НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Веб-застосування «Ігрова бібліотека»*

Виконав: студент IV курсу, групи *ІП-61 Шатровський Андрій Олександрович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., Крамар Ю.М.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

старший викладач кафедри ТК Олена СИРОТА _____
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

ID перевірки:
1004053777

Дата перевірки:
15.06.2020 16:57:17 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
15.06.2020 17:26:56 EEST

ID користувача:
77149

Назва документу: **Shatrovskij_ip61_2**

ID файлу: **1004066714** Кількість сторінок: **64** Кількість слів: **8995** Кількість символів: **65639** Розмір файлу: **1.07 MB**

13.9% Схожість

Найбільша схожість: **10.8%** з джерело бібліотеки. ID файлу: **1003892361**

4.54% Схожість з Інтернет джерелами	162	Page 66
13.7% Текстові збіги по Бібліотеці акаунту	613	Page 69

0.93% Цитат

Цитати	5	Page 70

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	4

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Шатровському Андрію Олександровичу
(прізвище, ім’я, по батькові)

1. Тема проєкту *Веб-застосування «Ігрова бібліотека»*

керівник проєкту *Крамар Юлія Михайлівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «10» *червня 2020 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: загальні положення, опис функціональних та нефункціональних вимог, аналіз відомих програмних продуктів, визначення цілі та задач розробки

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення, безпека паролю та автентифікації

3) Аналіз якості та тестування програмного забезпечення:опис процесів Тестування, підходи до тестування, опис контрольного прикладу

4) Впровадження та супровід програмного забезпечення

5) Керівництво користувача, опис програми

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*21.02.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*03.03.2020*	
3.	*Постановка та формалізація задачі*	*18.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*30.03.2020*	
5.	*Алгоритмізація задачі*	*06.04.2020*	
6.	*Моделювання програмного забезпечення*	*11.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*15.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*21.04.2020*	
9.	*Розробка програмного забезпечення*	*28.04.2020*	
10.	*Налагодження програми*	*12.05.2020*	
11.	*Виконання графічних документів*	*16.05.2020*	
12.	*Оформлення пояснювальної записки*	*17.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*		
15.	*Подання ДП на основний захист*	*10.06.2020*	

Студент _____ Андрій ШАТРОВСЬКИЙ
(підпис)

Керівник _____ Юлія КРАМАР
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4		Завдання на дипломний проєкт	2	
2	A4	КПІ.ІП-6127.045440.01.81	Пояснювальна записка	70	
3	A4	КПІ.ІП-6127.045440.02.91	Технічне завдання	12	
4	A4	КПІ.ІП-6127.045440.03.34	Керівництво користувача	21	
5	A4	КПІ.ІП-6127.045440.04.13	Опис програми	41	
6	A3	КПІ.ІП-6127.045440.05 СС	Схема структурна варіантів використань	1	
7	A3	КПІ.ІП-6127.045440.06 СБД	Схема бази даних	1	
8	A3	КПІ.ІП-6127.045440.07 КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6127.045440.00.90		
Розробн.		Шатровський А.О			Відомість дипломного проєкту	Літ.	Лист 4 / Листів 1
Керівн.		Крамар Ю.М.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-61	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 13 рисунків, 16 таблиць, 3 додатки, 10 джерел.

Мета. Дипломний проєкт присвячений розробці програмного забезпечення для полегшення процесу пошуку ігор за різними критеріями, такими як жанром гри, її видавцем, роком виходу, та іншими, обумовленими певними вподобаннями користувачів, а також для спрощення перегляду необхідної інформації про ігри, та для спрощення обговорення гри.

Основними задачами є створення зручного інтерфейсу для керування іграми, новинами, жанрами, видавцями та ігровими платформами, а також розробка алгоритму пошуку ігор за багатьма критеріями.

У розділі аналізу вимог до програмного забезпечення були визначені основні вимоги, проаналізована предметна область, та розглянуті відомі аналоги.

У розділі моделювання та конструювання програмного забезпечення було розроблено архітектуру серверної таклієнтської частини, та виконано моделювання програмного забезпечення.

У розділі аналіза якості та тестування програмного забезпечення були описані підходи до тестування, а також розглянутий контрольний приклад тесту.

У розділі впровадження та супроводу програмного забезпечення було описано процес розгортання програмного продукту

КЛЮЧОВІ СЛОВА : ВЕБ-ЗАСТОСУВАННЯ, ВІДЕОІГРИ, ІГРИ

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of four sections, contains 13 images, 16 tables, 3 applications, 10 sources.

Goal. The dissertation project is designed to develop software to facilitate the search for games by various criteria, such as game genre, publisher, release year, and others, due to certain user preferences, as well as to simplify viewing the necessary information about games and to simplify game discussion.

The main tasks are to create a user-friendly interface for managing games, news, genres, publishers and game platforms, as well as to develop a game search algorithm by many criteria.

In the section of the analysis of requirements to the software the basic requirements were defined, the subject area is analyzed, and known analogues are considered.

In the section of software modeling and design, the architecture of the server client part was developed, and software modeling was performed.

The section of quality analysis and software testing described approaches to testing, as well as a test example.

The software deployment and maintenance section described the software deployment process

KEY WORDS: WEB APPLICATIONS, VIDEOGAMES, GAMES

Пояснювальна записка
до дипломного проєкту

на тему: Веб-застосування «Ігрова бібліотека»_____
_____
_____

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Геймер (gamer з англ. - «гравець») - людина, що грає у відеоігри, або в настольні ігри.

Токен – це електронний ключ доступу до певної інформації.

JWT – стандарт створення токенів для надання доступу до певного набору інформації з використанням JSON формату.

API – програмний інтерфейс застосування, сукупність правил і засобів, які вможливлюють взаємодію між різними складовими частинами програмного забезпечення.

Багаторівнева архітектура - клієнт-серверна архітектура, в якій розділяються функції представлення, обробки і зберігання даних.

Вебсайт – сукупність, пов’язаних між собою веб-сторінок, зазвичай об’єднаних одним доменним ім’ям.

Фреймворк (іноді фреймворк; англіцизм, неологізм від framework - остов, каркас, структура) - програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту.

Патерни Програмування – це напрацьовані ефективні підходи, техніки та правила розв’язання задач при створенні програмного забезпечення. Вони не прив’ящані до певних мов програмування.

Тест кейс – сукупність умов, кроків, вхідних параметрів і очікуваних результатів для випробування коректної роботи продукту та перевірки виконання зазначених вимог.

ВСТУП

Індустрія відеоігор — економічний сектор, пов'язаний з розробкою, просуванням та продажем відеоігор. У неї входить велика кількість спеціальностей, за якими працюють тисячі людей по всьому світу.

Зараз, як ніколи, ігри лежать в основі розважального бізнесу. Спосіб взаємодії споживачів з іграми постійно змінюється. Це призводить до залучення нових груп налеселння до ігрової індустрії. Зараз у всьому світі є понад 2,5 мільярда геймерів. У поєднанні на ігри вони витратили близько 138 мільярдів доларів США в 2018 році, та близкьо 150 мільярдів в 2019 році.

Аналітичне агентство NewZoo зазначає, що в 2018 вперше за весь час гри на мобільні пристрої принесли більше грошей, ніж для PC. Дохід від ігор на мобільних пристроях - 70,3 мільярда доларів, PC - 32,9 мільярдів доларів (з них близько 4 мільярдів прийшло з браузерних ігор), в той час у ігрових консолей цей показник дорівнює - 34,6.

Лише в Україні кількість геймерів вже становить близко 15 мільйонів осіб. При такій їх кількості стає очевидним необхідність в веб-застосуваннях де користувачі могли б підшукати гру за інтересами, чи слідкувати за ігровими новинами. Також росте кількість людей зацікавлених в настольних іграх.

