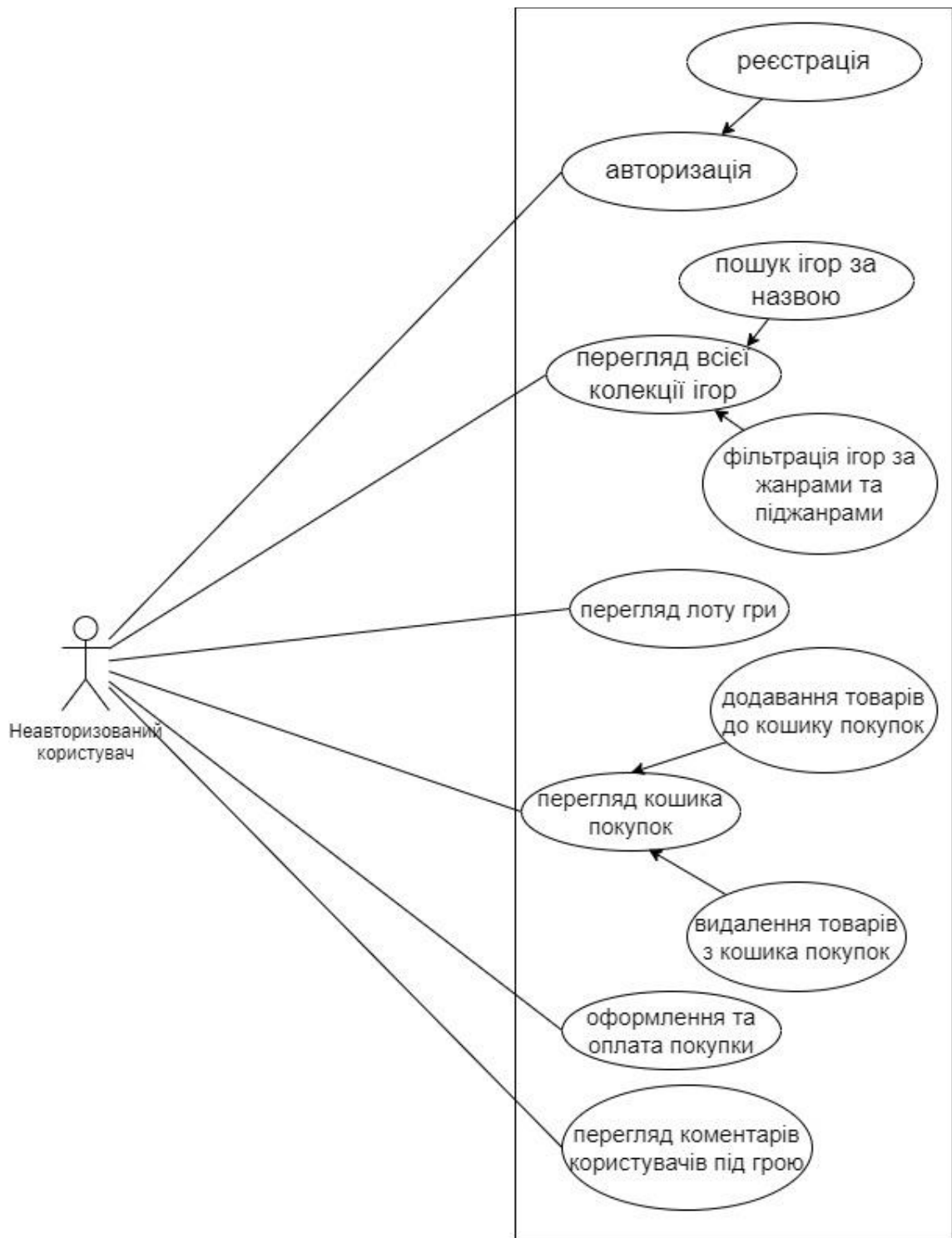


Рисунок 2.1 – Діаграма прецедентів неавторизованого користувача

Покупець може переглядати та змінювати дані у своєму профілі, повністю взаємодіяти з продуктами в магазині, залишати оцінки до них та додавати у кошик для подальшої реєстрованої покупки. Сценарій використання сервісу авторизованим користувачем указано на рис. 2.2.



Рисунок 2.2 – Діаграма прецедентів авторизованого користувача

Керуючу роль має адміністратор та менеджер. Менеджер може бути призначеним та головна його задача стежити за функціонуванням сайту та бути у ролі модератора системи. Сценарій використання сервісу менеджером указано на рис. 2.3.

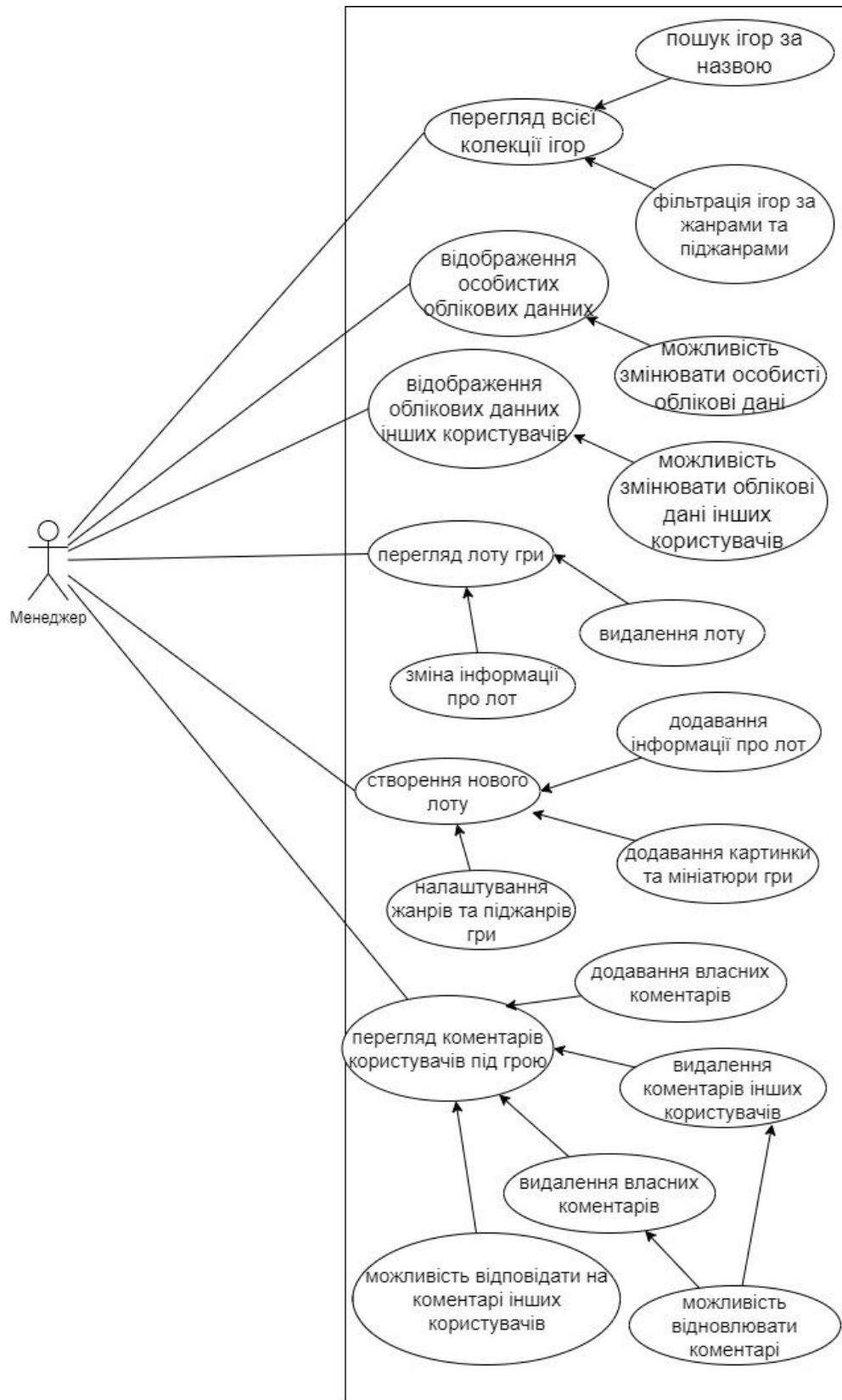


Рисунок 2.3 – Діаграма прецедентів менеджера

Профіль адміністратора створюється автоматично при розгортанні та запуску системи через важливість у працездатності критично важливого

функціоналу сайту. Адміністратор має можливість переглядати та змінювати всі дані на сторінках користувачів та товарів, додавати та видаляти ресурси вручну. Сценарій використання сервісу адміністратором указано на рис. 2.4.

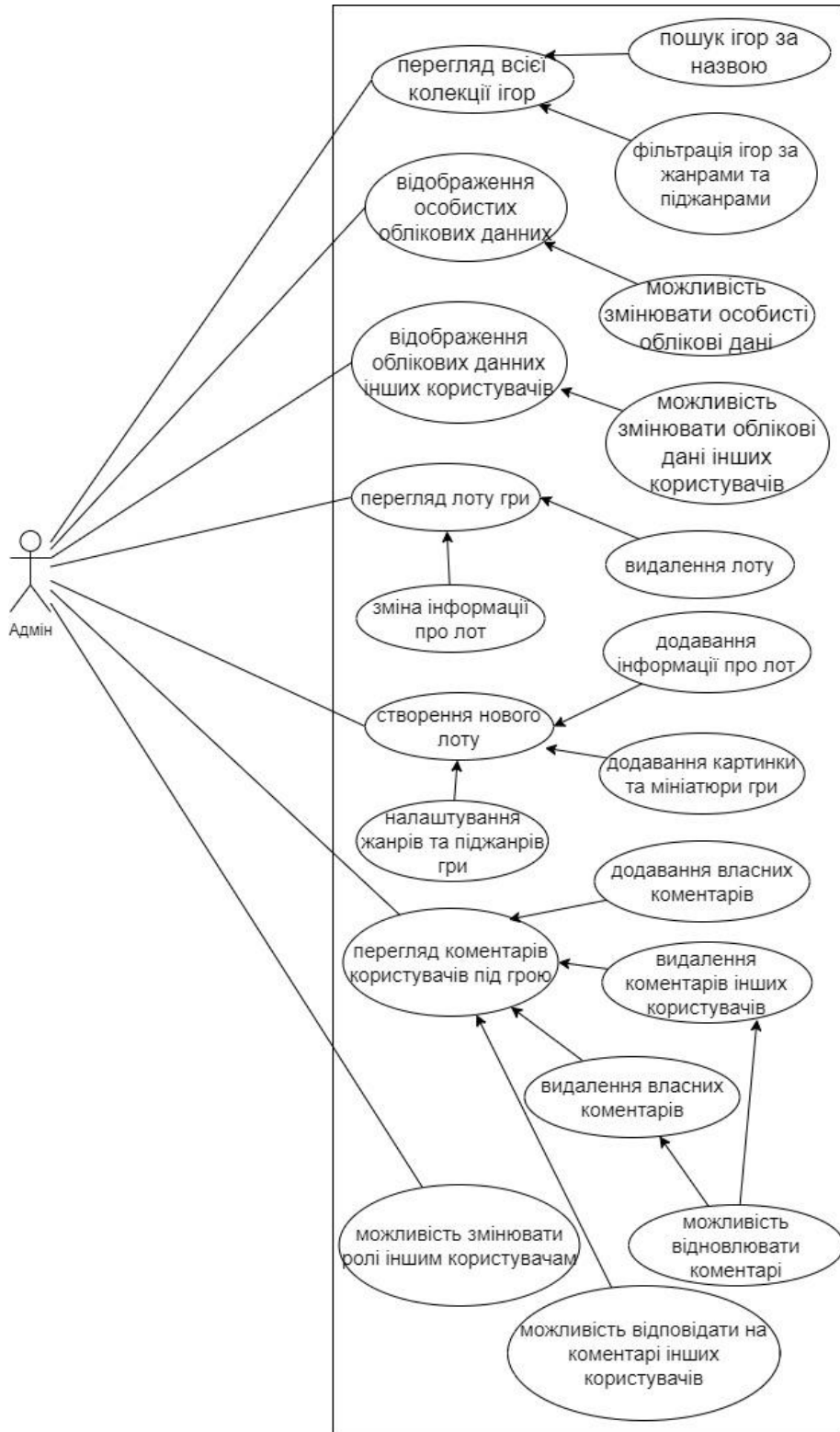


Рисунок 2.4 – Діаграма прецедентів адміністратора

2.2 Вибір засобів розробки програмного продукту

Перед початком розробки було проведено аналіз існуючих рішень та мов програмування для реалізації серверної складової. В результаті була обрана мова програмування Python.

Python — це високорівнева, універсальна і дуже популярна мова програмування. Python використовується у веброзробці, додатках машинного навчання, а також в усіх передових технологіях в індустрії програмного забезпечення. Дуже добре підходить для початківців, а також для досвідчених програмістів з іншими мовами програмування, такими як C++ і Java.

На даний момент Python є найбільш широко використовуваною багатоцільовою мовою програмування високого рівня. Python дозволяє програмувати в об'єктно-орієнтованих і процедурних парадигмах. Програми на Python, як правило, менші, ніж інші мови програмування, такі як Java. Програмістам доводиться вводити відносно менше, а вимога до відступів у мові робить їх читаними весь час. Мова Python використовується майже всіма технологічними гігантами, такими як Google, Amazon, Facebook, Instagram, Dropbox, Uber, тощо.

Найбільшою перевагою Python є величезна колекція стандартних бібліотек що включають в себе: машинне навчання, програми з графічним інтерфейсом, вебфреймворки, такі як Django та Flask, які використовуються YouTube, Instagram, Dropbox, обробка зображень, вебскребінг, тестінг, обробка та аналіз мультимедіа, наукові обчислення, обробка тексту, та інше [6].

Найпопулярнішими вебфреймворками, що використовуються для написання бекенду на мові Python, являються Flask та Django.

Flask – це мікрофреймворк, що пропонує базові функції веббібліотеки. Цей фреймворк немає залежностей від зовнішніх бібліотек. Фреймворк пропонує розширення для перевірки форми, об'єктно-реляційні

перетворювачі, відкриті системи аутентифікації, механізм завантаження та кілька інших інструментів [7].

Django – це середовище веброботи для Python. Цей фреймворк пропонує стандартний метод швидкої та ефективної розробки вебсайтів. Це допомагає у створенні та підтримці якісних вебдодатків та дозволяє зробити процес розробки плавним та економити час. Вебфреймворк високого рівня, що дозволяє виконувати швидку розробку, також основною метою цієї вебінфраструктури є створення складних вебсайтів, керованих базами даних [8].

Важливі різниці між цими бібліотеками зображені у таблиці 2.1.

Таблиця 2.1 – Порівняння Python фреймворків

Характеристика	Flask	Django
1	2	3
Призначення	Вебфреймворк Python, створений для швидкої розробки	Вебфреймворк Python, створений для важких і комплексних проєктів
Тип використання вебсерверів	WSGI(Web Server Gateway Interface) фреймворк	Фуллстек вебфреймворк
Підтримка API	надає підтримку API (Application Programming Interface)	Django не підтримує API
Підтримка Visual Debug	Підтримка Visual Debug	Немає підтримки Visual Debug
Підтримка використання декількох баз даних	Дозволяє використовувати кілька типів баз даних	Не дозволяє працювати з декількома типами баз даних

Продовження таблиці 2.1

1	2	3
Підтримка динамічних сторінок HTML	Не працює з динамічними сторінками HTML (HyperText Markup Language)	Працює з динамічними сторінками HTML
Підтримка різноманітних парадигм до написання коду	Пропонує різноманітний стиль роботи	Пропонує монолітний стиль роботи
Додаткові модулі	Підтримує сторонні розширення, які можуть бути реалізовані у роботі	Django має власну бібліотеку модулів
Структура вебфреймворка	Структура макета проєкту для вебфреймворка Flask є довільною	Структура макета проєкту для Django чітка та незмінна
Підхід до URL-адресів системи	Диспетчер URL-адрес вебфреймворку - це запит REST (Representational State Transfer)	Диспетчер URL-адрес цього фреймворку Django заснований на контролері-регулярних виразів
Ступінь новизни вебфреймворку	Flask є набагато молодшою платформою в порівнянні з Django	Django — це дуже зрілий фреймворк

Найкращими характеристиками Flask є його легкість, відкритий

вихідний код і мінімальне кодування для розробки програми, на відміну кращими особливостями Django є середня швидкість розробки, велика спільнота, легкість навчання [9].

На основі наведених фактів, Flask був визнаний більш підходящим та зручним рішенням для поставленого завдання.

Для клієнтської складової не було обрано спеціалізованого фреймворку, замість цього було вирішено обмежитись стандартною мовою браузерної взаємодії Javascript.

JavaScript — це мова програмування, яка використовується як на стороні клієнта, так і на стороні сервера, що дозволяє зробити вебсторінки інтерактивними.

Якщо HTML і CSS (Cascading Style Sheets) є мовами, які надають структуру та стиль вебсторінкам, JavaScript надає вебсторінкам інтерактивні елементи, які залучають користувача.

Включення JavaScript покращує роботу вебсторінки, перетворюючи її зі статичної сторінки в інтерактивну, бо JavaScript додає поведінку вебсторінкам та в основному використовується для вебдодатків і веббраузерів [10].

При виборі системи управління баз даних були розглянуті PostgreSQL та MySQL.

Postgres — це об'єктно-реляційна база даних, а MySQL — чисто реляційна база даних. Це означає, що Postgres включає такі функції, як успадкування таблиць і перевантаження функцій, які можуть бути важливими для певних програм. Postgres також точно дотримується стандартів SQL [11].

SQL дозволяє об'єднувати таблиці за допомогою кількох рядків вихідного коду із простою структурою, яку може швидко освоїти більшість нетехнічних працівників. При використанні SQL аналітикам не потрібно знати, де на диску знаходиться таблиця замовлень, як здійснити пошук для пошуку певного замовлення або як з'єднати таблиці замовлень та клієнтів.