Тому основною метою розробки є створення Веб-застосування для перегляду ігрових новин, перегляд інформації про ігри, та їх пошук за вподобаннями користувачів.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Для розробки описаного програмного забезпечення було вибрано форму веб-застосування. Веб-застосування - клієнт-серверне застосування, в якому клієнт взаємодіє з веб-сервером за допомогою браузера. Логіка веб-застосування розподіляється поміж сервером і клієнтом, дані зберігаються на сервері, також в мережі може здійснюватися обмін інформацією. Перевагою є те, що клієнти можуть користуватися іншими операційними системами користувача, при цьому не залежать від конкретної. Отже веб-додатки є міжплатформеними службами. Веб-застосування являє собою веб-сайт, де розташовані сторінки з незакінченим вмістом. Відвідувачу веб-сайту необхідно запросити сторінку з веб-сервера, щоб отримати остаточний сформований вміст. Динамічною називається сторінка, яка залежить від запиту користувача. В моєму випадку використання веб-застосування є дуже доречним, так як більшості користувачів в інтернеті зручніше всього шукати та переглядати інформацію через браузер.

Для реалізації цілей користувачів веб-застосування «Ігрова бібліотека» також необхідні самі дані про ігри, та інші складові які з ними пов’язані. Заповнення веб-сайту інформацією буде здійснюватись адміністраторами, які зможуть постійно вносити як інформацію про нові ігри та різні новини, так і заповнювати базу даними про вже вийшовші з фокусу користувачів ігровими компаніями, платформами, жанрами.

Найкраще користувачі зможуть поділитись своїми враженнями та думками про різні сутності в додатку за допомогою коментарів.

1.2 Змістовний опис і аналіз предметної області

Важливою частиною даної роботи є дослідження того, в якій саме інформації будуть зацікавлені потенціальні користувачі. Насамперед це буде

сам опис сюжету чи основних механік гри, але крім цього існують багато важливих областей.

Однією з таких предметних областей є інформація про розробників цих ігор. Розробкою комп'ютерних ігор може займатися як одна людина, так і фірма (колектив розробників). Комерційні гри створюються командами розробників, найнятими однією фірмою. Фірми можуть спеціалізуватися на виробництві ігор для персональних комп'ютерів, ігрових приставок або планшетних комп'ютерів. Також окремою сферою є настільні ігри. Розробка може фінансуватися іншою, більш великою фірмою - видавцем. Фірма-видавець після закінчення розробки займається розповсюдженням гри і бере на себе пов'язані з цим витрати. Протилежним підходом є така розробка, коли фірма самостійно (без участі видавців) поширює копії ігор, наприклад, засобами цифрової дистрибуції.

Приклади відомих компаній наведені нижче.

– Nintendo.
– Ubisoft.
– Electronic Arts.
– Sony Computer Entertainment.
– Square Enix.
– Xbox Game Studios.
– Bandai Namco.
– Activision Blizzard.
– Bethesda Softworks.

Розробка найбільш високобюджетних ігор може коштувати десятки мільйонів доларів США, причому в останні 2 десятиліття ці бюджети безперервно росли, як і чисельність команд розробників і терміни розробки.

Не менш важливою частиною інформації про гру є ігрофа платформа. Найпопулярнішими зараз є Персональний Комп’ютер (ПК), ігрова консоль, VR. Також в моєму веб-застосуванні в якості ігрової платформи може виступати настільна гра.

Ігрова приставка (ігрова консоль) - спеціалізоване електронний пристрій, призначений для відеоігор. Для таких пристроїв, на відміну від персональних комп'ютерів, запуск і відтворення відеоігор є основним завданням. Домашні ігрові приставки використовують телевізор, проектор або комп'ютерний монітор в якості відображення. Гральні консолі мають свій вбудований пристрій відображення(тобто, не приставляються ні до якої системи), тому недоречно давати їм назву «ігрові приставки». Ігрові приставки, спершу відрізнялися від ПК. Тому що, вони використовували телевізор в якості головного пристрою, а також тому, що вони не підтримували пристрої, які були створені для ПК, такі як: клавіатура, або мишка. На сьогоднішній день, майже всі ігрові консолі призначалися для запуску ігор, які поширюються з умовою відсутності підтримки інших консолей. Як виняток, програмне забезпечення деяких консолей може поширюватись під вільними ліцензіями.

Персона́льний комп'ю́тер (скорочено ПК) — це електронно-обчислювальний пристрій, що необхідний для задоволення потреб більшості людей, який включає в себе зберігання та переробку інформації. Спочатку комп'ютер був створений як обчислювальна машина, але його можна застосовувати з іншою метою, як пристрій доступу в мережу, та як платформу для мультимедіа (мультімедіастанція) і комп'ютерних ігор (ігровий ПК).

Віртуальна реальність (ВР, англ. Virtual reality, VR, штучна реальність) - створений технічними засобами світ, , в якому людині надається можливість поринути в «віртуальну реальність» через його відчуття: слух, зір, вібрації і інші. Штучна реальність імітує в формі матеріального й ідеального, яка змушує відчути реальність через комп’ютерний синтез ознак і реакцій штучної (віртуальної) реальності, відбувається в реальному часі. Об'єкти віртуальної реальності зазвичай ведуть себе близько та подібні до поведінки предметів матеріальної реальності. Суб’єкт впливає на ці предмети в гармонії з реальними законами фізики (властивостями води, гравітацією, зіткненням з об’єктами, і тому подібне.). Але, часто в розважальних цілях користувачам

віртуальних світів дозволяється мати більше можливостей, ніж в реальному житті (наприклад: мати супер силу, злітати в космос і тому подібне). Не варто плутати штучну реальність з доповненою. Тому що, їх суттєва відмінність в тому, що віртуальна реальність створює штучний світ, а доповнена вносить корективи в сприйняття реального світу. Для ігор з віртуальною реальністю потрібне спеціальне обладнання – ВР шолом, контролери, та інше.

Однією з найважливіших частин інформації про гру є її ігрові жанри. Для класифікації віртуальних ігор використовується жанр, який відповідає ігровим діям гравця. Жанр гри, від сценарію не залежить, на відміну від кінофільмів, чи літератури. В відеоіграх немає класифікації жанрів, тому їх можуть віднести до різних категорій. Також гру можна змішати з іншими жанрами, тому її неможливо віднести до конкретного жанру.

Основою сучасних розподілів ігор на жанри прилягає вид активності, що частіше за все здійснює гравець в іграх цього жанру. Загалом відеоігри поділяються на ігри руху, планування, а також сюжету та спілкуванню, руху і контролю. В більшості класифікацій визначення жанру відтворюється за кількома осями. До прикладу, за двома осями: сюжет, свобода дії; можливий ще варіант за трьома: абстракція, симуляція, свобода. Найчастіше класифікація, яка була використана, хоч і не прийнятою усіма, жанри які можна зустріти в більшості існуючих, наведена нижче, що виключає багаторівневі поділи або осі:

Екшн - в такому ігровому жанрі потрібно використовувати рефлекси та швидкість реакції для того, щоб подолати більшість ігрових обставин. Він являється одним з базових жанрів, який водночас найпоширеніший. Зазвичай ігри цього жанру, пов'язані із агресивними діями, які відносяться до противників та оточення. Гравець обирає собі персонажа, який повинен: переслідувати ціль, стріляти, битися, та самому не стати цілю для переслідування. Ті екшн-ігри, що мають наявні значні елементи пригодницьких ігор, використовується термін «Action-adventure». Деякі з них виділяються напрямом аркадних ігор, ігровий процес, що вирізняється

простотою та легкістю сприйняття. Такий жанр, який ділиться на велику кількість піджанрів, посеред яких основниий список наведений нижче.

– Шутери — потребують від гравця боротьби з противником шляхом перестрілки. В залежності від перспективи, що поділяються на шутери від першої особи (Counter Strike), чи третьої (Project Winter). Бувають види як тактичні, де ігровий персонаж виступає у складі команди (Warface), аркади (Pac-Man), а також стелс-екшн, ціллю якого є незримі дії для виконання доручень, без прямого знищення ворогів (серія ігор Splinter Cell). Стосовно ігор, які виступають основою ігрового процесу є знищення великих кількостей супротивників, де стрілянина має прерогативу над тактикою і влучністю, застосовується термін Shoot 'em up., застосовується термін Shoot 'em up (R-Type, Touhou, Contra).

– Файтинги — відтворюють ближній бій, який відбувається зазвичай один на один, на спеціально підготовлених аренах. До прикладу: Mortal Kombat, Injustice.

– Beat 'em up — похожі до файтингів, але мають різницю в тому, що персонажі можуть з легкістю переміщуватися ігровим світом, де вони б'ються проти багатьох ворогів одночасно. Приклади: Hack and slash, Shoot ’em up, Double Dragon.

– Платформери — у цьому під жанрі персонаж змушений рухатись, також може рухатися платформами, стрибаючи по них, та таким чином долає перешкоди . До прикладу: Mario, Hollow Knight.

– Лабіринти — під жанр даної гри змушує персонажа рухатись лабіринтом, з ціллю знайти вихід, уникнути пасток. Зазвичай в таких іграх є обмеження на час. Приклади: Pac-Man, Танчики, Boulder.

Стратегія - зміст стратегічних ігор залежить від планування та побудови стратегії для досягнення конкретної цілі. До прикладу, захопити певну територію, здобути прерогативу в польовій операції. Гравець має можливість керувати не однією одиниці техніки, або персонажем, а цілим військом,

фірмою, або хоч світом. Стратегічні комп'ютерні ігри поділяються на різні типи які наведені нижче.

– Покрокові стратегічні ігри — гравець, та його супротивник виконують діяння один за одним, поетапно, що дає їм змогу за один ігровий хід здійснити певну кількість дій. Приклади: Warcraft, серія ігор Цивілізація.

– Стратегічні ігри в реальному часу — гравці виконують свої стратегічні плани одночасно, але нерідко масштаб часу різниться від реального. Наприклад, зведення певних споруд триває кілька секунд, а ось ігрова година, складає декілька хвилин реального часу. Приклади: Age of Empires II HD, Supreme Commander, Company of Heroes.

– Tower Defense — такий жанр, де гравці керують захисними спорудами, для того щоб не підпустити хвилі ворогів.

– MOBA — в цьому жанрі гравець керує одним персонажем, робить все можливе, щоб не дати ворогу знищити свою базу, та знищити ворожу.

– MMORTS — це користувацькі онлайн стратегії в дійсному часі, які орієнтуються на суперництво, об'єднання у фракції, клани, а також на співпрацю з іншими реальними користувачами, для того, щоб досягнути колективної цілі.

– Різновидом стратегічних ігор, які віддзеркалюють суто бойові дії, є wargames. Ігри, де гравець керує велетенськими державами, всесвітами і т.п., при цьому приймає участь у всіх аспектах їхнього життя, включаючи: науку, торгівлю та масштабні війни, такі ігри отримали назву глобальних стратегій. Приближеним до них є ігри, в яких гравець виступає в ролі надзвичайного створіння, яке керує світом завдяки магії, володарів, управляє силами стихій.

Рольова гра - це гра, де гравець співвідноситься з персонажем, виступає лідером команди, котрі діють відповідно до встановлених правил. До прикладу, воїн не може робити того, що може зробити чародій, будь-яка роль користується особливостями, які набула, та деколи від неї залежить і розвиток сюжету. У ігрового процесу є ціль, яка полягає у виконанні різних завдань, що

сприяє розвитку одного персонажа або групи. Є різні варіанти подій, які залежать від обраного образу діяча, де попередньо визначено, або сформовано самим гравцем. Частіше всього рольові ігри поєднуються з іншими жанрами, що відповідно утворює Action-RPG, тактичні рольові ігри, MUD-и і т.п.. MMORPG – це рольові багатокористувацькі онлайн ігри, де гравці взаємодіють один з одним у віртуальному світі за допомогою чату, голосових чатів, що відіграє головну роль ігрового процесу. Деколи такі, рольові відеоігри поділяються за дизайном і конструкцією сюжету. Таким чином, є умовний розподіл на рольові ігри західного зразка та східного.

Симулятор – за широким розумінням, всі ігри являються симуляторами. В тоншому значенні - це комп’ютерні ігри, призначені для складання уявлення про дійсність за підтримкою відображення певних реальних явищ та властивостей у штучному (віртуальному) середовищі. Є багато під жанрів, як технічних, аркадних, спортивних, економічних, та інших

Пригоди - в цьому жанрі відеоігри створенні для того, щоб гравець керував ігровим персонажем, у якого є можливість рухатися по сюжету та виконувати зумовлені сценарієм завдання, де він може покладається на свою уважність та логіку, де він втілює пошуки підказок і розв'язує головоломки.

В центрі пригодницького жанру видаються головні під жанри: інтерактивні фільми, інтерактивна літератур та візуальні романи.

Нерідко за аналогічністю до пригодницьких кінофільмів пригодницькими називаються ті відеоігри, де сюжет динамічно розгортається, де відбуваються яскраві події, швидко змінюється обстановка, де персонажі виказують здогадливість та безстрашність, а не грубу силу. Приклади: Володар сторінок, Подорож до центру Землі, серія про Індіану Джонса, Disney's Aladdin in Nasira's Revenge, Syberia.

1.3 Постановка задачі

1.3.1 Призначення розробки

Дане програмне забезпечення призначене для людей, які при виборі ігор зважають на різноманітні специфічні критерії, а також для людей, які бажають поділитися своїми враженнями від проходження або перегляду ігор з іншими.

1.3.2 Цілі та задачі розробки

Метою розробки є полегшення процесу пошуку ігор за різними критеріями, такими як жанром гри, її видавцем, роком виходу, та іншими, обумовленими певними вподобаннями користувачів, а також для спрощення перегляду необхідної інформації про ігри, та для спрощення обговорення гри.

Для досягнення поставленої мети необхідно розв’язати наступні задачі:

– реєстрація та авторизація користувачів;

– перегляд, створення, редагування та видалення ігор, жанрів, та інших ігрових категорій адміністраторами;

– можливість коментування;

– можливість оцінювання ігор;

– пошук ігор за назвою, жанрами, видавцями, платформами, роком виходу.

1.4 Аналіз відомих програмних продуктів

Вже існуть різні веб-застосунки для обговорення та пошуку ігор, так званих ігрових бібліоте. У кожного з них є свої переваги та недоліки. Для клієнта зручніше всього знайти один сайт, і постійно користуватись лише ним. З свого досвіду, та досвіду своїх знайомих я зробив висновок, що для користувача головним є зручний в використанні інтерфейс, та гнучкий пошук ігор. Коли користувач не знає ігор які він хотів би пограти, він заходить на такі сайти з надією що там буде зручний пошук. Під зручним пошуком я розумію

можливість відфільтрувати ігри одразу не за одним, а за декількома жанрами. Також важливим фактором при пошуку є ігрова платформа, так як користувачу не потрібно виводити ігри для консолей, якщо в нього є лише ПК. Зауважую, що для аналізу було обрано сайти, написані іноземними мовами, оскільки не було знайдено україномовних варіантів, що надають достатній об’єм функціональності.