База даних компілює запит та обчислює правильні точки даних.

І MySQL, і PostgreSQL підтримують нотацію об'єктів JavaScript (JSON)(JavaScript Object Notation) для зберігання та передачі даних, хоча PostgreSQL також підтримує JSONB, двійкову версію JSON, яка усуває дублювання ключів та сторонні прогалини.

Обидві бази даних пропонують надійну підтримку спільноти на додаток до традиційних механізмів підтримки. Вони надають доступ до наявних даних та допомагають отримати нові ідеї для прийняття більш ефективних рішень або виявлення нових можливостей.

PostgreSQL підтримує як нереляційні, і реляційні типи даних. Вона була названа однією з найсумісніших, стабільних і зрілих реляційних баз даних, доступних сьогодні, і може легко обробляти складні запити.

Особливості PostgreSQL включають:

- відновлення на час для відновлення баз даних на певний час;
- упереджувальний журнал, який реєструє всі зміни у базі даних;
- збережені процедури для створення та збереження підпрограм користувача.

Основні переваги PostgreSQL:

- продуктивність та масштабованість, включаючи геопросторову підтримку та необмежений паралелізм, а також глибокий та великий аналіз даних різних типів;
- підтримка паралелізму за рахунок використання керування паралелізмом кількох версій, який забезпечує одночасне виконання операцій запису та читання;
- глибока мовна підтримка завдяки його сумісності та підтримці кількох мов програмування, включаючи Python, Java, JavaScript, C/C++ та Ruby;
- безперервність бізнесу з високою доступністю послуг завдяки методам асинхронної або синхронної реплікації між серверами;
- велика гнучкість та економічно ефективні інновації завдяки

технології управління базами даних із відкритим вихідним кодом.

MySQL – швидка, надійна, масштабована і проста у використанні система реляційних баз даних з відкритим вихідним кодом - призначена для роботи з критично важливими, високонавантаженими виробничими програмами. Це проста і проста в запуску база даних з низьким рівнем використання пам'яті, диска і ЦП, керована системою управління реляційними базами даних. MySQL Community Edition - це безкоштовна завантажена версія, що підтримується активним онлайн-спільнотою.

MySQL надає ринку безліч переваг, у тому числі такі:

- неперевершена безпека даних у порівнянні з іншими платформами управління базами даних. Забезпечується цілісність даних, що робить її популярною базою даних для вебзастосунків;
- висока продуктивність, оскільки структура механізму зберігання MySQL підтримує вимогливі програми з високошвидкісними частковими індексами, повнотекстовими індексами та унікальними кешами пам'яті для чудової продуктивності бази даних;
- масштабованість та підтримка необмеженого збільшення обсягу сховища при невеликому розмірі;
- гнучка платформа з відкритим вихідним кодом з підтримкою обробки транзакцій, хоч і не така гнучка, як нереляційні бази даних, такі як NoSQL.

Функціонал вебсервісу включає в себе використання важких статичних файлів, таких як картинки. Для вирішення проблеми їх зберігання було вибрано використання файлового вебхостингу для зберігання та отримання даних. Наразі був обраний Amazon Simple Storage Service, це сервіс зберігання об'єктів, що пропонує найкращі в галузі показники продуктивності, масштабованості, доступності та безпеки даних. Є можливість зберігати та захищати необхідний обсяг даних для практично будь-якого прикладу використання. Наприклад, для кластерів даних, хмарних програм та мобільних програм. Вигідні класи сховища та прості у

використанні інструменти адміністрування дозволяють оптимізувати витрати, організувати дані та точно налаштувати обмеження доступу відповідно до потреб бізнесу чи законодавчих вимог [12].

При створенні складних багаторівневих систем потрібно мати повну інформацію про стани модулів системи залежно від стадії розробки. Для перевірки статусу роботи функціоналу вдаються до юніт тестування та інтеграційного тестування. Для розробки тестів програми було обрано вбудований Python модуль Pytest. Цей фреймворк дозволяє легко писати невеликі, читабельні тести і може масштабуватися для підтримки складного функціонального тестування для програм і бібліотек [13]. Для спрощення автотестування було прийнято рішення використати сервіс тривалої інтеграції. Для використання було обрано Travis CI. Travis CI — розподілений вебсервіс для складання та тестування програмного забезпечення, що використовує GitHub як хостинг вихідного коду. Програмна складова сервісу також розташовується на GitHub [14].

Таким чином, PostgreSQL та MySQL можуть використовуватись по-різному, і вибір між ними залежить від завдань та ресурсів підприємства. В цілому, PostgreSQL – це надійна, просунута система управління базами даних, що підходить великим проєктам з складною структурою. Тим не менш, MySQL – ідеальне рішення для компанії, обмеженої бюджетом і простором [15].

На основі аналізу при докладному розборі специфіки кожної системи управління баз даних, для даної роботи була обрана MySQL.

2.3 Проектування бази даних

Схема бази даних складається з 9 сутностей, пов'язаних 9 зв'язками.

Сутність Game представляє лот товару що продається. Має наступні поля: id (ключ), title (назва товару), description (опис товару), uuid (складний кастомний ключ), is_visible (флаг, чи схована гра), price (ціна), is_delete

(флаг, чи видалена гра).

Сутність Comment представляє коментарій користувача під певним лотом. Має наступні поля: id (ключ), uuid (складний кастомний ключ), text (текст коментаря), created_date (дата створення), reply_to_message (посилання на ключ до коментаря на який постлається), user_id_from (ключ на користувача), game_id_from (ключ на гру).

Сутність User представляє профіль користувача. Має наступні поля: id (ключ), username (нік користувача), name (ім'я користувача), surname (прізвище користувача), email_address (почта користувача), password (пароль), uuid (складний кастомний ключ), role_id (посилання на ключ ролі до якої відноситься користувач).

Сутність Role представляє роль. Має наступні поля: id (ключ), title (назва), code (код ролі, для).

Сутність Genre представляє жанр гри. Має наступні поля: id (ключ), title (назва).

Сутність Subgenre представляє піджанр гри. Має наступні поля: id (ключ), title (назва).

Сутність User_Game представляє додаткову таблицю для зв'язку сутностей User з Game. Має наступні поля: user_id (посилання на ключ користувача), game_id (посилання на ключ гри), count (кількість копій однієї гри що придбав користувач).

Сутність Genre_Subgenre представляє додаткову таблицю для зв'язку сутностей Genre та Subgenre. Має наступні поля: id (ключ), genre_id (посилання на ключ жанру), subgenre_id (посилання на ключ субжанру).

Сутність Game_Genre_Subgenre представляє додаткову таблицю для зв'язку сутностей Genre_Subgenre та Game. Має наступні поля: game_subgenre_id (посилання на ключ додаткової таблиці з жанрами-піджанрами), game_id (посилання на ключ гри).

Схема БД представлена на рис. 2.5.

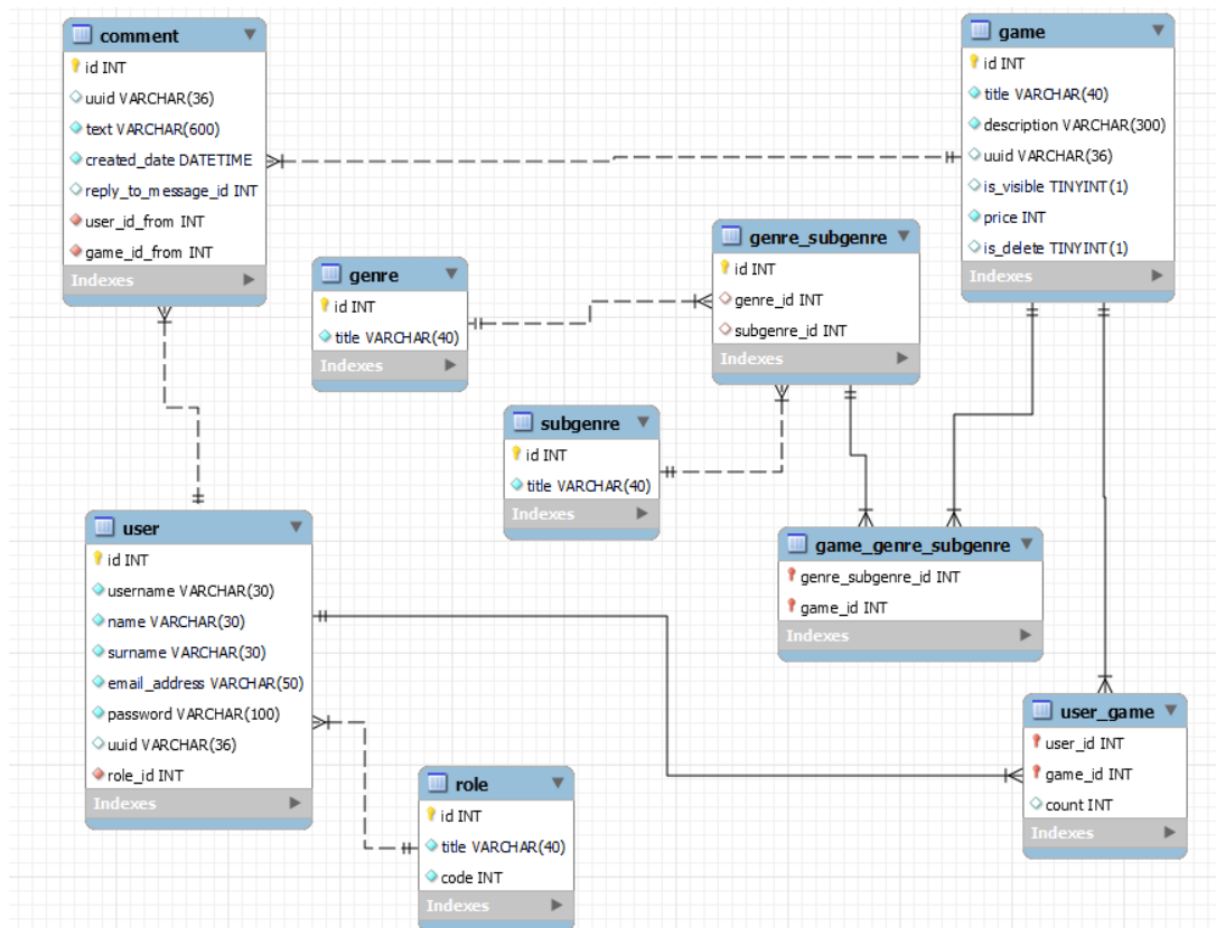


Рисунок 2.5 – Схема БД

Зв'язок між таблицями User і Game багато-до-багатьох, що реалізується за допомогою допоміжної таблиці User_Game, бо один користувач може мати декілька ігор, та одна гра може належати декільком користувачам.

Зв'язок між таблицями User і Role один-до-багатьох, бо один користувач може належати тільки до однієї ролі у системі, та може бути декілька користувачів з однаковими ролями.

Зв'язок між таблицями User і Comment один-до-багатьох, бо один екземпляр коментаря може бути написаним лише одним автором, та користувач може написати безліч коментарів.

Зв'язок між таблицями Comment і Game один-до-багатьох, бо один екземпляр коментаря може належати тільки до одного товару, та одна гра може містити безліч коментарів.

Зв'язок Genre_Subgenre і Genre та Genre_Subgenre і Subgenre один-до-багатьох та один-до-багатьох відповідно. Genre_Subgenre має ключі Genre та Subgenre, це дає можливість скомпіювати у таблиці Genre_Subgenre кастомні жанри для нових продуктів, маючи статичні жанри та піджанри.

Зв'язок Game і Genre_Subgenre багато-до-багатьох, що реалізується за допомогою допоміжної таблиці Game_Genre_Subgenre, бо один гра може мати декілька кастомних жанрів, та один кастомний жанр може належати декільком іграм.

2.4 Висновки за розділом 2

В даному розділі було визначено функціональні вимоги для реалізації програмного забезпечення. Було обрано мову програмування Python, Flask фреймворк для реалізації веб, та СКБД MySQL. Було розроблено структуру бази даних проєкту.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування програмного забезпечення

Логіка роботи сервісу залежить від ролі користувача в системі, що її використовує. Залежно від знаходження в одній із ролей: неавторизований покупець, авторизований покупець, менеджер, адміністратор, можливості та права відрізняються.

Спільним для всіх ролей є можливість перегляду головної сторінки сервісу з переліком всіх товарів, інформацією та ціною.

Також головна сторінка пропонує всім користувачам такі функції:

- відображення картинок-мініатюр з назвою продукту, ціною, жанром та піджанром, гіперпосиланням на сторінку продукту з більш детальною інформацією;
- можливість пошуку товару за назвою;
- можливість фільтрації товарів за жанром та піджанром.

Ще однією сторінкою, однаково доступною для перегляду всіма типами користувачів, є сторінка товару. При цьому візуальна та функціональні складові можуть відрізнятися.

Основні функції, що може надати сторінка товару:

- перегляд обкладинки гри у високій якості;
- перегляд ціни та авторського опису продукту;
- можливість залишати та переглядати коментарі, що стосуються продукту;
- можливість додати товар у кошик для подальшої покупки, якщо користувач не є управляючим персоналом.

Перелічені вище можливості безпосередньо впливають на інфраструктуру купівлі-продажу товарів у магазині. Інший функціонал, такий як профіль користувача, налаштування та додавання товару в асортимент, редагування кошика з покупками є опціональним, і залежить повністю від прав поточного користувача.

Схема функціонування сервісу наведена на рис. 3.1.

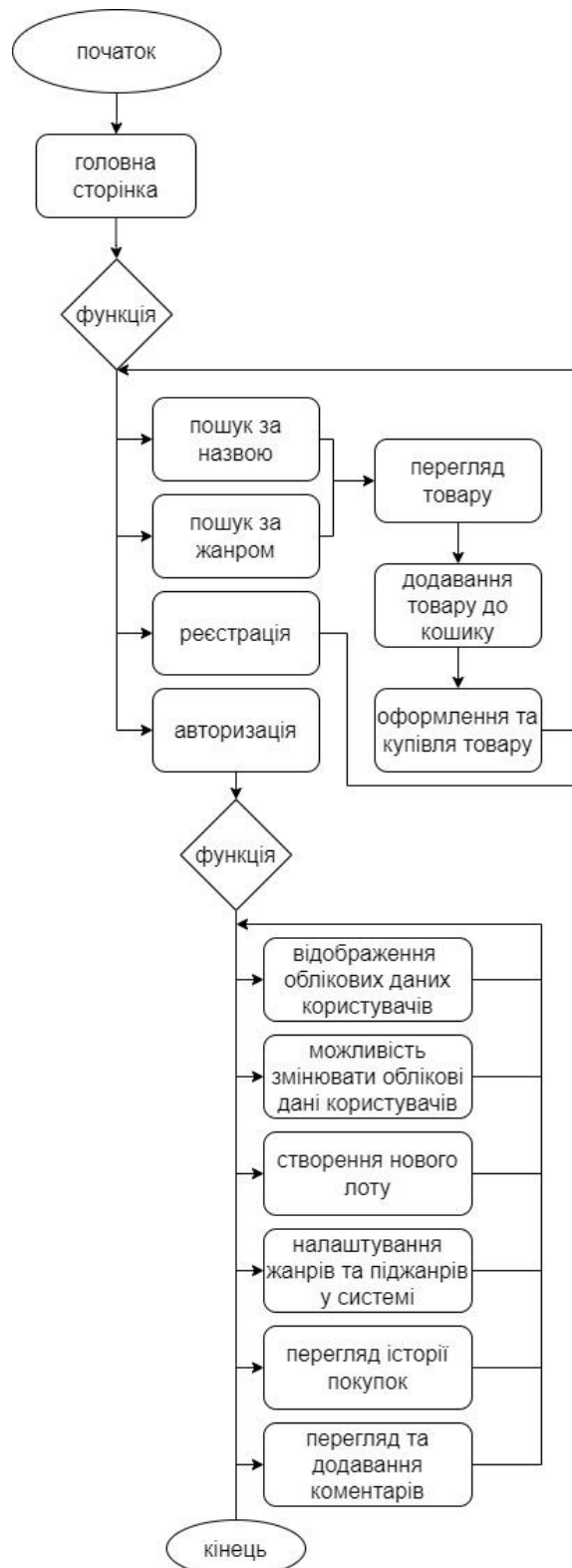


Рисунок 3.1 – Схема функціонування сервісу

3.2 Структура реалізованого програмного забезпечення

Структура поточної реалізації програмного забезпечення передбачає умовний розподіл ресурсів та файлів імплементації алгоритмів за директорією відповідних програмної логіки.

Створено наступні розділи:

- Resources (логіка кожного з доступних вебресурсів сервісу);
- Database (схеми та програмні реалізації моделі баз даних);
- Static (статичні файли та ресурси для фронтенд складової сервісу);
- Templates (шаблони вебсторінок);
- Other (допоміжний функціонал який не входить до основної логіки сервісу).

До розділу Resources належать наступні файли:

- addgame.py – логіка ресурсу з створення лоту;
- buygame.py – логіка ресурсу придбання лоту;
- carts.py – логіка ресурсу кошика з придбаними лотами;
- editgame.py – логіка ресурсу налаштування лоту;
- game.py – логіка ресурсу перегляду лоту;
- login.py – логіка ресурсу з авторизації;
- main.py – логіка ресурсу головної сторінки;
- order.py – логіка ресурсу з налаштування покупки;
- register.py – логіка ресурсу з реєстрації користувача;
- user.py – логіка ресурсу з перегляду профіля користувача.

До розділу Database належать наступні файли:

- inserts.py – скрипт для налаштування вхідних тестових даних у базу даних;
- models.py – схема моделей бази даних.

До розділу Static належать наступні файли:

- favicon.ico – іконка вебзастосунку;
- logo.png – логотип вебзастосунку;
- script.js – фронтенд логіка;
- style.css – фронтенд стилі.