1.4.1 Аналіз веб-застосування Igromania.ru

Igromania.ru (Рисунок 1.1 -) містить певні переваги та недоліки.

Перелік переваг:

– є сторінка з блогами;
– є сторінка з галереями;
– розділи з комп’ютерним обладнанням.

Перелік недоліків:

– відсутність пошуку по декільком жанрам;
– відсутність пошуку по декільком платформам;
– занадто багато відділів на сайтах, відвертає увагу;
– незручні сторінки з описом гри – вони перериваються рекламою та іншими блоками з схожою інформацією.

Рисунок 1.1 - Інтерфейс веб-застосування Igromania.ru

1.4.2 Аналіз веб-застосування Gamer.ru

Gamer.ru (Рисунок 1.2) містить певні переваги та недоліки.

Перелік переваг:

– розділ про кіно;

– розділи з комп’ютерним обладнанням.

Перелік недоліків:

– відсутність пошуку по декільком жанрам;

– відсутність пошуку по декільком платформам;

– застарілий інтерфейс.

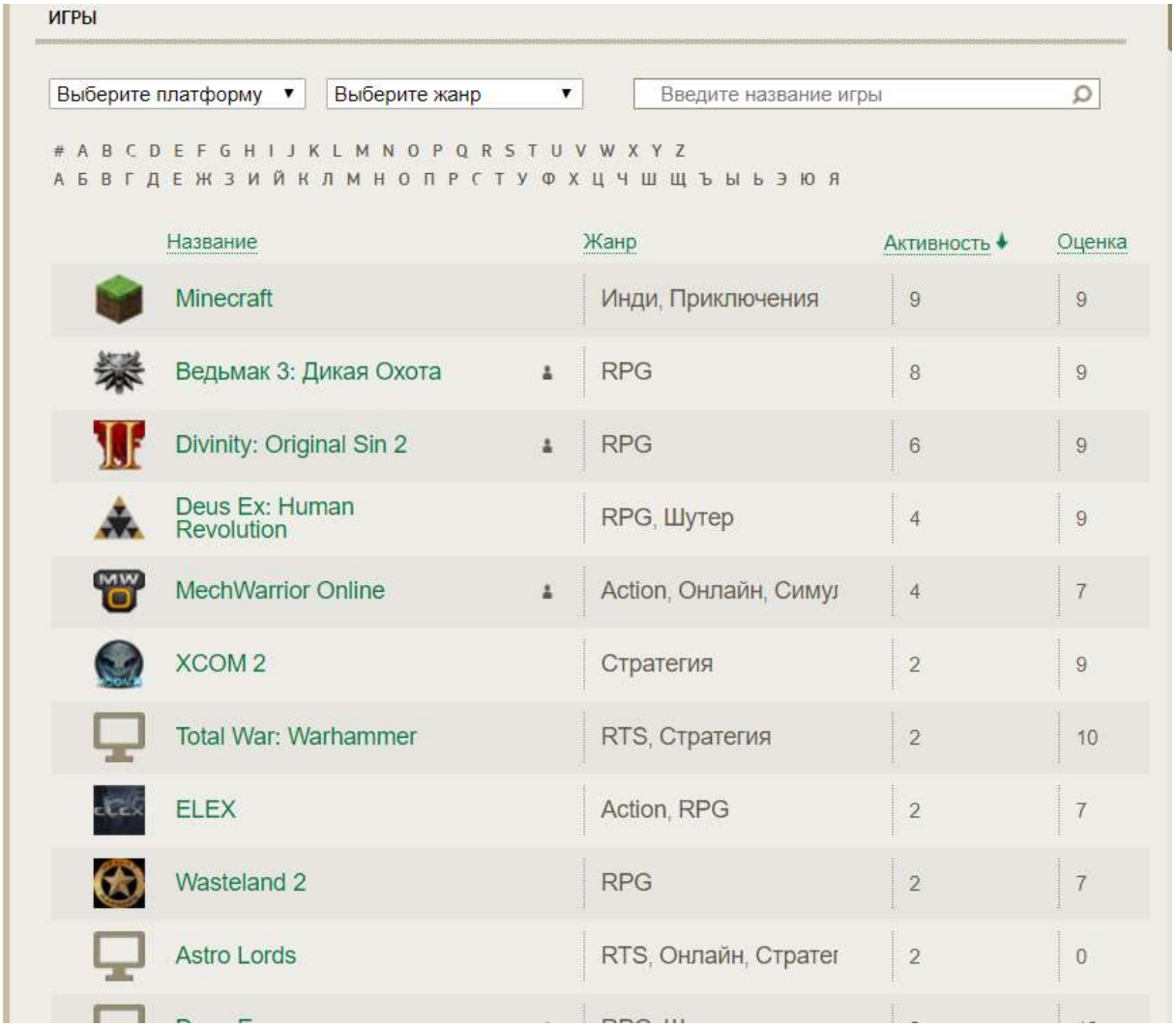


Рисунок 1.2 - Інтерфейс веб-застосування Gamer.ru

1.5 Аналіз вимог до програмного забезпечення

В системі передбачаються три ролі: гість, користувач, та адміністратор.

Гостями є всі неавторизовані користувачі. Гість може переглядати майже всі сторінки – Ігри, Новини, Жанри, Розробники, Платформи, а відповідні сторінки для однієї гри, новини, і так далі.

Користувач має розширений відносно гостя функціонал. Він може оцінювати та коментувати ігри.

Адміністратор має права на всі можливі операції. Саме адміністратори будуть займатись заповненням бази даних актуальною інформацією. Вони

можуть редагувати, додавати, видаляти різні сутності, а саме – ігри, видавців, платформи, новини.

Популярним та зручним способом задання вимог є діаграми використання (use-case diagrams), вона наведена у додатку «Схема структурних варіантів використань».

Нижче наведено таблицю із ідентифікатором використання та найменуванням

Таблиця 1.1 - Варіанти використання

Ідентифікатор	**Назва**
UC01	Реєстрація
UC02	Авторизація
UC03	Перегляд даних про гру
UC04	Редагування даних про гру
UC05	Перегляд даних про жанр
UC06	Редагування даних про жанр
UC07	Перегляд даних про розробника
UC08	Редагування даних про розробника
UC09	Перегляд даних про ігрову платформу
UC10	Редагування даних про ігрову платформу
UC11	Перегляд даних про новину
UC12	Редагування даних про новину

Продовження таблиці 1.1

UC13	Коментування гри
UC14	Оцінювання гри

1.5.2 Розроблення функціональних вимог

На основі наведених вище варіантів використання було сформульовано наступні функціональні вимоги.