До розділу Templates належать наступні файли:

- 404.html – шаблон вебсторінки з помилкою помилкового ресурсу;
- addgame.html – шаблон вебсторінки з створення лоту;
- base.html – шаблон вебсторінки з початковою структурою;
- carts.html – шаблон вебсторінки з кошиком придбання;
- editgame.html – шаблон вебсторінки з налаштуванням лоту;
- game.html – шаблон вебсторінки з переглядом лоту;
- main.html – шаблон вебсторінки з головним ресурсом;
- order.html – шаблон вебсторінки з налаштуванням покупки;
- register.html – шаблон вебсторінки з реєстрацією;
- user.html – шаблон вебсторінки з профілем користувача.

До розділу Other належать наступні файли:

- wsgi.py – скрипт запуску програми;
- __init__.py – логіка налаштування вебсервісу;
- aws_func.py – логіка взаємодії з хмарним сховищем;
- routes.py – налаштування шляхів до вебресурсів;
- token.py – логіка зберігання вебсесії користувача.

Схематичне представлення структури програмного забезпечення представлено на рис. 3.2.

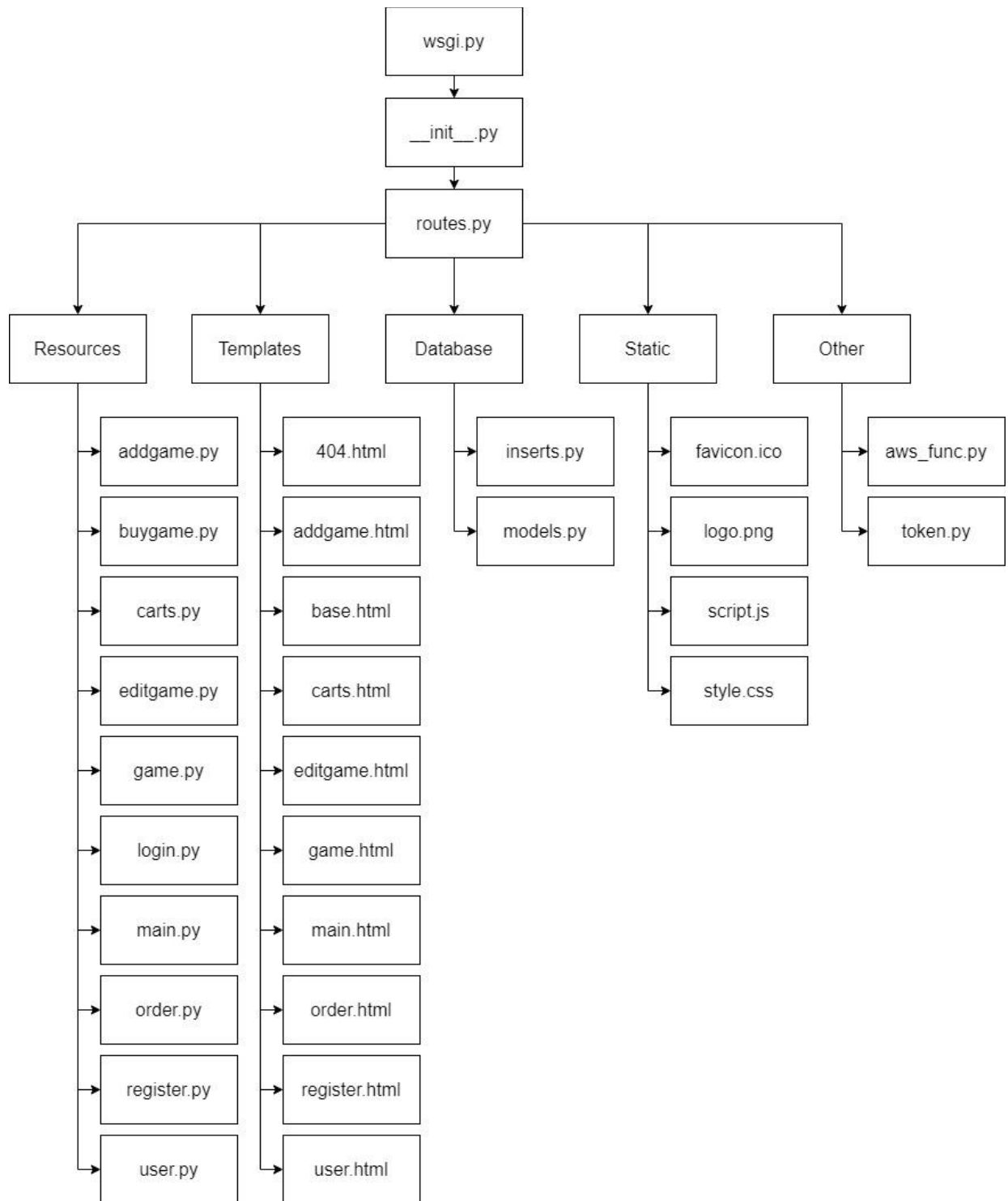


Рисунок 3.2 – Структурна схема програмного забезпечення

3.3 Опис прийнятих під час реалізації рішень

На основі структури схеми бази даних було реалізовано програмну репрезентацію в об'єктно-орієнтованому стилі для подальшого перекладу шаблону на сервер MySQL.

Клас користувача User включає в себе наступні атрибути:

- id – ключ користувача;
- username – юзернейм користувача;
- name – ім'я користувача;
- surname – прізвище користувача;
- email_address – електронна пошта користувача;
- password – пароль;
- uuid – складний ключ користувача;
- role_id – ключ ролі користувача;
- user_game_to – зв'язок з таблицею UserGame.

Клас гри Game включає в себе наступні атрибути:

- id – ключ гри;
- title – назва гри;
- description – анотація гри;
- uuid – складний ключ гри;
- is_visible – прапор видимості гри;
- is_delete – прапор видалення гри;
- price – ціна гри;
- user_game_to – зв'язок з таблицею UserGame;
- genre_subgenre_to – зв'язок з таблицею GameGenreSubgenre.

Клас ролі Role включає в себе наступні атрибути:

- id – ключ ролі;
- title – назва ролі;
- code – умовний код ролі;
- user_role_to – зв'язок з таблицею User.

Клас коментаря Comment включає в себе наступні атрибути:

- id – ключ ролі;
- uuid – складний ключ коментаря;
- text – зміст коментаря;

- `created_date` – дата створення коментаря;
- `reply_to_message_id` – ключ коментаря, якщо на нього відповідали;

- `user_id_from` – умовний зв'язок з таблицею `User`;
- `game_id_from` – умовний зв'язок з таблицею `Game`.

Клас жанру `Genre` включає в себе наступні атрибути:

- `id` – ключ жанру;
- `title` – назва жанру.

Клас піджанру `Subgenre` включає в себе наступні атрибути:

- `id` – ключ піджанру;
- `title` – назва піджанру.

Клас збирання жанрів і піджанрів `GenreSubgenre` включає в себе наступні атрибути:

- `id` – ключ;
- `genre_id` – ключ жанру;
- `subgenre_id` – ключ піджанру;
- `genre_subgenre_to` – зв'язок з таблицею `GameGenreSubgenre`.

Клас збирання користувачів та ігор `UserGame` включає в себе наступні атрибути:

- `user_id` – ключ користувача;
- `game_id` – ключ гри;
- `count` – кількість копій ігор.

Клас збирання зв'язка жанрів та піджанрів з грою `GameGenreSubgenre` включає в себе наступні атрибути:

- `genre_subgenre_id` – ключ `GenreSubgenre`;
- `game_id` – ключ гри.

Для доступу на вебресурсі були реалізовані відповідні класи з HTTP методами (GET, POST, DELETE).

Клас `AddGame` включає в себе наступні методи:

- `def get(self):` – повертає сторінку додавання гри з усіма включеними налаштуваннями;

- `def post(self):` – перевіряє отримані дані та в позитивному випадку додає нову гру;

- `def delete(self):` – видаляє гру;

Клас `BuyGame` включає в себе метод `def post(self):`, що додає гру в кошик і перенаправляє користувача на неї.

Клас `Carts` включає в себе наступні методи:

- `def get(self):` – повертає кошик з покупками;

- `def post(self):` – видаляє певну гру з кошика з покупками.

Клас `EditGame` включає в себе наступні методи:

- `def get(self):` – повертає сторінку налаштування гри з усіма включеними налаштуваннями;

- `def post(self):` – перевіряє отриманні дані та в позитивному випадку змінює дані гри.

Клас `Game` включає в себе наступні методи:

- `def get(self):` – повертає сторінку гри;

- `def post(self):` – додає коментар до гри;

- `def delete(self):` – видаляє коментар зі сторінки гри.

Клас `Login` включає в себе наступні методи:

- `def get(self):` – перенаправляє запит до головної сторінки.

- `def post(self):` – перевіряє дані користувача та в позитивному випадку авторизує його.

Клас `Logout` включає в себе наступні методи:

- `def get(self):` – перенаправляє запит до головної сторінки;

- `def post(self):` – відмінює авторизацію користувача.

Клас `Main` включає в себе метод `def get(self):`, що повертає головну сторінку.

Клас `Order` включає в себе наступні методи:

- `def get(self):` – повертає сторінку оформлення покупки;
- `def post(self):` – перевіряє відправлену форму покупки та в позитивному випадку підтверджує покупку.

Клас `Register` включає в себе наступні методи:

- `def get(self):` – повертає форму реєстрації користувача;
- `def post(self):` – перевіряє відправлену форму реєстрації та в позитивному випадку створює обліковий запис нового користувача

Клас `User` включає в себе наступні методи:

- `def get(self):` – повертає сторінку користувача;
- `def post(self):` – перевіряє дані та в позитивному випадку змінює інформацію про користувача.

3.4 Висновки за розділом 3

У цьому розділі було складено і реалізовано алгоритм функціонування програмного забезпечення. Було створено структуру вебсервісу та описано функціонал кожного модуля системи. Також були описані призначення класів та функцій, що вони реалізують.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ

4.1 Призначення програми

Розроблений вебсервіс призначений для продажу цифрових копій комп'ютерних ігор і являє собою інтернет-магазин з розподіленою системою ролей доступу серед користувачів. Програма включає в себе зручний інтерфейс адміністрування лотів, пошук з налаштуваннями за жанровою специфікою, система купівлі та доставки, можливість оцінювати, коментувати та ділитися своєю думкою з спільнотою на рахунок товару.

4.2 Умови виконання програми

Технічні вимоги сервера:

- жорсткий диск з вільним простором – 1,5 Гб;
- оперативна пам'ять – 4 Гб;
- процесор: 4 ядра, 2,8 ГГц частота.

У даній технічній реалізації встановлено безкоштовний тариф AWS S3 Storage, що обмежений завантаженням малої кількості зображень. Для повноцінної версії знадобиться зміна тарифного плану.

Для справної роботи вебсервісу на стороні користувача, потрібний сучасний браузер з можливістю підключення до мережі Інтернет.

4.3 Виконання програми

Точкою початку роботи із сервісом є головна сторінка. На ній знаходяться основний функціонал програми та посилання на інші сторінки. У верхньому лівому куті знаходиться логотип магазину з посиланням на головну сторінку. За умови неавторизованого користувача сторінка має такий вигляд (рис. 4.1).

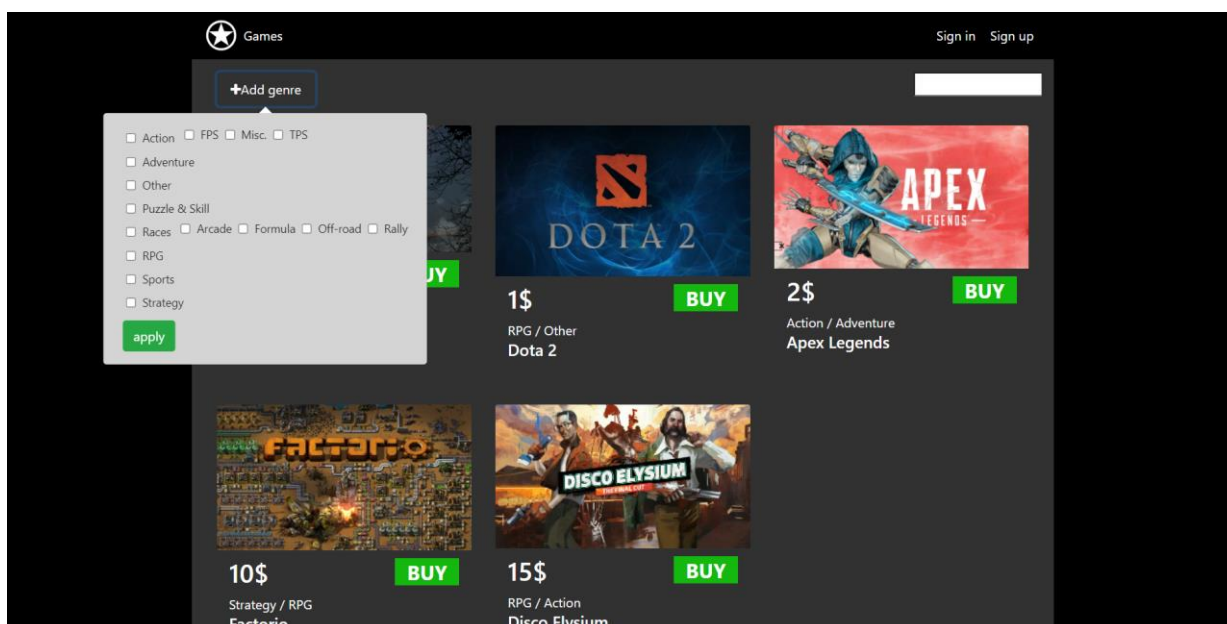


Рисунок 4.1 – Вигляд головної сторінки магазину неавторизованого користувача

У верхньому правому куті знаходяться кнопки «Sign in», при натисканні на яку, з'являється спливаюче вікно авторизації, і «Sign up» - посилання на сторінку реєстрації користувача.

В області, що знаходиться нижче є кнопка «Add genre», що дозволяє фільтрувати продукти за жанрами та піджанрами до яких вони відносяться, вибираючи їх з списку.

У протилежному боці знаходиться форма пошуку за назвою продукту, алгоритм ігнорує табуляцію і включає пошук як за повною назвою, так і по частині.

Далі на сторінці представлений список існуючих товарів, що схильний до фільтрації та пошуку в залежності від форм вище. Кожен лот включає картинку мініатюру гри, вартість (у доларах), список жанрів до яких відноситься назва і кнопку для покупки.

При авторизації користувача у верхньому правому куті сторінки замість кнопок авторизації та реєстрації з'являється мініатюра іконки

профілю, ім'я користувача, його кошик та кнопка виходу із сесії.

За наявності прав адміністратора або менеджера магазину у користувача з'являється кнопка «Add Game» з посиланням на форму додавання нової гри. Також при наведенні курсору на лот з'являється можливість змінити інформацію про продукт або запустити процес його видалення (рис. 4.2).

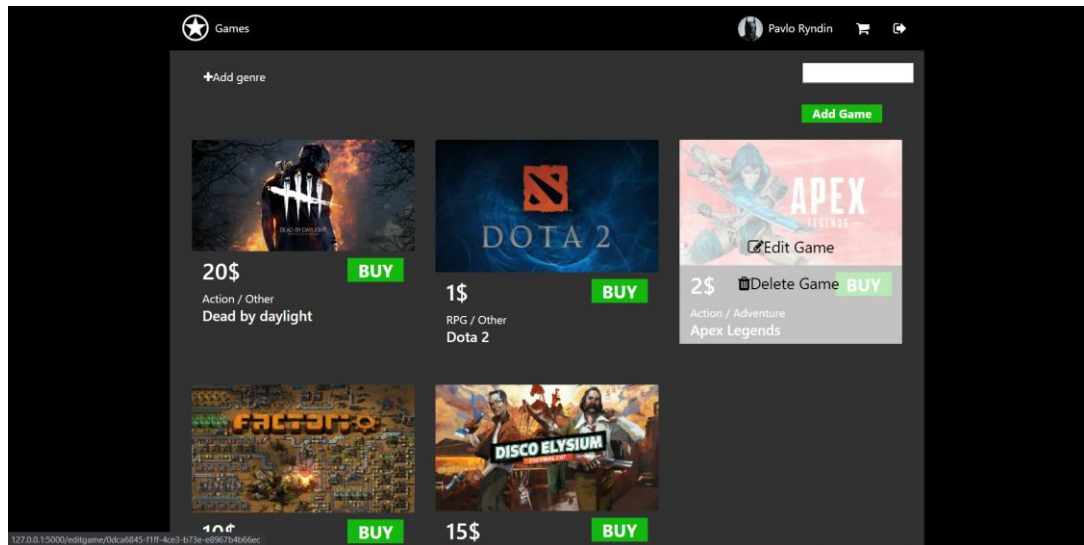


Рисунок 4.2 – Вигляд головної сторінки магазину з боку адміністратора

Реєстрація являє собою форму з шістьма полями: Ім'я, Прізвище, Нік, пошта, пароль та повторення пароля для перевірки. Після натискання кнопки «Register», якщо всі поля було введено коректно, створюється профіль користувача (рис. 4.3).

Рисунок 4.3 – Вигляд сторінки реєстрації

Авторизація являє собою форму з двома полями: нік користувача та пароль. Також є можливість зберегти сесію на тривалий період часу, відзначивши галочку під пунктом «Remember me» (рис. 4.4).

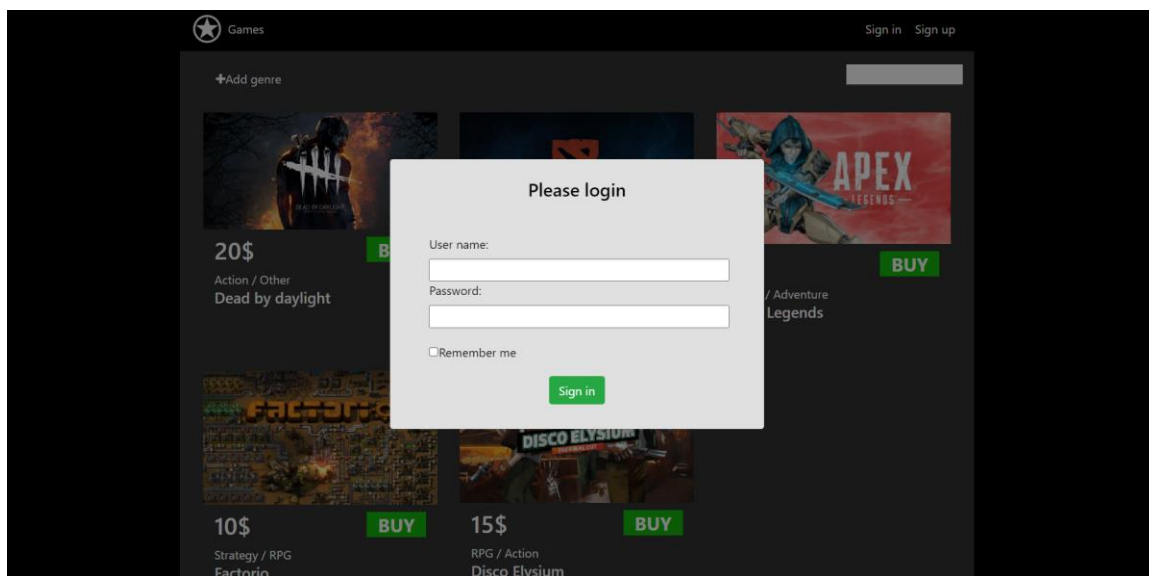


Рисунок 4.4 – Вигляд форми авторизації

При перегляді лота гри на сторінці є картинка гри, назва, ціна, опис, список жанрів та поджанрів до яких належить, кнопка додавання у корзину. У нижній частині знаходиться відсік для коментарів (рис. 4.5).



Рисунок 4.5 – Вигляд сторінки товару

Кожен коментар, залишений користувачем, має дату створення, посилання на профіль коментатора і текст коментаря. Також є можливість видалення коментаря користувачем, що його залишив, або адміністратором (рис. 4.6).

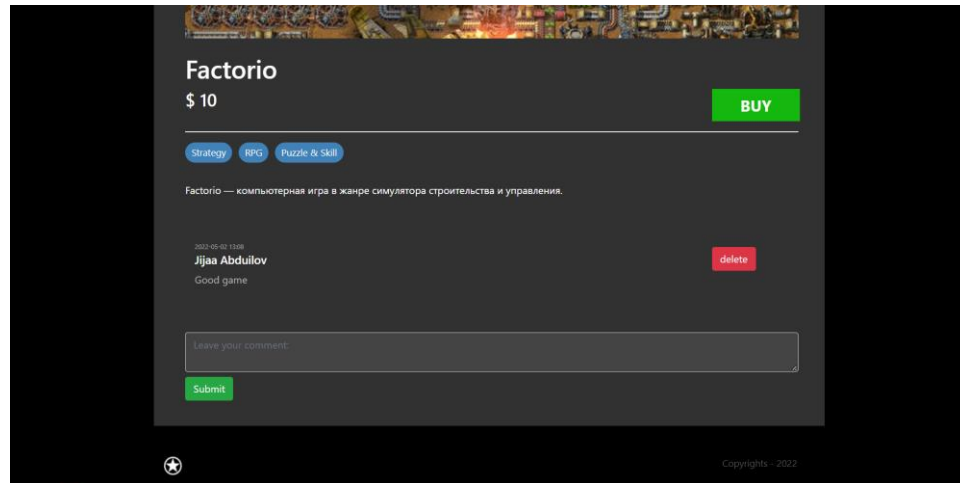


Рисунок 4.6 – Вигляд коментарів на сторінці товару

При перегляді кошика можна побачити всі ігри, які були в нього додані. Кожен лот має посилання на власну сторінку в магазині, ціну, інтерактивний лічильник бажаної кількості копій товару, загальну ціну всіх копій, та віджет у формі хрестика для видалення. У верхньому правому куті відображається загальна ціна за всі ігри та кнопка «Proceed» для оформлення покупки (рис. 4.7).

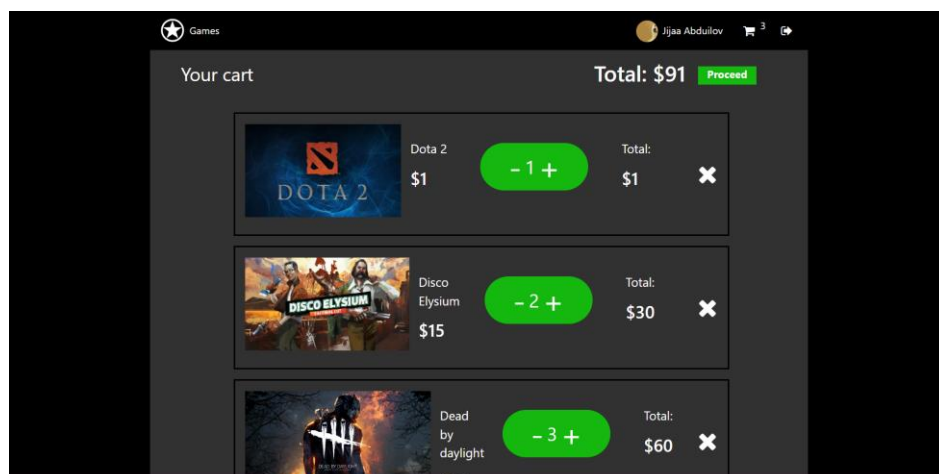


Рисунок 4.7 – Вигляд сторінки кошика з покупками

Сторінка оформлення замовлення складається з форми для заповнення та містить наступні поля: Ім'я, Прізвище, Пошту, Телефон, Тип оплати, що опційно налаштовується залежно від реалізації зовнішніх ресурсів онлайн-банкінгу, коментар до покупки. При натисканні на кнопку «Order» оформляється покупка і на пошту користувачеві надходить товар (рис. 4.8).

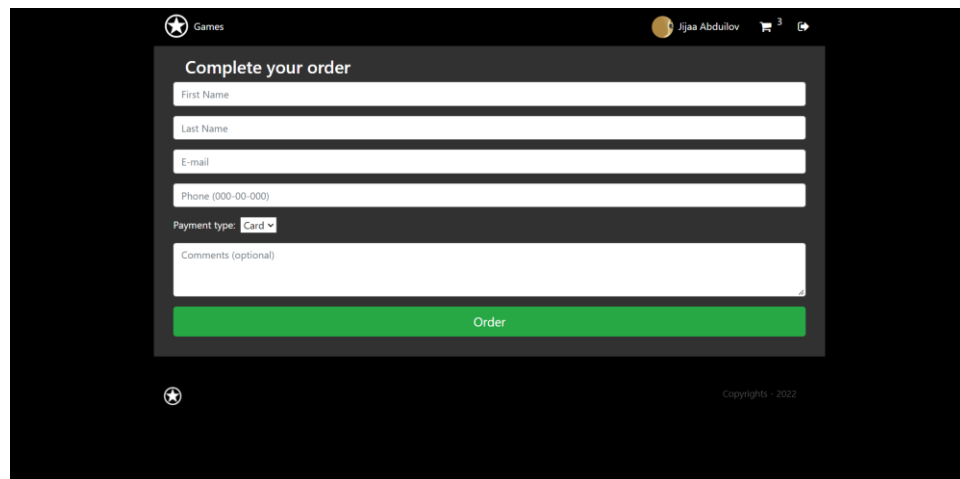
The screenshot shows a web interface for a game store. At the top, there's a header with a star icon and the word 'Games', a user profile icon for 'Jjaa Abdulov', a shopping cart icon with '3' items, and a share icon. The main content area is titled 'Complete your order'. It contains several input fields: 'First Name', 'Last Name', 'E-mail', and 'Phone (000-00-000)'. Below these is a 'Payment type' dropdown menu currently set to 'Card'. There is also a 'Comments (optional)' text area. At the bottom of the form is a large green button labeled 'Order'. The footer of the page has a star icon on the left and 'Copyrights - 2022' on the right.

Рисунок 4.8 – Вигляд сторінки оформлення товару

При перегляді власного профілю або при перегляді з правами адміністратора, можна переглянути або змінити наступну інформацію: Ім'я, Прізвище, Пошта, пароль, фотографія. При натисканні кнопки «Save» внесені зміни зберігаються до БД (рис. 4.9).

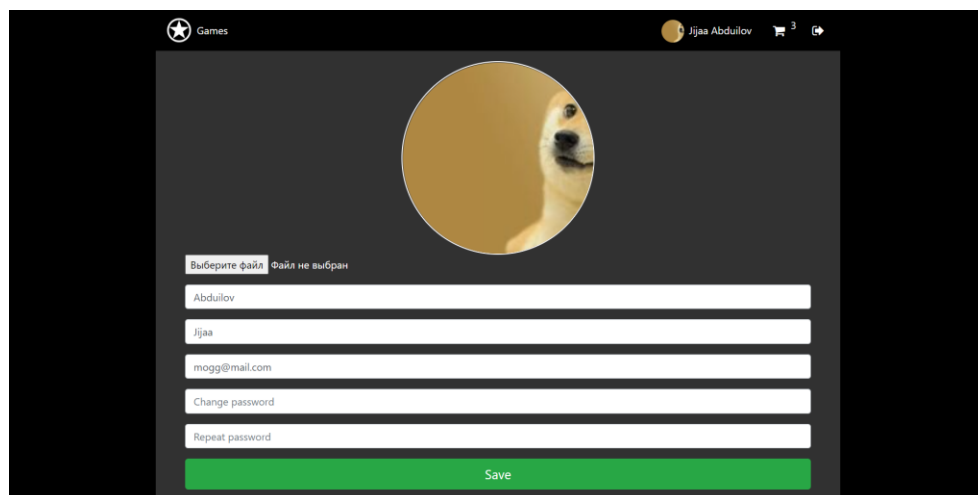
The screenshot shows a user profile page. The header is identical to the previous screenshot. The main content area features a large circular profile picture of a dog. Below the picture is a file upload section with the text 'Выберите файл' and 'Файл не выбран'. Underneath are five input fields: the first contains 'Abdulov', the second 'Jjaa', the third 'mogg@mail.com', the fourth is labeled 'Change password', and the fifth is labeled 'Repeat password'. At the bottom of the form is a large green button labeled 'Save'.

Рисунок 4.9 – Вигляд сторінки профілю користувача

Сторінки додавання та зміни товару дуже схожі своїм функціоналом. На них є форма з наступними полями Ім'я, опис, вартість, завантаження обкладинки, налаштування жанрів і поджанрів. Після натискання відповідних кнопок «Create» і «Save» зміни зберігаються, і лот створюється. (рис. 4.10).

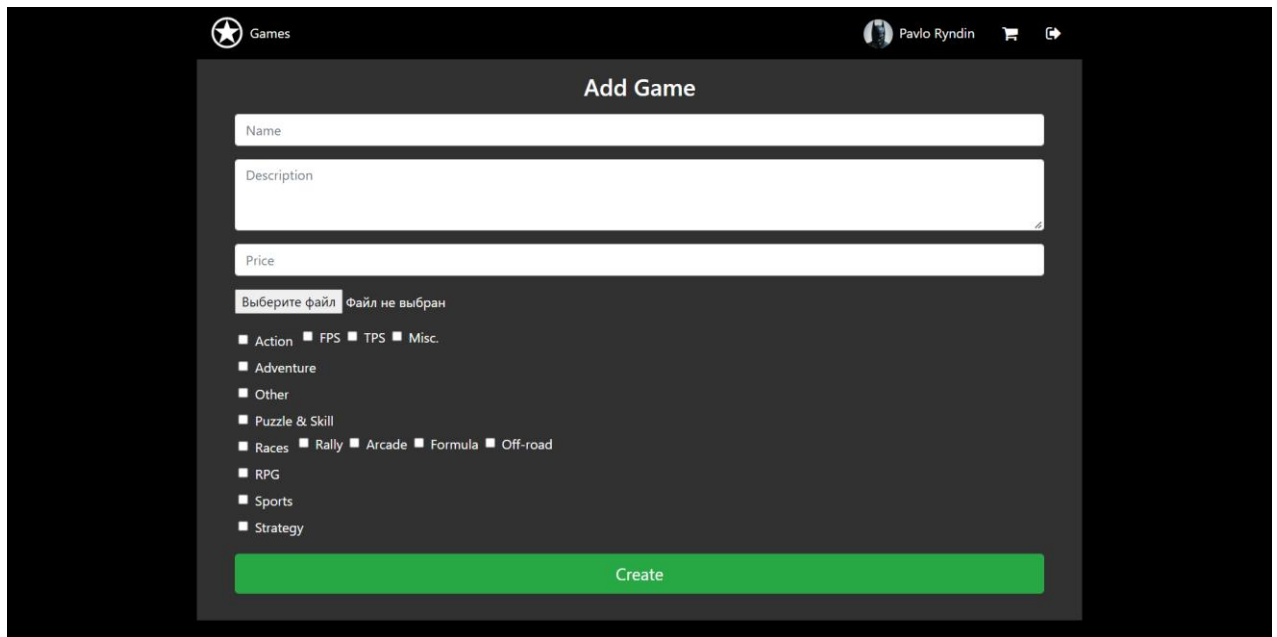
The image shows a web application interface for adding a new game. At the top, there is a navigation bar with a star icon and the word 'Games' on the left, and a user profile icon with the name 'Pavlo Ryndin' on the right. The main content area is titled 'Add Game'. It contains a form with several input fields: 'Name', 'Description', and 'Price'. Below these fields is a file upload section with a button labeled 'Выберите файл' (Choose file) and a status 'Файл не выбран' (File not selected). Underneath the file upload section is a list of game genres and sub-genres, each with a checkbox: Action, FPS, TPS, Misc., Adventure, Other, Puzzle & Skill, Races, Rally, Arcade, Formula, Off-road, RPG, Sports, and Strategy. At the bottom of the form is a large green button labeled 'Create'.

Рисунок 4.10 – Вигляд сторінки додавання лоту

4.4 Тестування розробленої програми

Модулем для тестування було обрана бібліотека Pytest. Для автоматизації тестування був обраний сервіс тестування програмного забезпечення Travis CI. За допомогою нього було налаштовано локальне середовище проекту та інфраструктуру тестування прямо з GitHub програми. Результат тестування наведено на рис. 4.11.

```

tests/test_addgame.py ... [ 10%]
tests/test_buygame.py .. [ 16%]
tests/test_carts.py .... [ 30%]
tests/test_editgame.py ... [ 40%]
tests/test_game.py ..... [ 53%]
tests/test_login.py ... [ 63%]
tests/test_main.py .. [ 70%]
tests/test_order.py ... [ 80%]
tests/test_register.py ... [ 90%]
tests/test_user.py ... [100%]

----- coverage: platform linux, python 3.8.12-final-0 -----
Name                                Stmts  Miss  Cover
-----
src/__init__.py                      39      4    90%
src/aws_func.py                     18      6    67%
src/database/__init__.py              0      0   100%
src/database/inserts.py             133    133     0%
src/database/models.py              102     18    82%
src/resources/addgame.py             87     61    30%
src/resources/buygame.py            26      9    65%
src/resources/carts.py              48     17    65%
src/resources/editgame.py           81     57    30%
src/resources/game.py               82     47    43%
src/resources/login.py              32      4    88%
src/resources/main.py              62     23    63%
src/resources/order.py              60     20    67%
src/resources/register.py           61     19    69%
src/resources/user.py               89     60    33%
src/routes.py                       26      1    96%
src/token.py                        63     17    73%
-----
TOTAL                               1009    496    51%

===== 30 passed in 10.63s =====
The command "pytest --cov=src tests/" exited with 0.

```

Рисунок 4.11 – Результати автотестування

4.5 Висновки за розділом 4

За результатами розділу було визначено технічні умови використання розробленої програми, були проаналізовані та розглянуті ресурси та функції передбачені даною технічною реалізацією. Були розроблені інтеграційні тести та налаштоване середовище Travis CI для автоматичного їх запуску та аналізу.

ВИСНОВКИ

За результатами виконання дипломної кваліфікаційної роботи виконано такі задачі:

- проведено аналіз концепції продажу цифрових товарів, розглянуто підходи до реалізації онлайн-магазинів, проведено порівняння з програмними реалізаціями;
- визначено функціональні вимоги для реалізації програмного забезпечення, обрано мову програмування Python, Flask фреймворк, СКБД MySQL, створено структуру бази даних проєкту;
- реалізовано весь запланований функціонал програмного забезпечення, створено структуру вебсервісу та описано кожний модуль системи. Також описані призначення класів та функцій та їх реалізацій;
- проведено тестування розробленої системи. Модулем для тестування було обрана бібліотека Pytest. Для автоматизації тестування був обраний сервіс тестування програмного забезпечення Travis CI.

Розроблений вебсервіс призначений для продажу цифрових копій комп'ютерних ігор і являє собою інтернет-магазин з розподіленою системою ролей доступу серед користувачів. Програма включає в себе зручний інтерфейс адміністрування лотів, пошук з налаштуваннями за жанровою специфікою, систему купівлі та доставки, можливість оцінювати, коментувати та ділитися з спільнотою відгуками стосовно товару.

Завдання дипломної кваліфікаційної роботи в результаті повністю виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. GOG GAME STORE [Electronic resource]. – Access mode: <https://www.gog.com/en>.
2. ABOUT GOG GAME STORE [Electronic resource]. – Access mode: https://www.gog.com/en/about_gog.
3. HUMBLEBUNDLE GAME STORE [Electronic resource]. – Access mode: <https://www.humblebundle.com/store>.
4. GREEN MAN GAMING STORE [Electronic resource]. – Access mode: <https://www.greenmangaming.com/ru/>.
5. Python Programming Language [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/python-programming-language/>.
6. Lutz, M. Learning Python: Powerful Object-Oriented Programming / M. Lutz. – 2000. – 1650 p.
7. Grinberg, M. Flask Web Development: Developing Web Applications with Python / M. Grinberg. – 2014. – 666 p.
8. William, S. Django for Beginners: Build websites with Python and Django / S. William. – 2018. – 317 p.
9. Flask vs Django: What's the Difference Between Flask & Django? [Electronic resource]. – Access mode: <https://www.guru99.com/flask-vs-django.html>.
10. Crockford, D. How JavaScript Works / D. Crockford. – 2018. – 279 p.
11. MySQL vs PostgreSQL -- Choose the Right Database for Your Project [Electronic resource]. – Access mode: <https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres>.
12. Derry, R. Aws: AMAZON WEB SERVICES: The Complete Guide / R. Derry. – 2019. – 400 p.
13. Percival, H. Python. Test Driven Development / H. Percival. – 2018. – 622 p.
14. GitHub [Electronic resource]. – Access mode:

<https://github.com/about>.