Таблиця 1.2 - Опис функціональних вимог

Ідентифікатор	**Опис**	**Пріоритет**
REQ01	Система має дозволяти зареєстрованому користувачу авторизуватись у веб-застосуванні	Високий
EQ02	Система має надавати можливість користувачу зареєструватись у веб-застосуванні	Високий
REQ03	Система має надавати можливість всім типам користувачів переглядати дані про гру	Високий
REQ04	Система має надавати можливість авторизованому адміністратору редагувати дані про гру	Високий

Продовження таблиці 1.2

REQ05	Система має надавати можливість всім типам користувачів переглядати дані про жанр	Високий
REQ06	Система має надавати можливість авторизованому адміністратору редагувати дані про жанр	Високий
REQ07	Система має надавати можливість всім типам користувачів переглядати дані про гру розробника	Високий
REQ08	Система має надавати можливість авторизованому адміністратору редагувати дані про розробника	Високий
REQ09	Система має надавати можливість всім типам користувачів переглядати дані про ігрову платформу	Високий
REQ10	Система має надавати можливість авторизованому адміністратору редагувати дані про ігрову платформу	Високий
REQ11	Система має надавати можливість всім типам користувачів переглядати дані про новину	Середній

Продовження таблиці 1.2

REQ12	Система має надавати можливість авторизованому адміністратору редагувати дані про новину	Середній
REQ13	Система має надавати можливість авторизованому користувачу та адміністратору коментувати ігри	Середній
REQ14	Система має надавати можливість авторизованому користувачу та адміністратору оцінювати ігри	Середній
REQ 15	Система має надавати можливість всім типам користувачів можливість переглядати коментуварі	Середній
REQ 16	Система має надавати можливість всім типам користувачів можливість переглядати оцінки ігор.	Середній

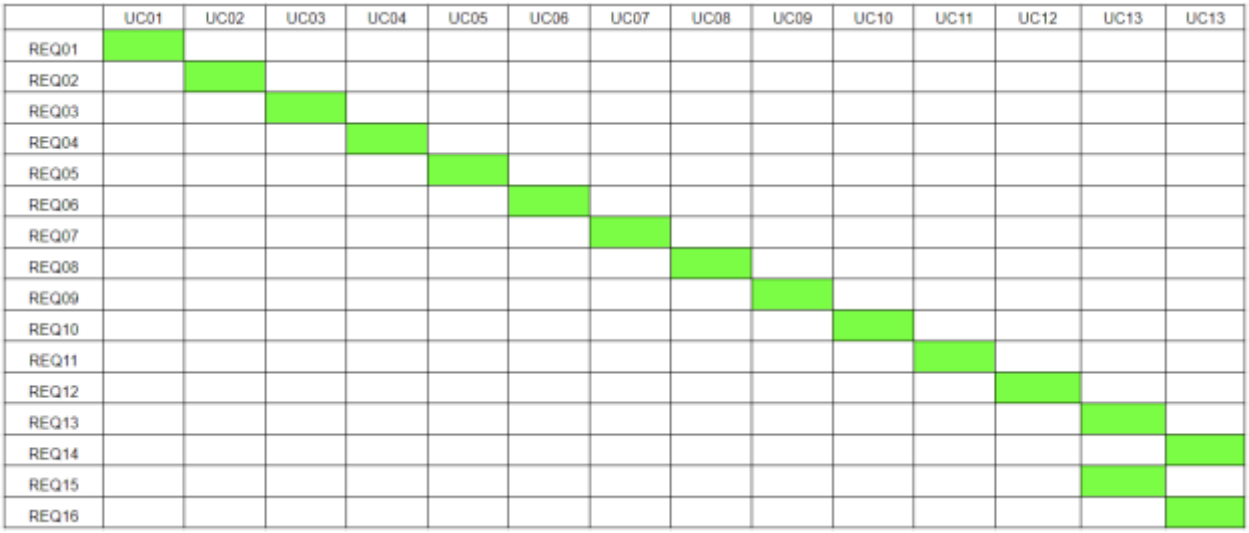


Рисунок 1.3 - Інтерфейс веб-застосування Gamer.ru

1.5.3 Розроблення нефункціональних вимог

Веб-застосування Ігрова Бібліотека складається із серверної частини та клієнтського додатку.

Сервер надає Application Program Interface для роботи з іграми, новинами, жанрами, обліковими записами користувачів, та іншими даними. Клієнстький додаток посилає запити на ендпоінти серверної частини для отримання та обробки всієї інформації необхідної для роботи користувача. Дані відправляються у форматі JSON, це текстовий формат обміну даними між комп'ютерами, який базується на тексті, що може бути прочитаним людиною.. Авторизація реалізована з використанням JSON web token, зареєстровані користувачі мають зберігаються в базі даних додатку. Усі паролі мають зберігатися у захешованому вигляді. Дані сервера мають бути ізольовані від клієнтського додатку.

Дане програмне забезпечення має відповідати наступним нефункціональним вимогам:

– коректний показ в браузері Google Chrome версії 48.0 та вище;

– мова клієнтського інтерфейсу: українська;

Висновок до розділу

У даному розділі була проаналізована предметна область необхідна для даного застосунку. Були розглянуті різні типи інформації про ігри, їхні класифікації.

Були описані основні сценарії роботи для даного веб-застосування, а також розглянути необхідні ролі користувачів в системі.

Також в цьому розділі описані нефункціональні та функціональні вимоги, а також сценарії використання. Крім того були проаналізовані існуючі аналоги для того щоб при розробці програмного забезпечення уникнути недоліків допущених в них.

Так як ігрова індустрія стрімко розвивається, а кількість геймерів росте з кожним роком, була доведена актуальність розробки даного веб-застосування.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

На рисунку 2.1 наведено BPMN діаграму, яка описує процес обробки запиту отримання ігор за вподобаннями користувача. Користувач налаштовує фільтр для ігор за жанрами, видавцем, та іншими критеріями, і клієнтський додаток відправляє запит на сервер. Сервер формує відповідний запит в базу даних. Після отримання результату йде перетворення даних з таблиць в моделі які потім повертає клієнтському додатку. Після цього клієнтський додаток коректно відображає результати пошуку на сторінці, і користувач може переглянути його.

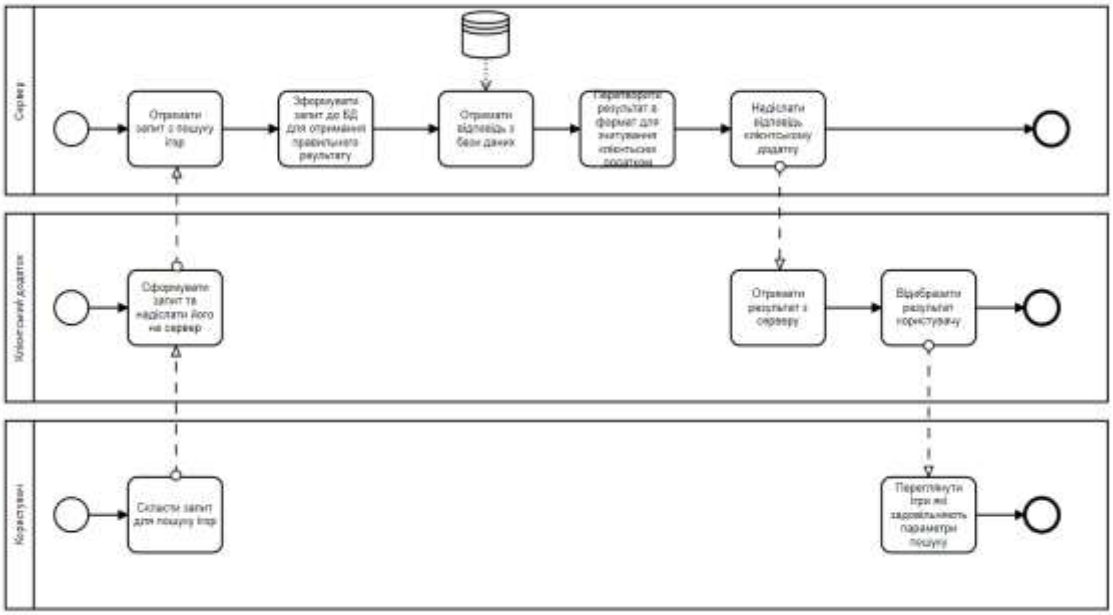


Рисунок 2.1 - Схема бізнес процесу пошуку ігор за вподобаннями

2.2 Архітектура програмного забезпечення

2.2.1 Опис використаних рішень

Для створення серверної частини мовою програмування було вибрано C#. В якості середи розробки слугувала Visual Studio 2019. Так, як розроблюване застосування є клієнт-серверним програмним забезпеченням з

точною потребою у гнучкості у процесі розробки, то для його написання було вирішено використовувати .NET Core, як найсучасніший фреймворк для C#.