15. PostgreSQL vs. MySQL: What's the Difference? [Electronic resource]. – Access mode: <https://www.ibm.com/cloud/blog/postgresql-vs-mysql-whats-the-difference>.

ДОДАТОК А
Технічне завдання

Вступ

Розроблений вебсервіс спрямований на комерційну реалізацію цифрових копій комп'ютерних ігор. Також сервіс передбачає можливість прямого зв'язку покупця з продавцем або посередником у магазині і надає необхідний для цього функціонал.

A.1 Підстави для розробки

Підставою для розробки було завдання на дипломну кваліфікаційну роботу за темою «Програмна реалізація інтернет-магазину цифрових товарів» на основі наказу №102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Розроблений вебсервіс призначений для продажу цифрових копій комп'ютерних ігор і являє собою інтернет-магазин з розподіленою системою ролей доступу серед користувачів. Програма включає в себе зручний інтерфейс адміністрування лотів, пошук з налаштуваннями за жанровою специфікою, систему купівлі та доставки, можливість оцінювати, коментувати та ділитися з спільнотою відгуками стосовно товару.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Сервіс повинен забезпечувати наступний функціонал адміністратора:

- відображення всієї колекції ігор у магазині;
- можливість пошуку ігор за назвою;
- фільтрація ігор за жанрами;

- фільтрація ігор за піджанрами;
- можливість реєстрації нового користувача;
- входження до системи за допомогою облікових даних;
- відображення особистих облікових даних;
- відображення облікових даних інших користувачів;
- можливість змінювати особисті облікові дані;
- можливість змінювати облікові дані інших користувачів;
- можливість змінювати ролі іншим користувачам;
- перегляд лоту гри;
- створення нового лоту;
- додавання інформації про лот;
- налаштування жанрів та піджанрів гри;
- додавання картинки та мініатюри до гри;
- зміна інформації про лот;
- видалення лоту;
- перегляд коментарів інших користувачів під грою;
- залишення своїх коментарів;
- видалення власних коментарів ;
- видалення коментарів інших користувачів;
- можливість відповідати на коментарі інших користувачів;
- можливість відновлювати коментарі.

Функціональні вимоги менеджера є наступними:

- відображення всієї колекції ігор у магазині;
- можливість пошуку ігор за назвою;
- фільтрація ігор за жанрами;
- фільтрація ігор за піджанрами;
- можливість реєстрації нового користувача;
- входження до системи за допомогою облікових даних;
- відображення особистих облікових даних;
- відображення облікових даних інших користувачів;

- можливість змінювати особисті облікові дані;
- можливість змінювати облікові дані інших користувачів;
- перегляд лоту гри;
- створення нового лоту;
- додавання інформації про лот;
- налаштування жанрів та піджанрів гри;
- додавання картинки та мініатюри до гри;
- зміна інформації про лот;
- видалення лоту;
- перегляд коментарів інших користувачів під грою;
- залишення своїх коментарів;
- видалення власних коментарів ;
- видалення коментарів інших користувачів;
- можливість відповідати на коментарі інших користувачів;
- можливість відновлювати коментарі.

Функціональні вимоги покупця включають:

- відображення всієї колекції ігор у магазині;
- можливість пошуку ігор за назвою;
- фільтрація ігор за жанрами;
- фільтрація ігор за піджанрами;
- можливість реєстрації нового користувача;
- входження до системи за допомогою облікових даних;
- відображення особистих облікових даних;
- відображення облікових даних інших користувачів;
- можливість змінювати особисті облікові дані;
- перегляд лоту гри;
- перегляд кошика покупок;
- додавання товарів до кошику покупок;
- видалення товарів з кошика покупок;
- оформлення та оплата покупки;

- перегляд історії своїх покупок;
- можливість оцінювати лот;
- перегляд коментарів інших користувачів під грою;
- залишення своїх коментарів;
- видалення власних коментарів;
- можливість відповідати на коментарі інших користувачів.

A.3.2 Вимоги до надійності

Надійність працездатності сервісу залежить від правильності налаштування сервера бази даних з урахуванням її програмної реалізації у системі. Так само система включає статичні файли, що зберігаються на AWS Cloud серверах, що вимагають налаштування тарифу використання даних.

A.3.3 Умови експлуатації

Для коректної роботи системи потрібен сервер для програмної частини програми, сервер бази даних та налаштування тарифу AWS Cloud.

A.3.4 Вимоги до складу та параметрів технічних засобів

Технічні вимоги сервера:

- жорсткий диск з вільним простором – 1,5 Гб;
- оперативна пам'ять – 4 Гб;
- процесор: 4 ядра, 2,8 ГГц частота.

У даній технічній реалізації встановлено безкоштовний тариф AWS S3 Storage, що обмежений завантаженням малої кількості зображень. Для продакшн версії знадобиться зміна тарифного плану.

Для справної роботи вебсервісу на стороні користувача, потрібний сучасний браузер з можливістю підключення до мережі Інтернет.

A.4 Порядок контролю та приймання

Контроль за виконанням дипломної кваліфікаційної роботи покладається на керівника. Остаточне рішення приймається екзаменаційною комісією під час захисту роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла wsgi.py

```
from src import app

#app starting here
if __name__ == "__main__":
    app.run()
```

Б.2 Текст файла __init__.py

```
from flask import Flask
from flask_restful import Api
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_bcrypt import Bcrypt
from flask_admin import Admin
from flask_admin.contrib.sqla import ModelView
from simple_settings import settings
from logging import FileHandler, WARNING
import boto3

from os import path, environ

#
# Setting and adding the required variables
#

__author__ = 'Pavlo Ryndin'

#configuring paths of templates and static
temp_dir = path.abspath(path.dirname(__file__))
app = Flask(__name__, static_folder=path.join(temp_dir, 'static'),
            template_folder=path.join(temp_dir, 'templates'))

#configuring environment
try:
    app.config.update(**settings.as_dict())
except:
    app.config.from_pyfile(path.join(path.dirname(
        temp_dir), 'configs', 'common_deploy.py'))
    print("Configuration has been installed as \'common_deploy\'")

#services declarations
api = Api(app)
db = SQLAlchemy(app)
migrate = Migrate(app, db)
bcrypt = Bcrypt(app)
```

```

try:
    aws_client = boto3.client('s3',
aws_access_key_id=environ["ACCESS_KEY_ID"].split('\')[1],

aws_secret_access_key=environ["SECRET_ACCESS_KEY"].split('\')[1])
    # aws_client = boto3.client('s3',
aws_access_key_id=environ["ACCESS_KEY_ID"],
    #
aws_secret_access_key=environ["SECRET_ACCESS_KEY"])
    f = True
    for i in aws_client.list_buckets()['Buckets']:
        if i["Name"] == "gamestorebucket":
            f = False
    if f:
        raise Exception("No such a bucket!")
except Exception as e:
    print(e)

#logging customization
if not app.config['DEBUG']:
    file_error_handler = FileHandler('logs/errorlog.txt')
    file_error_handler.setLevel(WARNING)
    app.logger.addHandler(file_error_handler)

from src import routes
from src.database import models

```

Б.3 Текст файла routes.py

```

from flask import render_template
from src import api, app

from src.resources.main import Main
from src.resources.register import Register
from src.resources.login import Login, Logout
from src.resources.addgame import AddGame
from src.resources.game import Game
from src.resources.buygame import BuyGame
from src.resources.carts import Carts
from src.resources.order import Order
from src.resources.editgame import EditGame
from src.resources.user import User

#
# Declarations of app routes in type of rest web-architecture
#

# routes

```

```

api.add_resource(Main, '/', '/main', strict_slashes=False)
api.add_resource(Register, '/register', strict_slashes=False)
api.add_resource(Login, '/login', strict_slashes=False)
api.add_resource(Logout, '/logout', strict_slashes=False)
api.add_resource(AddGame, '/addgame', strict_slashes=False)
api.add_resource(Game, '/game/<uuid>', strict_slashes=False)
api.add_resource(BuyGame, '/buygame', strict_slashes=False)
api.add_resource(Carts, '/carts', strict_slashes=False)
api.add_resource(Order, '/order', strict_slashes=False)
api.add_resource(EditGame, '/editgame/<uuid>', strict_slashes=False)
api.add_resource(User, '/user/<uuid>', strict_slashes=False)

# handling of 404 page error
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

```

Б.4 Текст файла `aws_func.py`

```

from src import aws_client
from io import BytesIO
import base64
from src import aws_client

def upload_img(file, object_name):
    bytesio = BytesIO(file.read())
    ext = file.filename[-4:]
    aws_client.upload_fileobj(bytesio, "gamestorebucket",
    object_name+ext, ExtraArgs=None)

def upload_user_img(file, object_name):
    bytesio = BytesIO(file.read())
    ext = file.filename[-4:]
    aws_client.upload_fileobj(bytesio, "gamestoreuserbucket",
    object_name+ext, ExtraArgs=None)

def get_aws_image(bucket, obj):
    a_file = BytesIO()
    aws_client.download_fileobj(bucket, obj+".jpg", a_file)
    a_file.seek(0)
    str_equivalent_image =
base64.b64encode(a_file.getvalue()).decode()
    return ("data:image/png;base64," + str_equivalent_image)

```

Б.5 Текст файла `token.py`

```

import jwt
from src import app

```

```

from flask import request
from flask import flash, redirect, request, url_for
from src.database.models import User, Role, Game
from functools import wraps

def get_user_by_token():
    token = request.cookies.get('token')
    uuid = jwt.decode(token, app.config['SECRET_KEY'],
algorithms=["HS256"])[ 'user_id' ]
    user = User.query.filter_by(uuid=uuid).first()
    return user

def token_required(func):
    @wraps(func)
    def wrapper(self, *args, **kwargs):
        #wrapper of current func

        #take csrf-token from cookies
        token = request.cookies.get('token')
        if not token:
            #returning login page
            flash("Authentication required", category='danger')
            return redirect(url_for('login'))
        #decoding token
        try:
            uuid = jwt.decode(token, app.config['SECRET_KEY'],
algorithms=["HS256"])[ 'user_id' ]
        except:
            #returning login page
            flash("Token timeout", category='danger')
            return redirect(url_for('login'))
        #get current user
        user = User.query.filter_by(uuid=uuid).first()
        if not user:
            #returning login page
            flash("Profile error", category='danger')
            return redirect(url_for('login'))
        return func(self, *args, **kwargs)
    return wrapper

def role_handler(roles):
    def dec(func):
        @wraps(func)
        def wrapper(self, *args, **kwargs):
            #wrapper of current func
            try:
                user = get_user_by_token()
            except:
                if not 0 in roles:
                    flash("You have no permission for this page!",
category='danger')

```

```

        return redirect(url_for('main'))
    else:
        role =
Role.query.join(User).filter_by(username=user.username).first()
        if not role.code in roles:
            flash("You have no permission for this page!",
category='danger')
            return redirect(url_for('main'))
        return func(self, *args, **kwargs)
    return wrapper
return dec

def get_games():
    g_obj_list = []
    gamelist = request.cookies.get('gamelist')
    if not gamelist:
        return g_obj_list
    m = gamelist.split('&')

    for i in m:
        game = Game.query.filter_by(uuid=i).first()
        if game:
            f = True
            for j in g_obj_list:
                if j==game:
                    f=False
                    break
            if f:
                g_obj_list.append(game)
    return g_obj_list

```

Б.6 Текст файла models.py

```

import uuid
from flask_bcrypt import check_password_hash, generate_password_hash
from src import db
import datetime

class UserGame(db.Model):
    __tablename__ = 'user_game'
    user_id = db.Column(db.ForeignKey('user.id'), primary_key=True)
    game_id = db.Column(db.ForeignKey('game.id'), primary_key=True)
    count = db.Column(db.Integer, default=1)

class GameGenreSubgenre(db.Model):
    __tablename__ = 'game_genre_subgenre'
    genre_subgenre_id = db.Column(db.ForeignKey('genre_subgenre.id'),
primary_key=True)
    game_id = db.Column(db.ForeignKey('game.id'), primary_key=True)

```

```

    def __init__(self, genre_subgenre_id, game_id):
        self.genre_subgenre_id = genre_subgenre_id
        self.game_id = game_id

class Comment(db.Model):
    __tablename__ = 'comment'

    id = db.Column(db.Integer, primary_key=True)
    uuid = db.Column(db.String(36), unique=True)
    text = db.Column(db.String(600), nullable=False)
    created_date = db.Column(db.DateTime,
default=datetime.datetime.utcnow, nullable=False)
    reply_to_message_id = db.Column(db.Integer, default=None)
    user_id_from = db.Column(db.Integer, db.ForeignKey("user.id"),
nullable=False)
    game_id_from = db.Column(db.Integer, db.ForeignKey("game.id"),
nullable=False)

    def __init__(self, text, reply_to_message_id, user_id_from,
game_id_from):
        self.uuid = str(uuid.uuid4())
        self.text = text
        self.reply_to_message_id = reply_to_message_id
        self.user_id_from = user_id_from
        self.game_id_from = game_id_from

class Role(db.Model):
    """
    Role model
    """
    __tablename__ = 'role'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(40), nullable=False, unique=True)
    code = db.Column(db.Integer, nullable=False)
    user_role_to = db.relationship("User", backref='role', lazy=True)

    def __repr__(self):
        return f'Role({self.title})'

class User(db.Model):
    """
    User model
    """
    __tablename__ = 'user'

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(30), nullable=False, unique=True)
    name = db.Column(db.String(30), nullable=False)
    surname = db.Column(db.String(30), nullable=False)

```

```

        email_address = db.Column(db.String(50), nullable=False,
unique=True)
        password = db.Column(db.String(100), nullable=False)
        uuid = db.Column(db.String(36), unique=True)
        role_id = db.Column(db.Integer, db.ForeignKey('role.id'),
nullable=False)

        user_game_to = db.relationship("UserGame",
backref='user',lazy=True)

    def __init__(self, username, name, surname, email_address,
password, code):
        self.username = username
        self.name = name
        self.surname = surname
        self.email_address = email_address
        self.password =
generate_password_hash(password).decode('utf8')
        self.uuid = str(uuid.uuid4())

        role = db.session.query(Role).filter(Role.code ==
code).first()
        self.role_id = role.id

    def __repr__(self):
        return f'User({self.surname}, {self.name})'

    def check_password(self, attempted_password):
        """
        Compare passwords hash
        return bool
        """
        return check_password_hash(self.password, attempted_password)

class Game(db.Model):
    """
    Game model
    """
    __tablename__ = 'game'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(40), nullable=False, unique=True)
    description = db.Column(db.String(1000), nullable=False)
    uuid = db.Column(db.String(36), unique=True)
    is_visible = db.Column(db.Boolean, default = True)
    is_delete = db.Column(db.Boolean, default = False)
    price = db.Column(db.Integer, nullable=False)

    user_game_to = db.relationship("UserGame",
backref='game',lazy=True)

```

```

        genre_subgenre_to = db.relationship("GameGenreSubgenre",
backref='game', lazy=True)

    def __init__(self, title, description, price, is_visible=True,
is_delete=False):
        self.title = title
        self.description = description
        self.price = price
        self.uuid = str(uuid.uuid4())
        self.is_visible = is_visible
        self.is_delete = is_delete

    def __repr__(self):
        return f'Project({self.title})'

class GenreSubgenre(db.Model):
    """
    GenreSubgenre model
    """
    __tablename__ = 'genre_subgenre'
    id = db.Column(db.Integer, primary_key=True)
    genre_id = db.Column(db.Integer, db.ForeignKey("genre.id"),
nullable=True)
    subgenre_id = db.Column(db.Integer, db.ForeignKey("subgenre.id"),
nullable=True)

    genre_subgenre_to = db.relationship("GameGenreSubgenre",
backref='genre_subgenre', lazy=True)

    __table_args__ = (db.UniqueConstraint(genre_id, subgenre_id),)

class Genre(db.Model):
    """
    Genre model
    """
    __tablename__ = 'genre'

    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(40), nullable=False, unique=True)

    def __repr__(self):
        return f'Genre({self.title})'

class Subgenre(db.Model):
    """
    Genre model
    """
    __tablename__ = 'subgenre'

    id = db.Column(db.Integer, primary_key=True)

```

```

title = db.Column(db.String(40), nullable=False, unique=True)

def __repr__(self):
    return f'Subgenre({self.title})'

```

Б.7 Текст файла addgame.py

```

from flask import make_response, render_template, request, flash,
redirect, url_for
from flask_restful import Resource
from src import db
from src.database.models import GameGenreSubgenre, Genre,
GenreSubgenre, Subgenre, Game
from src.token import get_user_by_token, token_required, role_handler,
get_games
from src.aws_func import upload_img, get_aws_image

class AddGame(Resource):
    RESOURCE_ROLES = [2,3]

    @role_handler(RESOURCE_ROLES)
    @token_required
    def get(self):
        try:
            #get user and user img
            user = get_user_by_token()
            try:
                user_icon = get_aws_image("gamestoreuserbucket",
user.uuid)
            except:
                user_icon=False
        except:
            # return page with no user session
            return make_response(render_template("main.html",), 200)
        dict_genre_subgenre = {}
        genres = db.session.query(Genre).all()
        for i in genres:
            subgenres =
db.session.query(Subgenre).join(GenreSubgenre).filter_by(genre_id =
i.id).all()
            dict_genre_subgenre[i] = list(map(lambda x: x ,subgenres))
        cart_count = len(get_games())
        #return page with user session
        return make_response(render_template("addgame.html",user=user,
dict_genre_subgenre = dict_genre_subgenre, cart_count=cart_count,
user_icon=user_icon), 202)

    @role_handler(RESOURCE_ROLES)