.NET Core - це модульна платформа для розробки програмного забезпечення з відкритим вихідним кодом. Сумісна з такими операційними системами як Windows, Linux і macOS. Була випущена компанією Microsoft. .NET Core заснована на .NET Framework. Платформа .NET Core відрізняється від неї модульностью, кроссплатформенную, можливістю застосування хмарних технологій, і тим, що в ній відбувся поділ між бібліотекою CoreFX і середовищем виконання CoreCLR. .NET Core - модульна платформа. Кожен її компонент оновлюється через менеджер пакетів NuGet, а значить можна оновлювати її модулі окремо, в той час як .NET Framework оновлюється цілком. Кожна програма може працювати з різними модулями і не залежить від єдиного поновлення платформи.

Важливою частиною .NET Core є контейнери для підтримки Inversion of Control (інверсії управління). Інверсія управління або Ioc – це такий принцип побудови програми за якого її частини переважно отримують потік керування з спільної бібліотеки. Однією з найчастіше використовуваних реалізацій цього принципу є Dependency Injection (Впровадження залежностей). При впровадженні залежностей об’єкти у програмі не створюють самі об’єкти від яких вони залежні, натомість ця за цю задачу в цьому випадку відповідає IoC контейнер.

При розробці веб-застосування було вирішено використовувати трирівневу архітектуру. У комп'ютерних технологіях трирівнева архітектура, синонім триланкова архітектура (англ. three-tier або Multitier architecture) передбачає наявність наступних компонент програми: клієнтський застосунок (зазвичай говорять «тонкий клієнт» або термінал), підключений до сервера застосунків, який в свою чергу підключений до серверу бази даних.

Рисунок 2.2 - Схема для представлення трирівневої архітектура

Тобто серверна частина поділена на три рівні.

Рівень представлення – відповідає за отримання запитів, перенаправлення їх для обробки в необхідні сервіси шару Бізнес-логіки. Для цього шару використовується фреймворк ASP.NET Core. ASP.NET Core — вільне та відкрите програмне забезпечення каркаса вебзастосунків, з продуктивністю вищою ніж у ASP.NET, розроблена корпорацією Microsoft і співтовариством . Це модульна структура, яка працює як на повній платформі .NET Framework, так і на платформі .NET Core. Фреймворк являє собою повний перепис, який об'єднує раніше окремі ASP.NET MVC та ASP.NET Web API у єдину програмувальну модель.

Рівень Бізнес-логіки. Бізнес-логіка - в розробці інформаційних систем - сукупність правил, принципів, залежностей поведінки об'єктів предметної області (сфери людської діяльності, яку система підтримує). Інакше можна сказати, що бізнес-логіка - це реалізація правил і обмежень автоматизуюємих операцій. Є синонімом терміна «логіка предметної області» (англ. Domain logic). Бізнес-логіка задає правила, яким підкоряються дані предметної області. Простіше кажучи, бізнес-логіка - це реалізація предметної області в

інформаційній системі. До неї відносяться, наприклад, формули розрахунку щомісячних виплат по позиках (у фінансовій індустрії), автоматизована відправка повідомлень електронної пошти керівника проекту після закінчення виконання частин завдання всіма підлеглими (в системах управління проектами), відмова від готелю при скасуванні рейсу авіакомпанією (в туристичному бізнесі) і т.д. У цьому рівні відбувається основна обробка отриманих даних, та повернення результату на рівень Представлення.

Рівень доступу до даних. Шар доступу до даних (Data Access Layer - DAL) в програмному забезпеченні - це шар комп'ютерної програми, який надає спрощений доступ до даних, що зберігаються в постійному сховище будь-якого типу, такому як реляційна база даних. Цей акронім в основному використовується в Microsoft ASP.NET оточенні.

Для прикладу, DAL може повертати посилання на об'єкт (в термінах об'єктно-орієнтованого програмування) з його атрибутами замість рядків полів з таблиці бази даних. Це дозволяє створювати клієнтські (або призначені для користувача) модулі з більш високим рівнем абстракції. Такого роду модель може бути реалізована шляхом створення класу з методами доступу до даних, які безпосередньо посилаються на відповідний набір процедур бази даних. Інша реалізація може потенційно отримувати або записувати записи в або з файлової системи. DAL приховує складність лежить в основі сховища даних від зовнішнього світу.

Натомість використання таких команд як «створити», «видалити» або «оновити» в певній таблиці в базі, клас і кілька збережених процедур можуть бути створені в базі. Ці процедури можуть викликатися з методу всередині класу, який поверне об'єкт, що містить запитані значення. Або команди створення, видалення та оновлення можуть бути виконані всередині простих функцій як registerUser або loginUser, збережені в шарі доступу до даних.

Також методи бізнес-логіки з програми можуть бути співвіднесені до шару доступу до даних. Так для прикладу, замість створення запиту до бази

даних, щоб отримати всіх користувачів з кількох таблиць, додаток може зробити один виклик методу з DAL для цього додатка.

Для доступу до даних був використаний Entity Framework Core. Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну і расширяемую технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати базами даних, але є більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Entity Framework Core підтримує безліч різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-якої СУБД, якщо для неї є потрібний провайдер.

Для побудови клієнтної частини застосунку було вирішено використовувати популярний фреймворк Angular (2+). Angular (зазвичай так називають фреймворк Angular 2 або Angular 2+, тобто вищі версії) — написаний на TypeScript front-end фреймворк з відкритим кодом, який розробляється під керівництвом Angular Team у компанії Google, а також спільнотою приватних розробників та корпорацій. Angular — це AngularJS, який переосмислили та який був повністю переписаний тією ж командою розробників.

Односторінковий застосунок (англ. single-page application, SPA), також відомий як односторінковий інтерфейс (англ. single-page interface, SPI) - це веб-застосунок чи веб, який вміщується на одній сторінці з метою забезпечити користувачеві досвід близький до користування настільною програмою.В односторінковому застосунку весь необхідний код - HTML, JavaScript, та CSS - завантажується разом зі сторінкою , або динамічно довантажується, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє

користувача до іншої сторінки у процесі роботи з нею. Взаємодія з односторінковим застосунком часто включає в себе динамічний зв'язок з веб-сервером

2.2.2 Опис сховища даних

Будь-яке застосування, який має зберігати дані користувачів має мати своє сховище даних Структура бази даних (схема з відображенням відношень між її сутностями) зображена в додатку «Структура бази даних». На схемі не відображена частина системних таблиць які використовуються .NET Core для авторизації користувачів. Для розуміння роботи достатньо таблиці ApplicationUser з частиною полів. Проаналізуємо основі відношення.

ApplicationUser - таблиця з даними користувачів, які будуть використовуватись для її автентифікації у веб-застосунку.