```

```

@token_required
def post(self):
    content = request.form.to_dict()
    to_flash = []
    try:
        #check name
        title_to = content["name"]
        if len(title_to)<2 or len(title_to)>40:
            to_flash.append("Name must be less than 40
characters")
        #check description
        description_to = content["description"]
        if len(description_to)<2 or len(description_to)>1000:
            to_flash.append("Description must be less than 300
characters")
        #check price
        price_to = content["price"]
        if not price_to.isdecimal():
            to_flash.append("Price must be natural number")
        image = request.files['img'] # get file

    except:
        flash("Something went wrong",category='danger')
        return redirect(url_for('addgame'))
    #retrun is errors
    if to_flash:
        for num, mess in enumerate(to_flash):
            flash(mess,category='danger')
            if num == 4:
                break
        return redirect(url_for('addgame'))

    else:
        #add game
        g = Game(
            title=title_to,
            description=description_to,
            price=price_to
        )
        db.session.add(g)
        db.session.commit()
        flash("Game has been successfully
created",category='success')
        game = db.session.query(Game).filter_by(title =
title_to).first()
        #upload new game img
        upload_img(image,game.uuid)
        #set genres and subgenres
        for i in content:
            try:
                if i[0:5] == "Genre":

```

```

        gs =
db.session.query(GenreSubgenre).filter_by(genre_id = content[i],
subgenre_id = None).first()
        a = GameGenreSubgenre(
            genre_subgenre_id=gs.id,
            game_id=game.id
        )

        elif i[0:8] == "Subgenre":
            gs =
db.session.query(GenreSubgenre).filter_by(subgenre_id =
content[i]).first()
            a = GameGenreSubgenre(
                genre_subgenre_id=gs.id,
                game_id=game.id
            )
            db.session.add(a)
            db.session.commit()
        except:
            continue
    db.session.close()
    return redirect(url_for('main'))

@role_handler(RESOURCE_ROLES)
@token_required
def delete(self):
    #delete game resource
    game_uuid = request.form.get('del_game_uuid')
    game_obj = db.session.query(Game).filter_by(uuid =
game_uuid).first()
    if not game_uuid or not game_obj:
        #return main page
        flash("Not such a game",category='danger')
        return redirect(url_for('main'))
    #set as delete
    try:
        game_obj.is_delete = True
        db.session.commit()
        db.session.close()
    except:
        flash('Something went wrong!', category='warning')
    else:
        #successfully deleted
        flash("Game has been successfully
deleted",category='success')
        return redirect(url_for('main'))

```

Б.8 Текст файла buygame.py

```

from flask import make_response, request, flash, redirect, url_for
from flask_restful import Resource
from src.token import role_handler

RESOURCE_ROLES = [0, 1]

class BuyGame(Resource):
    @role_handler(RESOURCE_ROLES)
    def post(self):
        try:
            #set cookie with list of game uuid
            game_uuid = request.form.to_dict().get('game')
            gamelist = request.cookies.get('gamelist')
            if not gamelist:
                gamelist=game_uuid
            else:
                m = gamelist.split('&')
                f = True
                for i in m:
                    if i == game_uuid:
                        f=False
                        break
                if f:
                    gamelist+="{}{}".format('&', game_uuid)

            response = make_response(redirect(url_for('carts')))
            response.set_cookie('gamelist', gamelist)
            return response

        except:
            flash('Something went wrong!', category='warning')
            return redirect(url_for('main'))

```

Б.9 Текст файла carts.py

```

from flask import make_response, render_template, request, flash,
redirect, url_for
from flask_restful import Resource
from src.token import get_user_by_token, get_games, role_handler
from src.aws_func import get_aws_image

RESOURCE_ROLES = [0, 1]

class Carts(Resource):
    @role_handler(RESOURCE_ROLES)
    def get(self):

```

```

try:
    games = get_games()
    game_img = {}
    #carts img
    for i in games:
        img_tag = get_aws_image("gamestorebucket", i.uuid)
        game_img[i]=img_tag

    total = 0
    for i in games:
        total += i.price
    cart_count = len(get_games())
except:
    flash('Something went wrong!', category='warning')
    return redirect(url_for('main'))
#user img
try:
    user = get_user_by_token()
    try:
        user_icon = get_aws_image("gamestoreuserbucket",
user.uuid)
    except:
        user_icon=False
except:
    return make_response(render_template("carts.html", games =
game_img, total_price = total, cart_count=cart_count), 200)
    #return page with session
    return make_response(render_template("carts.html",user=user,
games = game_img, total_price = total, cart_count=cart_count,
user_icon=user_icon), 202)

def post(self):
    try:
        #if delete
        game_uuid = request.form.to_dict().get('delete')
        gamelist = request.cookies.get('gamelist')
        new_game_list = ""
        #delete from cookie list
        if gamelist:
            m = gamelist.split('&')
            for i in m:
                if i!=game_uuid:
                    if len(new_game_list) ==0:
                        new_game_list=i
                    else:
                        new_game_list+="{}{}".format('&', i)
        #set new cookie list
        response = make_response(redirect(url_for('carts')))
        response.set_cookie('gamelist', new_game_list)
        return response

```

```

except:
    flash('Something went wrong!', category='warning')
    return redirect(url_for('carts'))

```

Б.10 Текст файла editgame.py

```

from flask import make_response, request, flash, redirect, url_for,
render_template
from flask_restful import Resource
from src.token import token_required, role_handler, get_user_by_token,
get_games
from src.database.models import GameGenreSubgenre, Genre,
GenreSubgenre, Subgenre, Game
from src import db
from src.aws_func import upload_img, get_aws_image

class EditGame(Resource):
    RESOURCE_ROLES = [2,3]

    @role_handler(RESOURCE_ROLES)
    @token_required
    def get(self, uuid):
        try:
            user = get_user_by_token()
            try:
                user_icon = get_aws_image("gamestoreuserbucket",
user.uuid)
            except:
                user_icon=False
        except:
            # return page with no user session
            return make_response(render_template("main.html",), 200)
        game = db.session.query(Game).filter_by(uuid=uuid).first()
        dict_genre_subgenre = {}
        genres = db.session.query(Genre).all()
        for i in genres:
            subgenres =
db.session.query(Subgenre).join(GenreSubgenre).filter_by(genre_id =
i.id).all()
            dict_genre_subgenre[i] = list(map(lambda x: x ,subgenres))
        cart_count = len(get_games())
        #return page with sesison
        return
make_response(render_template("editgame.html",user=user, game=game,
dict_genre_subgenre = dict_genre_subgenre, cart_count=cart_count,
user_icon=user_icon), 200)

    @role_handler(RESOURCE_ROLES)
    @token_required

```

```

def post(self, uuid):
    content = request.form.to_dict()
    to_flash = []
    try:
        #check name
        title_to = content["name"]
        if len(title_to)<2 or len(title_to)>40:
            to_flash.append("Name must be less than 40
characters")
        #check description
        description_to = content["description"]
        if len(description_to)<2 or len(description_to)>1000:
            to_flash.append("Description must be less than 300
characters")
        #check price
        price_to = content["price"]
        if not price_to.isdecimal():
            to_flash.append("Price must be natural number")
        #check is visible
        is_visible = True
        if "is_visible" in content.keys():
            is_visible = False

    except:
        flash("Something went wrong",category='danger')
        return redirect(url_for('addgame'))
    #return if errors
    if to_flash:
        for num, mess in enumerate(to_flash):
            flash(mess,category='danger')
            if num == 4:
                break
        return redirect(url_for('addgame'))

    else:
        #update game
        db.session.query(Game).filter_by(uuid = uuid).update(
            dict(
                title=title_to,
                description=description_to,
                price=price_to,
                is_visible=is_visible
            )
        )
        db.session.commit()
        game = db.session.query(Game).filter_by(title =
title_to).first()
        try:
            image = request.files['img'] # get file
        except:
            pass

```

```

else:
    upload_img(image,game.uuid)

    g = db.session.query(GameGenreSubgenre).filter_by(game_id
= game.id).all()
    for i in g:
        db.session.delete(i)
    db.session.commit()
    #set genre and subgenres
    for i in content:
        try:
            if i[0:5] == "Genre":
                gs =
db.session.query(GenreSubgenre).filter_by(genre_id = content[i],
subgenre_id = None).first()
                a = GameGenreSubgenre(
                    genre_subgenre_id=gs.id,
                    game_id=game.id
                )

                elif i[0:8] == "Subgenre":
                    gs =
db.session.query(GenreSubgenre).filter_by(subgenre_id =
content[i]).first()
                    a = GameGenreSubgenre(
                        genre_subgenre_id=gs.id,
                        game_id=game.id
                    )
                    db.session.add(a)
                    db.session.commit()
            except:
                continue
        db.session.close()
        flash("Game has been successfully
updated",category='success')
        return redirect(url_for('main'))

```

Б.11 Текст файла game.py

```

from flask import make_response, render_template, request, flash,
redirect, url_for
from flask_restful import Resource
from src.token import get_user_by_token, token_required, get_games
from src.database.models import User, GenreSubgenre,
GameGenreSubgenre, Genre, Subgenre, Comment, Role
from src.database.models import Game as GameModel
from src import db
from src.aws_func import get_aws_image

```

```

class Game(Resource):
    def get(self, uuid):
        try:
            game_obj = db.session.query(GameModel).filter_by(uuid =
uuid).first()
            if not uuid or not game_obj:
                #return main page
                flash("Not such a game",category='danger')
                return redirect(url_for('main'))
            if not game_obj.is_visible or game_obj.is_delete:
                flash("Not such a game",category='danger')
                return redirect(url_for('main'))
            #create genre subgenre dict
            gen_sub =
db.session.query(Genre).join(GenreSubgenre).join(GameGenreSubgenre).fi
lter_by(game_id=game_obj.id).all()
            dict_genre_subgenre = {}
            for i in gen_sub:
                subgenres =
db.session.query(Subgenre).join(GenreSubgenre).filter_by(genre_id =
i.id).join(GameGenreSubgenre).filter_by(game_id=game_obj.id).all()
                dict_genre_subgenre[i] = list(map(lambda x: x
,subgenres))
            #compose game comments
            user_comment = []
            comments =
db.session.query(Comment).filter_by(game_id_from = game_obj.id).all()
            for i in comments:
                date = str(i.created_date)[0:-3]
                user =
db.session.query(User).filter_by(id=i.user_id_from).first()
                user_comment_list = []
                user_comment_list.append(user)
                user_comment_list.append(i)
                user_comment_list.append(date)
                user_comment.append(user_comment_list)
            cart_count = len(get_games())
            img_tag = get_aws_image("gamestorebucket", game_obj.uuid)

        except:
            flash('Something went wrong!', category='warning')
            return redirect(url_for('main'))
        #get user and user image
        try:
            user = get_user_by_token()
            role =
Role.query.join(User).filter_by(uuid=user.uuid).first()
            try:
                user_icon = get_aws_image("gamestoreuserbucket",
user.uuid)
            except:

```

```

        user_icon=False
    except:
        return make_response(render_template("game.html",
dict_genre_subgenre = dict_genre_subgenre, game = game_obj,
user_comment = user_comment, cart_count=cart_count, image=img_tag),
200)
    #return page with user session
    return
make_response(render_template("game.html",user=user,user_code=role.cod
e, game = game_obj, dict_genre_subgenre=dict_genre_subgenre,
user_comment = user_comment, cart_count=cart_count, image=img_tag,
user_icon=user_icon), 202)

@token_required
def post(self, uuid):
    user = get_user_by_token()
    game_obj = db.session.query(GameModel).filter_by(uuid =
uuid).first()
    if not uuid or not game_obj:
        #return main page
        flash("Not such a game",category='danger')
        return redirect(url_for('main'))
    #add comment
    try:
        content = request.form.to_dict()
        reply_to_message_uuid =
content.get('reply_to_message_uuid')
        reply_to_message_id=None
        if reply_to_message_uuid:
            rep_com =
db.session.query(Comment).filter_by(uuid=reply_to_message_uuid).first(
)
            reply_to_message_id = rep_com.id
        #add comment
        com = Comment(
            text=content.get('comment'),
            reply_to_message_id=reply_to_message_id,
            user_id_from=user.id,
            game_id_from=game_obj.id
        )
        db.session.add(com)
        db.session.commit()
        db.session.close()
    except:
        flash('Something went wrong!', category='warning')
        return redirect(url_for('game', uuid=uuid))

@token_required
def delete(self, uuid):
    try:
        user = get_user_by_token()

```

```

        content = request.form.to_dict()
        comment =
db.session.query(Comment).filter_by(uuid=content.get('delete_comment')
).first()
        role =
Role.query.join(User).filter_by(uuid=user.uuid).first()
        if role.code == 2 or role.code == 3 or user.id ==
comment.user_id_from:
            db.session.delete(comment)
            db.session.commit()
            db.session.close()
    except:
        flash('Something went wrong!', category='warning')
    return redirect(url_for('main'))

```

Б.12 Текст файла login.py

```

import datetime
import jwt
from flask_restful import Resource
from flask import flash, make_response, redirect, request, url_for
from flask_restful import Resource
from src import app
from src.database.models import User

class Login(Resource):
    def get(self):
        flash('Method is not allowed', category='danger')
        return redirect(url_for('main'))

    def post(self):
        #take data from response form
        auth = request.form.to_dict()
        #get user with from key "username"
        user =
User.query.filter_by(username=auth.get('username')).first()
        #if user or user hash are not match
        if not user or not user.check_password(auth.get('password')):
            flash('Username and password are not match! Please try
again',
                    category='danger')
            #returning redirect login
            return redirect(url_for('main'))
        #make csrf code by jwt algorithm
        hour = 168 if auth.get('remember') else 1
        token = jwt.encode(
            {
                "user_id": user.uuid,

```

```

        "exp": datetime.datetime.now() +
datetime.timedelta(hours=hour)
    }, app.config['SECRET_KEY'], algorithm="HS256"
)

flash('You have been authorized', category='success')
#set cookie token and return main page
response = make_response(redirect(url_for('main')))
response.set_cookie('token', token)
return response

class Logout(Resource):
    def get(self):
        flash('Method is not allowed', category='danger')
        return redirect(url_for('main'))
    def post(self):
        #set token expires as 0 to logout
        response = make_response(redirect(url_for('main')))
        response.set_cookie('token', expires=0)
        response.set_cookie('gamelist', "")
        return response

```

Б.13 Текст файла main.py

```

from flask import make_response, render_template, request, flash
from flask_restful import Resource
from src.token import get_user_by_token, get_games
from src.database.models import User, Role, Game, GenreSubgenre,
GameGenreSubgenre, Genre, Subgenre
from src import db
from src.aws_func import get_aws_image

class Main(Resource):
    def get(self):
        try:
            games = set()
            content = request.args.to_dict()
            is_genre_game = True
            for i in content:
                #take all genres
                if content[i] == "Genre":
                    curr_games =
db.session.query(Game).join(GameGenreSubgenre).join(GenreSubgenre).join(
Genre).filter_by(title=i[6:]).all()
                    if isinstance(curr_games, list):
                        for j in curr_games:
                            games.add(j)
                    if len(curr_games)==0:
                        is_genre_game = False

```

```

        #take all subgenres
        elif content[i] == "Subgenre":
            curr_games =
db.session.query(Game).join(GameGenreSubgenre).join(GenreSubgenre).join(
    Subgenre).filter_by(title=i[9:]).first()
            if isinstance(curr_games, list):
                for j in curr_games:
                    games.add(j)
#name search
if not games and is_genre_game:
    games = db.session.query(Game).all()
    search = request.args.get('search')

    if search:
        games =
db.session.query(Game).filter(Game.title.contains(search)).all()
        #create genre title
        game_genre = {}
        for i in games:
            gen_sub =
db.session.query(Genre).join(GenreSubgenre).join(GameGenreSubgenre).fi
lter_by(game_id=i.id).all()
            dop_data = []
            genres = ""
            if gen_sub:
                if len(gen_sub) > 1:
                    genres = "{} / {}".format(gen_sub[0].title,
gen_sub[1].title)
                else:
                    genres = gen_sub[0].title
            dop_data.append(genres)
            #get image from bucket
            img_tag = get_aws_image("gamestorebucket", i.uuid)
            dop_data.append(img_tag)

            game_genre[i] = dop_data
#get all subgenres from current genre in dict
dict_genre_subgenre = {}
genres = db.session.query(Genre).all()
for i in genres:
    subgenres =
db.session.query(Subgenre).join(GenreSubgenre).filter_by(genre_id =
i.id).all()
    dict_genre_subgenre[i] = list(map(lambda x: x
,subgenres))
    cart_count = len(get_games())

except:
    # return page with no games
    flash('Something went wrong!', category='warning')
    return make_response(render_template("main.html",games =

```

```

    {}, dict_genre_subgenre = {})), 200)
    try:
        #get user and user img
        user = get_user_by_token()
        try:
            user_icon = get_aws_image("gamestoreuserbucket",
user.uuid)
        except:
            user_icon=False

    except:
        # return page with no user session
        return make_response(render_template("main.html", games =
game_genre, dict_genre_subgenre = dict_genre_subgenre), 200)
        # return page in user session
        role =
Role.query.join(User).filter_by(username=user.username).first()

        return make_response(render_template("main.html", user=user,
user_code=role.code, games = game_genre, dict_genre_subgenre =
dict_genre_subgenre, cart_count=cart_count, user_icon=user_icon), 202)

```

Б.14 Текст файла order.py

```

from flask import make_response, render_template, request, flash,
redirect, url_for
from flask_restful import Resource
from src.token import get_user_by_token, get_games, role_handler
import re
from src.aws_func import get_aws_image

RESOURCE_ROLES = [0, 1]

class Order(Resource):
    @role_handler(RESOURCE_ROLES)
    def get(self):
        try:
            games = get_games()
            cart_count = len(games)
        except:
            flash('Something went wrong!', category='warning')
            return redirect(url_for('main'))

        try:
            #get user and user img
            user = get_user_by_token()
            try:
                user_icon = get_aws_image("gamestoreuserbucket",
user.uuid)

```

```

        except:
            user_icon=False
    except:
        return make_response(render_template("order.html", games =
games, cart_count=cart_count), 200)

    return make_response(render_template("order.html",user=user,
games = games, cart_count=cart_count, user_icon=user_icon), 202)

def post(self):
    to_flash = []
    try:
        #check name
        name = request.form.get('name')
        if len(name)>30 or len(name)==0:
            to_flash.append("Name must be less than 30
characters")
        #check surname
        last_name = request.form.get('surname')
        if len(last_name)>30 or len(last_name)==0:
            to_flash.append("Last namebe less than 30
characters")
        #check email
        new_email = request.form.get('email_address')
        regex = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-
z]{2,}\b'
        if not(re.fullmatch(regex, new_email)):
            to_flash.append("Invalid Email")
        #check phone
        phone = request.form.get('phone')
        if not phone:
            to_flash.append("Invalid Phone")
        #check payment type
        payment = request.form.get('payment')
        if payment != "card" and payment != "cash":
            to_flash.append("Invalid payment type")
        #check comment
        comment = request.form.get('comment')
        if comment:
            if len(comment)>300:
                to_flash.append("Comment be less than 300
characters")

    except:
        flash("Something went wrong",category='danger')
        return redirect(url_for('order'))
    #return page with errors
    if to_flash:
        for num, mess in enumerate(to_flash):
            flash(mess,category='danger')
            if num == 4:

```

```

        break
    return redirect(url_for('order'))
else:
    #success order
    flash("The order was completed
successfully",category='success')
    response = make_response(redirect(url_for('main')))
    response.set_cookie('gamelist', "")
    return response

```

Б.15 Текст файла register.py

```

from flask import make_response, render_template, request, flash,
redirect, url_for
from flask_restful import Resource
from src import db
from src.database.models import User
import re
from src.token import get_user_by_token

class Register(Resource):

    def get(self):
        try:
            get_user_by_token()
        except:
            return make_response(render_template("register.html",
nav=True), 200)
        flash('You already authorized', category='warning')
        return redirect(url_for('main'))

    def post(self):
        to_flash = []
        try:
            #check username
            new_username = request.form.get('username')
            u = db.session.query(User).filter_by(username =
new_username).first()
            if u:
                to_flash.append("User with username as \"{}\" already
exist".format(new_username))
                if len(new_username)>30:
                    to_flash.append("Username must be less than 30
characters")
            new_name = request.form.get('name')
            if len(new_name)>30:
                to_flash.append("Name must be less than 30
characters")

```

```

        new_surname = request.form.get('surname')
        if len(new_surname)>30:
            to_flash.append("Surname must be less than 30
characters")
        #check email
        new_email = request.form.get('email_address')
        regex = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-
z]{2,}\b'
        if not(re.fullmatch(regex, new_email)):
            to_flash.append("Invalid Email")
        e = db.session.query(User).filter_by(email_address =
new_email).first()
        if e:
            to_flash.append("User with email as \"{ }\" already
exist".format(new_email))
        #check password
        new_password = request.form.get('password')
        if len(new_password) < 5 or len(new_password) > 50:
            to_flash.append("Password must be more than 5 and less
than 50 characters")
        #check repeat password
        new_password_repeat = request.form.get('password_repeat')
        if new_password !=new_password_repeat:
            to_flash.append("Password mismatch")
    except:
        flash("Something went wrong",category='danger')
        return redirect(url_for('register'))
    #return page with errors
    if to_flash:
        for num, mess in enumerate(to_flash):
            flash(mess,category='danger')
            if num == 4:
                break
        return redirect(url_for('register'))
    else:
        #create user
        try:
            u = User(
                username=new_username,
                name=new_name,
                surname=new_surname,
                email_address=new_email,
                password=new_password,
                code = 1
            )
            db.session.add(u)
            db.session.commit()
            db.session.close()
        except:
            flash("Something went wrong",category='danger')
            return redirect(url_for('register'))

```

```

        else:
            flash("User has been successfully
created",category='success')
            return redirect(url_for('main'))

```

Б.16 Текст файла user.py

```

from flask import make_response, render_template, request, flash,
redirect, url_for
from flask_restful import Resource
from src.token import get_user_by_token, token_required, get_games
from src.database.models import Role
from src.database.models import User as UserModel
from src.aws_func import upload_user_img, get_aws_image
from src import db
import re
from flask_bcrypt import generate_password_hash

class User(Resource):
    @token_required
    def get(self, uuid):
        try:
            user_obj = db.session.query(UserModel).filter_by(uuid =
uuid).first()
            if not uuid or not user_obj:
                #return main page
                flash("Not such user",category='danger')
                return redirect(url_for('main'))
            #get auth user img
            curr_user = get_user_by_token()
            try:
                user_icon = get_aws_image("gamestoreuserbucket",
curr_user.uuid)
            except:
                user_icon=False
            #current user role
            role =
Role.query.join(UserModel).filter_by(uuid=curr_user.uuid).first()
            roles = db.session.query(Role).all()
            user_view_role =
db.session.query(Role).join(UserModel).filter_by(uuid=user_obj.uuid).f
irst()
            cart_count = len(get_games())
            #get current user img
            try:
                img_tag = get_aws_image("gamestoreuserbucket",
user_obj.uuid)
            except:

```

```

        img_tag = False
    except:
        flash('Something went wrong!', category='warning')
        return redirect(url_for('main'))
    #return user page with session
    return
make_response(render_template("user.html",user=curr_user, user_view =
user_obj, user_role=role,user_view_role = user_view_role, roles =
roles, cart_count=cart_count, image=img_tag, user_icon=user_icon),
200)

@token_required
def post(self, uuid):
    content = request.form.to_dict()
    to_flash = []
    try:
        #check name
        new_name = content['name']
        if len(new_name)>30:
            to_flash.append("Name must be less than 30
characters")
        #check surname
        new_surname = content['surname']
        if len(new_surname)>30:
            to_flash.append("Surname must be less than 30
characters")
        #check email
        new_email = content['email_address']
        regex = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-
z]{2,}\b'
        if not(re.fullmatch(regex, new_email)):
            to_flash.append("Invalid Email")
        e = db.session.query(UserModel).filter_by(email_address =
new_email).first()
        user_obj = db.session.query(UserModel).filter_by(uuid =
uuid).first()
        if e and e !=get_user_by_token() and e !=user_obj:
            to_flash.append("User with email as \"{ }\" already
exist".format(new_email))
        #if password then check password
        p=True
        if 'password' in content.keys() and 'password_repeat' in
content.keys():
            if len(content['password'])>0 and
len(content['password_repeat'])>0:
                new_password = request.form.get('password')
                if len(new_password) < 5 or len(new_password) >
50:
                    to_flash.append("Password must be more than 5
and less than 50 characters")
                new_password_repeat =

```

```

request.form.get('password_repeat')
    if new_password != new_password_repeat:
        to_flash.append("Password mismatch")
    else:
        p=False
    else:
        p=False
        new_code_role = content['role']
        role_obj =
db.session.query(Role).filter_by(code=new_code_role).first()
except:
    flash("Something went wrong",category='warning')
    return redirect(url_for('user', uuid=uuid))
#return page with errors
if to_flash:
    for num, mess in enumerate(to_flash):
        flash(mess,category='danger')
        if num == 4:
            break
    return redirect(url_for('user', uuid=uuid))
else:
    #update user info
    db.session.query(UserModel).filter_by(uuid = uuid).update(
        dict(
            name=new_name,
            surname=new_surname,
            email_address=new_email,
            role_id=role_obj.id
        )
    )
    db.session.commit()
    #update password
    if p:
        db.session.query(UserModel).filter_by(uuid =
uuid).update(
            dict(

password=generate_password_hash(new_password).decode('utf8')
            ))
        db.session.commit()

    user =
db.session.query(UserModel).filter_by(uuid=uuid).first()
    #update user img
    try:
        image = request.files['img'] # get file
    except:
        pass
    else:
        upload_user_img(image,user.uuid)
db.session.close()

```

```
        flash("User has been successfully  
updated",category='success')  
        #successfully updated  
        return redirect(url_for('user', uuid=uuid))
```

ДОДАТОК В
Слайди презентації

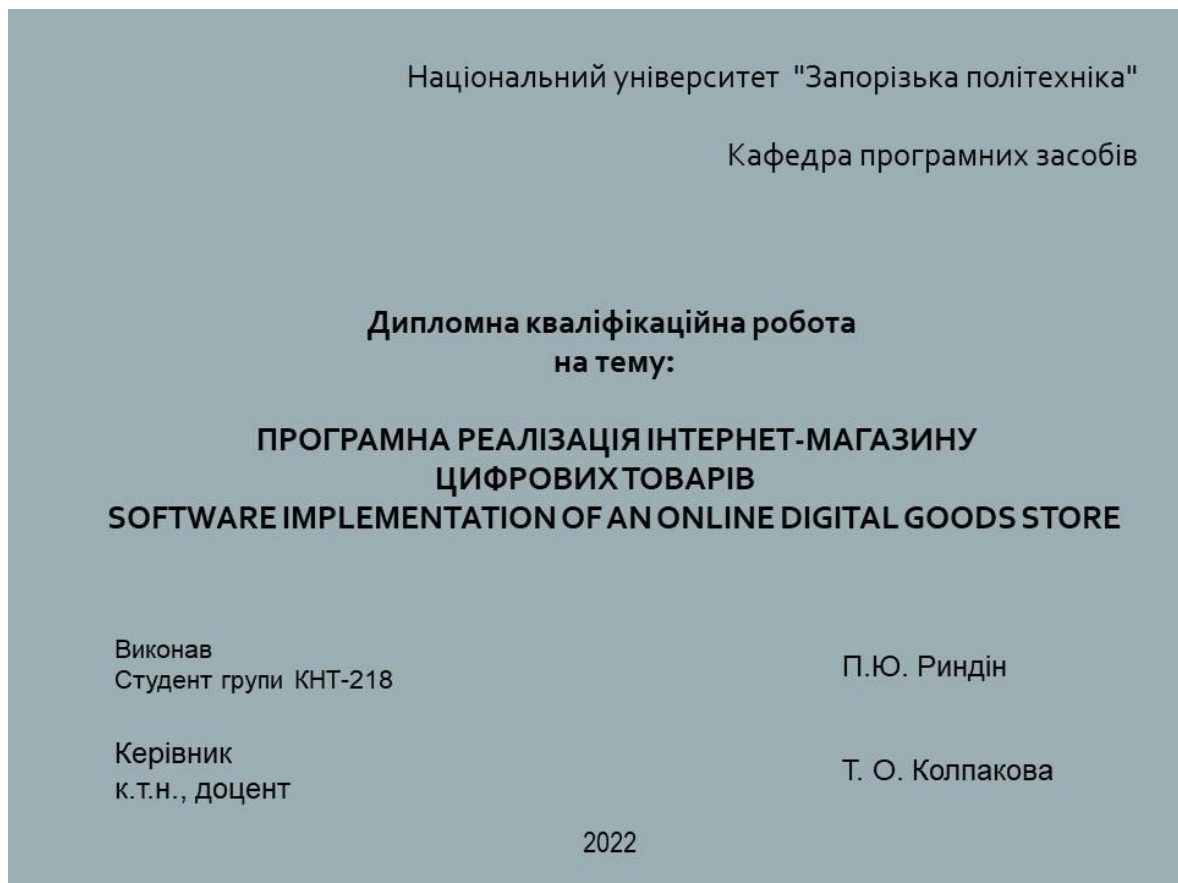


Рисунок В.1 – Слайд № 1

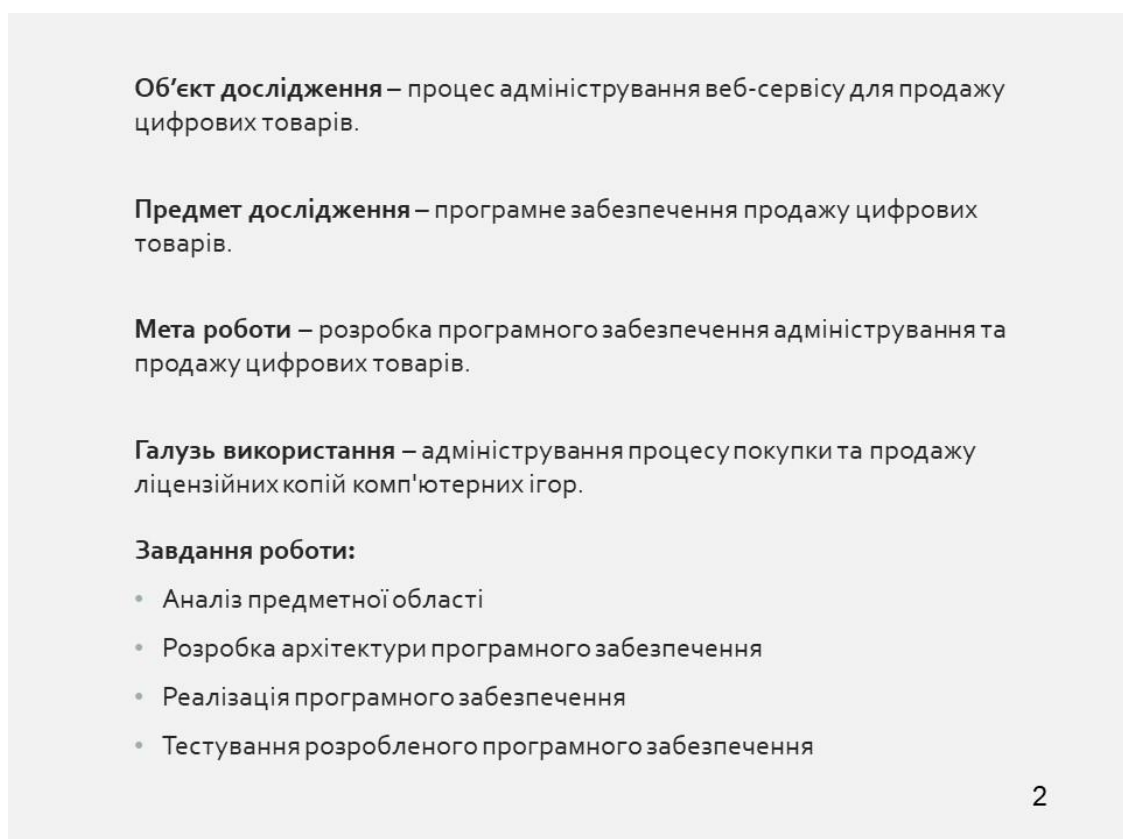
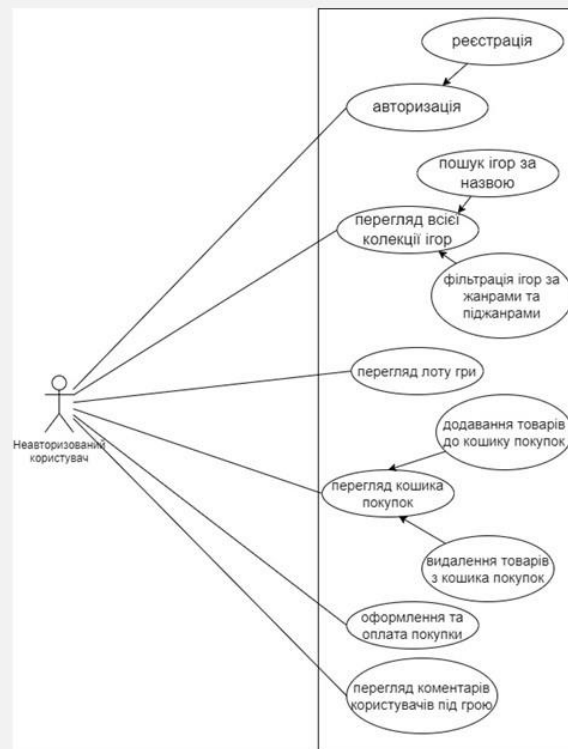