Таблиця 2.3 - Опис таблиці User

Назва поля	**Тип**	**Первинний ключ**	**Обов’язковий**	**Опис**
id	Varchar(256)	Так	Так	Ідентифікатор сутності
Email	Integer	Ні	Так	Ідентифікатор пов’язаної персони
UserName	Varchar(64)	Ні	Так	Унікальне ім’я користувача веб-застосунку

Одна з основних таблиць в базі даних з інформацією про ігри - Games

Таблиця 2.4 - Опис таблиці Games

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
Id	int	Так	Так	Ідентифікатор сутності
Name	Nvarchar(450)	Ні	Так	Назва
Description	Nvarchar (Max)	Ні	Ні	Опис
isDeleted	bit	Ні	Ні	Чи видалений запис
PublisherId	int		Ні	Ідентифікатор Видавця
Year	int	Ні	Ні	Дата випуску

Таблиця 2.5 - Опис таблиці Genres

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
Id	int	Так	Так	Ідентифікатор сутності
Name	Nvarchar(450)	Ні	Так	Назва
Description	Nvarchar (Max)	Ні	Ні	Опис
isDeleted	bit	Ні	Ні	Чи видалений запис

Таблиця 2.6 - Опис таблиці PlatformTypes

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
Id	int	Так	Так	Ідентифікатор сутності
Name	Nvarchar(450)	Ні	Так	Назва
Description	Nvarchar (Max)	Ні	Ні	Опис
isDeleted	bit	Ні	Ні	Чи видалений запис

Таблиця 2.7 - Опис таблиці Publishers

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
Id	int	Так	Так	Ідентифікатор сутності
Name	Nvarchar(450)	Ні	Так	Назва
Description	Nvarchar (Max)	Ні	Ні	Опис
isDeleted	bit	Ні	Ні	Чи видалений запис

Таблиця 2.8 - Опис таблиці Comments

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
Id	int	Так	Так	Ідентифікатор сутності
GameId	Int	Ні	так	Ідентифікатор гри
UserId	Nvarchar(450)	Ні	Так	Ідентифікатор користувача
Body	Nvarchar (Max)	Ні	Так	Сам коментар
isDeleted	bit	Ні	Ні	Чи видалений запис

Таблиця 2.9 - Опис таблиці GamesGenres

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
GameId	int	Так	Так	Ідентифікатор сутності Гри
GenreId	int	Так	Так	Ідентифікатор сутності Жанру

Таблиця 2.10 - Опис таблиці GamesPlatformTypes

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
GameId	int	Так	Так	Ідентифікатор сутності Гри
PlatformTypeId	int	Так	Так	Ідентифікатор сутності Жанру

Таблиця 2.11 - Опис таблиці GamesNews

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
GameId	int	Так	Так	Ідентифікатор сутності Гри
NewsId	int	Так	Так	Ідентифікатор сутності Жанру

Таблиця 2.12 - Опис таблиці News

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
Id	int	Так	Так	Ідентифікатор сутності
Name	Nvarchar(450)	Ні	Так	Назва
Description	Nvarchar (Max)	Ні	Ні	Опис
isDeleted	bit	Ні	Ні	Чи видалений запис

Таблиця 2.13 - Опис таблиці Marks

Назва поля	**Тип**	**Первинний ключ**	**Обов'язковий**	**Опис**
UserId	string	Так	Так	Ідентифікатор сутності
GameId	int	Так	Так	Назва
Mark	int	Ні	Так	Оцінка

2.2.3 Опис модулів коду серверної частини

Код серверної частини складається з проектів, список яких наведений нижче.

– GameLibrary.WEB – реалізує рівень Представлення в трьохрівневій архітектурі. Містить API серверної частини застосування.

– GameLibrary.Contracts – містить так звані контракти (інтерфейси) які використовуються в GameLibrary.WEB, та які повинен реалізувати шар Бізнес-логіки. Також містить моделі якими опирує шар Бізнес-логіки.

– GameLibrary.BusinessLogic – реалізує рівень Бізнес-логіки в трьохрівневій архітектурі. Містить сервіси які реалізують всю необхідну для роботи додатку функціональність.

– GameLibrary.DataAccess – реалізує рівень доступу до даних в трьохрівневій архітектурі. Інкапсулює в собі всі методи для зберігання, читання, та зміни даних в БД.

Нижче на рисунку 2.3 зображено структурну схему класів сервісів

Рисунок 2.3 - Схема структурна інтерфейсів сервісів

Рисунок 2.4 - Схема структурна об’єктів рівня доступу до даних

Нижче, у таблиці 2.14, можна побачити опис основних класів цього програмного забезпечення.

Таблиця 2.14 - Опис основних класів та інтерфейсів

Назва проекту	**Назва**	**Тип**	**Опис**
GameLibrary.WEB	ApplicationUserController	Клас	Клас, який є контролером, що буде обробляти запити які відносяться до користувачів
GameLibrary.WEB	CommentsController	Клас	Клас, який є контролером, що буде обробляти запити які відносяться до коментарів
GameLibrary.WEB	GamesController	Клас	Клас, який є контролером, що буде обробляти запити які відносяться до ігор

Продовження таблиці 2.14

GameLibrary.WEB	PublishersController	Клас	Клас, який є контролером, що буде обробляти запити які відносяться до видавців
GameLibrary.WEB	WebApiDependencies	Клас	Допоміжний клас для налаштування Dependency Injection (Впровадження залежностей) для рівня Представлення.
GameLibrary.WEB	Program	Клас	Клас з якого запускається серверна частина додатку
GameLibrary.WEB	Startup	Клас	Клас, в якому відбувається конфігурація додатку

Продовження таблиці 2.14

GameLibrary.Contracts	IApplicationUserService	Інтерфейс	Інтерфейс для сервісу який буде відповідати за роботу з користувачами
GameLibrary.Contracts	ICommentService	Інтерфейс	Інтерфейс для сервісу який буде відповідати за роботу з коментарями
GameLibrary.Contracts	IGameService	Інтерфейс	Інтерфейс для сервісу який буде відповідати за роботу з іграми
GameLibrary.Contracts	IGenreService	Інтерфейс	Інтерфейс для сервісу який буде відповідати за роботу з жанрами
GameLibrary.Contracts	INewsService	Інтерфейс	Інтерфейс для сервісу який буде відповідати за роботу з новинами

Продовження таблиці 2.14

GameLibrary.Contracts	IPlatformTypeService	Інтерфейс	Інтерфейс для сервісу який буде відповідати за роботу з ігровими платформами
GameLibrary.Contracts	IPublisherService	Інтерфейс	Інтерфейс для сервісу який буде відповідати за роботу з видавцями
GameLibrary.Contracts	ApplicationUserModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для роботи з реєстрацією
GameLibrary.Contracts	CommentModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для роботи з коментарями

Продовження таблиці 2.14

GameLibrary.Contracts	GameModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для роботи з іграми
GameLibrary.Contracts	NewsModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для роботи з новинами
GameLibrary.Contracts	MarkModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для отримання інформації про оцінку гри

Продовження таблиці 2.14

GameLibrary.Contracts	MarkInfoModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для отримання детальної інформації про оцінку гри
GameLibrary.Contracts	SearchGameModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для роботи з жанрами
GameLibrary.Contracts	GenreModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для роботи з жанрами

Продовження таблиці 2.14

GameLibrary.Contracts	LoginModel	Клас	Клас, який відповідає JSON об'єкту, який може отриманий від клієнта для авторизації
GameLibrary.Contracts	LoginResultModel	Клас	Клас, який використовуєтьс я для зберігання результату авторизації
GameLibrary.Contracts	NewsModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для роботи з новинами
GameLibrary.Contracts	PublisherModel	Клас	Клас, який відповідає JSON об'єкту, який може надісланий або отриманий від клієнта для роботи з видавцями

Продовження таблиці 2.14

GameLibrary.BusinessLogic	BusinessLogicDependencies	Клас	Допоміжний клас для налаштування Dependency Injection (Впровадження залежностей) для рівня Бізнес-логіки.
GameLibrary.BusinessLogic	BusinessLogicMappingProfile	Клас	Допоміжний клас для налаштування перетворення об’єктів з рівня доступу до даних в моделі з рівня Представлення і навпаки
GameLibrary.BusinessLogic	ApplicationUserService	Клас	Клас, який реалізує базовий інтерфейс I ApplicationUserService
GameLibrary.BusinessLogic	NewsService	Клас	Клас, який реалізує базовий інтерфейс INewsService