Рисунок В.2 – Слайд № 2

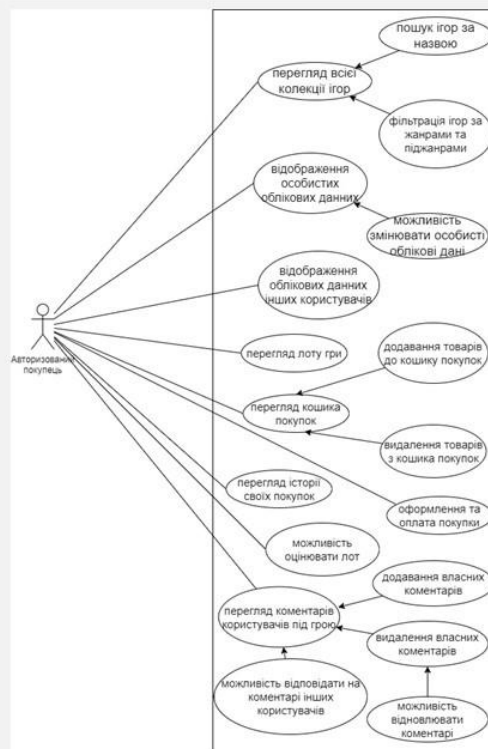
ДІАГРАМА ПРЕЦЕДЕНТІВ НЕАВТОРИЗОВАНОГО КОРИСТУВАЧА



3

Рисунок В.3 – Слайд № 3

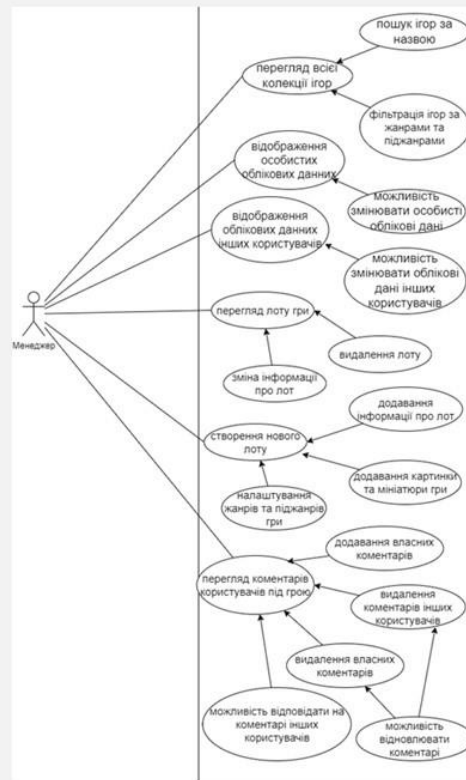
ДІАГРАМА ПРЕЦЕДЕНТІВ АВТОРИЗОВАНОГО КОРИСТУВАЧА



4

Рисунок В.4 – Слайд № 4

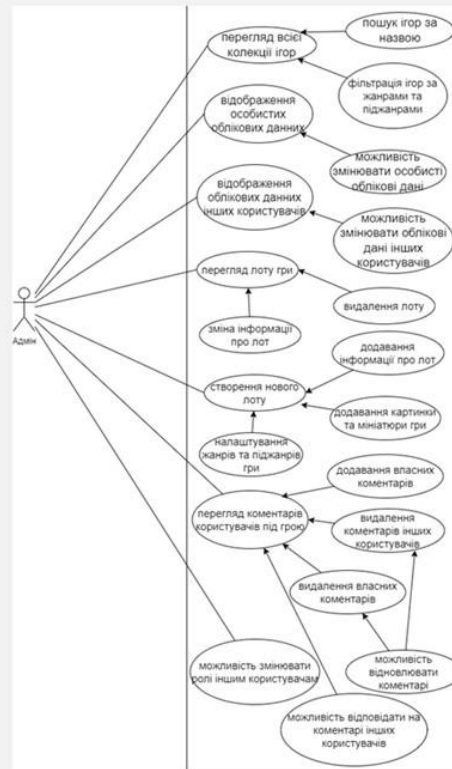
ДІАГРАМА ПРЕЦЕДЕНТІВ МЕНЕДЖЕРА



5

Рисунок В.5 – Слайд № 5

ДІАГРАМА ПРЕЦЕДЕНТІВ АДМІНІСТРАТОРА



6

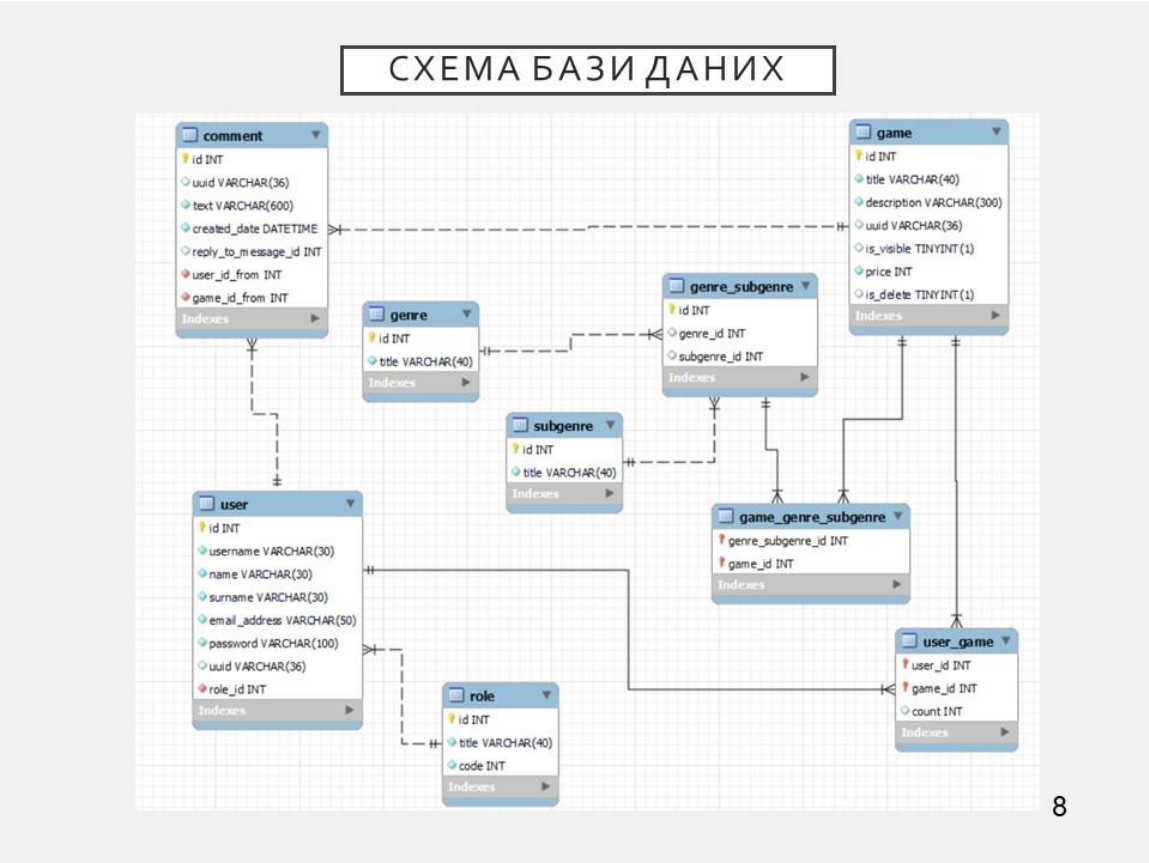
Рисунок В.5 – Слайд № 6

Порівняння Python web фреймворків		
Характеристика	Flask	Django
1	2	3
Призначення	Веб-фреймворк Python, створений для швидкої розробки	Веб-фреймворк Python, створений для важких і комплексних проектів
Тип використання веб-серверів	WSGI (Web Server Gateway Interface) фреймворк	фуллстек веб-фреймворк
Підтримка API	надає підтримку API (Application Programming Interface)	Django не підтримує API
Підтримка Visual Debug	Підтримка Visual Debug	Немає підтримки Visual Debug
Підтримка використання декількох баз даних	дозволяє використовувати кілька типів баз даних	не дозволяє працювати з декількома типами баз даних

1	2	3
Підтримка динамічних сторінок HTML	не працює з динамічними сторінками HTML (HyperText Markup Language)	працює з динамічними сторінками HTML
Підтримка різноманітних парадигм до написання коду	Flask пропонує різноманітний стиль роботи	Django пропонує монолітний стиль роботи
Додаткові модулі	Підтримує сторонні розширення, які можуть бути реалізовані у роботі	Django має власну бібліотеку модулів
Структура веб-фреймворка	Структура макета проекту для веб-фреймворка Flask є довільною	Структура макета проекту для Django чітка та незмінна
Підхід до URL-адрес системи	Диспетчер URL-адрес веб-фреймворку - це запит REST (Representational State Transfer)	Диспетчер URL-адрес цього фреймворку Django заснований на контролері-регулярних виразів
Ступінь новизни веб-фреймворку	Flask є набагато молодшою платформою в порівнянні з Django	Django — це дуже зрілий фреймворк

7

Рисунок В.7 – Слайд № 7



8

Рисунок В.8 – Слайд № 8

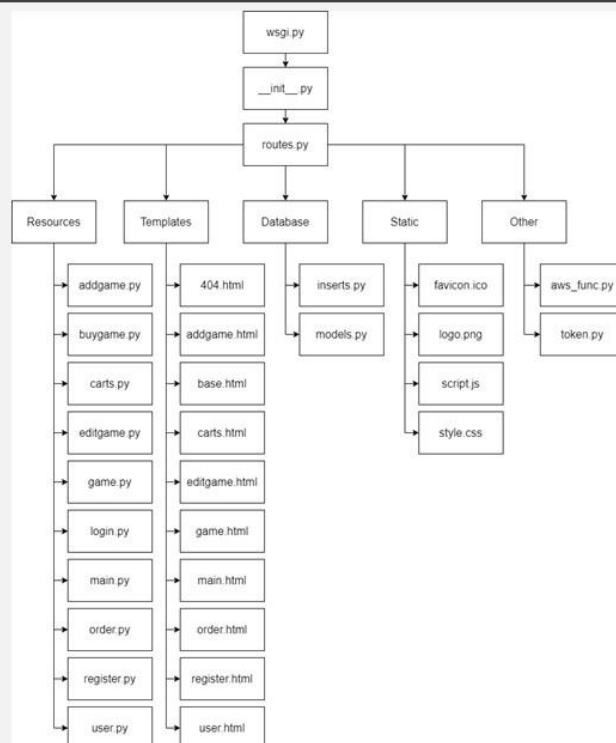
СХЕМА ФУНКЦІОНУВАННЯ СЕРВІСУ



9

Рисунок В.9 – Слайд № 9

СТРУКТУРНА СХЕМА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



10

Рисунок В.10 – Слайд № 10

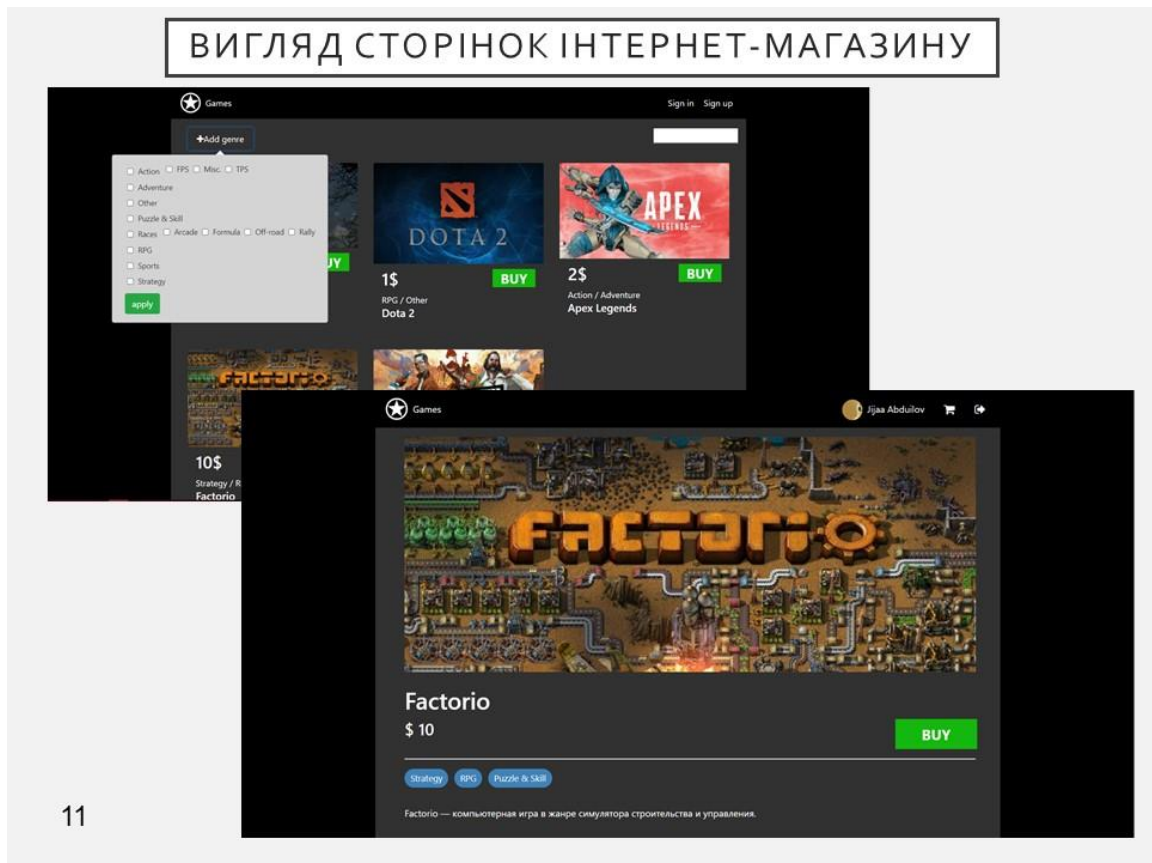


Рисунок В.11 – Слайд № 11

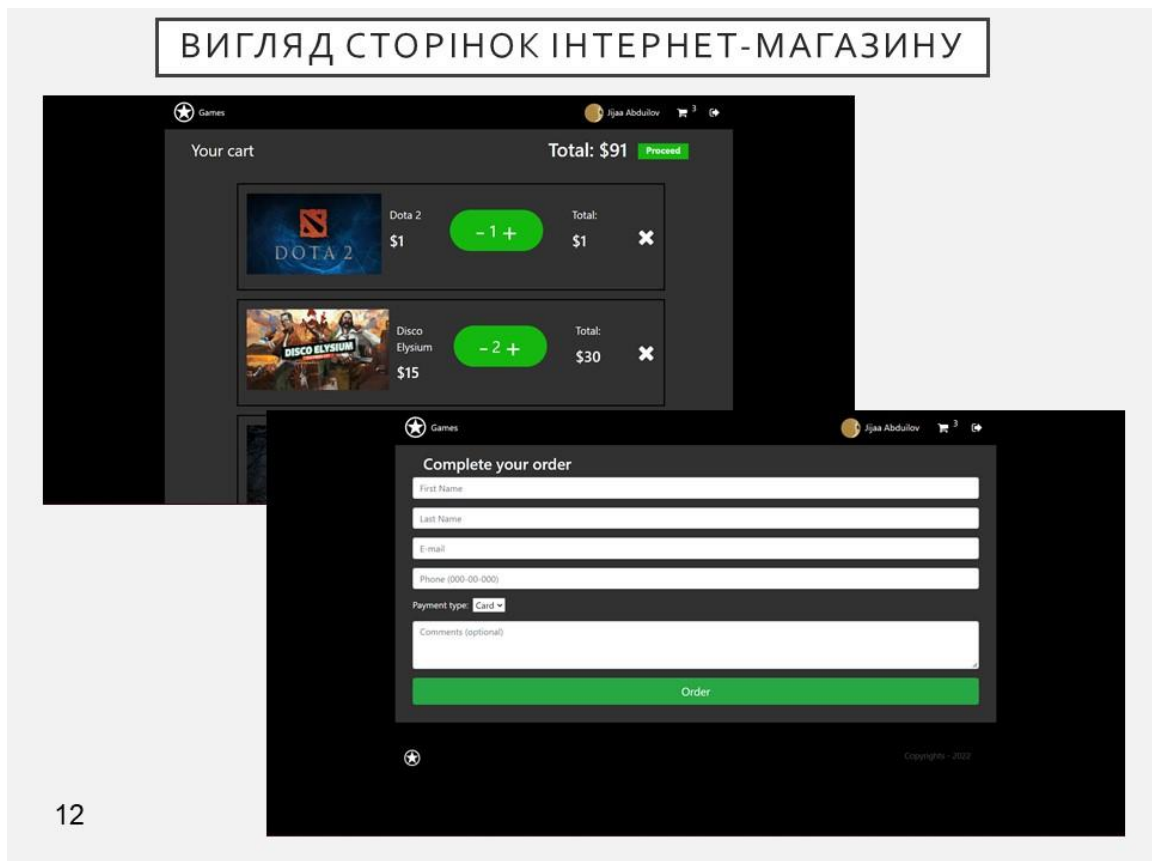


Рисунок В.12 – Слайд № 12

РЕЗУЛЬТАТИ АВТОТЕСТУВАННЯ

```

tests/test_addgame.py ... [ 10%]
tests/test_buygame.py .. [ 16%]
tests/test_carts.py ..... [ 30%]
tests/test_editgame.py .... [ 40%]
tests/test_game.py ..... [ 53%]
tests/test_login.py .... [ 63%]
tests/test_main.py .. [ 70%]
tests/test_order.py ... [ 80%]
tests/test_register.py ... [ 90%]
tests/test_user.py ... [100%]

----- coverage: platform linux, python 3.8.12-final-0 -----
Name                               Stmts   Miss  Cover
-----
src/__init__.py                      39      4    90%
src/aws_func.py                     18      6    67%
src/database/__init__.py              0      0   100%
src/database/inserts.py             133    133     0%
src/database/models.py              102     18    82%
src/resources/addgame.py             87     61    30%
src/resources/buygame.py             26      9    65%
src/resources/carts.py               48     17    65%
src/resources/editgame.py            81     57    30%
src/resources/game.py                82     47    43%
src/resources/login.py               32      4    88%
src/resources/main.py                62     23    63%
src/resources/order.py               60     20    67%
src/resources/register.py            61     19    69%
src/resources/user.py                89     60    33%
src/routes.py                       26      1    96%
src/token.py                         63     17    73%
TOTAL                               1009    496    51%

===== 30 passed in 10.63s =====
The command "pytest --cov=src tests/" exited with 0.

```

13

Рисунок В.13 – Слайд № 13

ВИСНОВКИ

За результатами виконання дипломної кваліфікаційної роботи виконано такі задачі:

проведено аналіз концепції продажу цифрових товарів, розглянуто підходи до реалізації онлайн-магазинів, проведено порівняння з програмними реалізаціями;

визначено функціональні вимоги для реалізації програмного забезпечення, обрано мову програмування Python, Flask фреймворк, СУБД MySQL, створено структуру бази даних проекту;

реалізовано весь запланований функціонал програмного забезпечення, створено структуру вебсервісу та описано кожний модуль системи. Також описані призначення класів та функцій та їх реалізацій;

проведено тестування розробленої системи. Модулем для тестування було обрано бібліотека Pytest. Для автоматизації тестування був обраний сервіс тестування програмного забезпечення Travis CI.

Розроблений вебсервіс призначений для продажу цифрових копій комп'ютерних ігор і являє собою інтернет-магазин з розподіленою системою ролей доступу серед користувачів. Програма включає в себе зручний інтерфейс адміністрування лотів, пошук з налаштуваннями за жанровою специфікою, систему купівлі та доставки, можливість оцінювати, коментувати та ділитися з спільнотою відгуками стосовно товару.

14

Рисунок В.14 – Слайд № 14