Продовження таблиці 2.14

GameLibrary.BusinessLogic	CommentService	Клас	Клас, який реалізує базовий інтерфейс ICommentService
GameLibrary.BusinessLogic	GameService	Клас	Клас, який реалізує базовий інтерфейс IGameService
GameLibrary.BusinessLogic	PlatformTypeService	Клас	Клас, який реалізує базовий інтерфейс IPlatformTypeService
GameLibrary.BusinessLogic	PublisherService	Клас	Клас, який реалізує базовий інтерфейс IPublisherService
GameLibrary.BusinessLogic	UserService	Клас	Інтерфейс для об’єкту, що буде оброблювати дані користувача

Продовження таблиці 2.14

GameLibrary.Data Access	DataAccessDependencies	Клас	Допоміжний клас для налаштування Dependency Injection (Впровадження залежностей) для рівня доступу до бази даних.
GameLibrary.Data Access	GameContext	Клас	Наслідує IdentityContext, сам контекст для роботи з базою даних
GameLibrary.Data Access	ApplicationUser	Клас	Клас, який реалізує модель таблиці ApplicationUser з бази даних
GameLibrary.Data Access	Comment	Клас	Клас, який реалізує модель таблиці Comment з бази даних

Продовження таблиці 2.14

GameLibrary.Data Access	Mark	Клас	Клас, який реалізує модель таблиці Marks з бази даних
GameLibrary.Data Access	Game	Клас	Клас, який реалізує модель таблиці Game з бази даних
GameLibrary.Data Access	GameGenre	Клас	Клас, який реалізує модель таблиці GameGenre з бази даних
GameLibrary.Data Access	GameNews	Клас	Клас, який реалізує модель таблиці GameNews з бази даних
GameLibrary.Data Access	GamePlatform	Клас	Клас, який реалізує модель таблиці GamePlatform з бази даних
GameLibrary.Data Access	Genre	Клас	Клас, який реалізує модель таблиці Genre бази даних

Продовження таблиці 2.14

GameLibrary.Data Access	News	Клас	Клас, який реалізує модель таблиці News з бази даних
GameLibrary.Data Access	PlatformType	Клас	Клас, який реалізує модель таблиці PlatformType з бази даних
GameLibrary.Data Access	Publisher	Клас	Клас, який реалізує модель таблиці Publisher з бази даних
GameLibrary.Data Access	CommentConfiguration	Клас	Допоміжний клас для конфігурації таблиці Comment з бази даних
GameLibrary.Data Access	GameConfiguration	Клас	Допоміжний клас для конфігурації таблиці Game з бази даних

Продовження таблиці 2.14

GameLibrary.Data Access	MarkConfiguration	Клас	Допоміжний клас для конфігурації таблиці Mark з бази даних
GameLibrary.Data Access	GameGenreConfiguration	Клас	Допоміжний клас для конфігурації таблиці GameGenre з бази даних
GameLibrary.Data Access	GameNewsConfiguration	Клас	Допоміжний клас для конфігурації таблиці GameNews з бази даних
GameLibrary.Data Access	GamePlatformConfiguration	Клас	Допоміжний клас для конфігурації таблиці GamePlatform з бази даних
GameLibrary.Data Access	NewsConfiguration	Клас	Допоміжний клас для конфігурації таблиці News з бази даних

Продовження таблиці 2.14

GameLibrary.Data Access	PlatformTypeConfiguration	Клас	Допоміжний клас для конфігурації таблиці PlatformType з бази даних
GameLibrary.Data Access	PublisherConfigurations	Клас	Допоміжний клас для конфігурації таблиці Publisher з бази даних
GameLibrary.Data Access	IUnitOfWork	Клас	Інтерфейс, який реалізує паттерн Одиниця роботи (UnitOfWork) декларує всі поля та методи необіхдні для роботи сервісів з даними.
GameLibrary.Data Access	UnitOfWork		Клас, який реалізує інтерфейс IUnitOfWork

Продовження таблиці 2.14

GameLibrary.Data Access	IRepository	Клас	Інтерфейс, який використовують ся за паттерну Репозиторій. Декларує методи які повинні реалізувати репозиторії для взаємодії з базою даних.
GameLibrary.Data Access	LoginResult	Клас	Модель для зберігання результату авторизації
GameLibrary.Data Access	ApplicationUserRepository	Клас	Клас, який реалізує інтерфейс IRepository, для взаємодії з базою даних при роботі з користувачами

Продовження таблиці 2.14

GameLibrary.Data Access	CommentRepository	Клас	Клас, який реалізує інтерфейс IRepository, для взаємодії з базою даних при роботі з коментарями
GameLibrary.Data Access	GameRepository	Клас	Клас, який реалізує інтерфейс IRepository, для взаємодії з базою даних при роботі з іграми
GameLibrary.Data Access	GenreRepository	Клас	Клас, який реалізує інтерфейс IRepository, для взаємодії з базою даних при роботі з жанрами

Продовження таблиці 2.14

GameLibrary.Data Access	NewsRepository	Клас	Клас, який реалізує інтерфейс IRepository, для взаємодії з базою даних при роботі з новинами
GameLibrary.Data Access	PlatformTypeRepository	Клас	Клас, який реалізує інтерфейс IRepository, для взаємодії з базою даних при роботі з ігровими платформами
GameLibrary.Data Access	PublisherRepository	Клас	Клас, який реалізує інтерфейс IRepository, для взаємодії з базою даних при роботі з видавцями

2.2.4 Опис модулів коду клієнтської частини

Клієнтська частина веб-застосування побудована за допомогою платформи Angular 9 на мовах програмування TypeScript (надбудова над Javascript), HTML, SCSS. Для виконання асинхронних операцій, таких як отримання даних з сервера, використовується бібліотека RxJS. Розміщення елементів на сторінці спрощується, завдяки використанню модулю макету контейнера Flexbox.

Для базової стилізації деяких елементів, таких як поля вводу, використовується бібліотека Angular Material.

Загалом проєкт має таку структуру:

а) admin – містить компоненти, що відповідають за адміністрування веб-застосування:

– AdminComponent – компонент, що відповідає за відображення вмісту сторінки адміністратора;

– AdminGameComponent – компонент, що відповідає за додавання та редагування ігор;

– AdminGenreComponent – компонент, що відповідає за додавання та редагування жанрів;

– AdminPlatformComponent – компонент, що відповідає за додавання та редагування ігрових платформ;

– AdminPublisherComponent – компонент, що відповідає за додавання та редагування видавців;

– AdminNewsComponent – компонент, що відповідає за додавання та редагування новин;

б) comments – містить компоненти які відповідають за коментування ігор;

в) games – містить компоненти, що відповідають за відображення списку ігор, детальної інформації про гру, оцінювання ігор, а також за секцію пошуку ігор:

– GamesComponent – компонент для відображення головної сторінки для перегляду ігор;

– GameListItemComponent – компонент для демонстрації певної гри в якості елементу списку ігор.

– MarkComponent – компонент, що відповідає за відображення оцінок гри;

– MarkDialogComponent – компонент, що є діалоговим вікном для виставлення оцінки гри авторизованим користувачем;

– SearchGameComponent – компонент, що відповідає за пошук ігор за різними категоріями;

г) gameCategories – містить в собі компоненти які відповідають за різні категорії ігор:

– genres – містить в собі компоненти які відповідають за відображення ігрових жанрів;

– publishers – містить в собі компоненти які відповідають за відображення ігрових видавців;

– news – містить в собі компоненти які відповідають за відображення ігрових новин;

– platforms – містить в собі компоненти які відповідають за відображення ігрових платформ;

д) user – містить компоненти, що відповідають за відображення сторінок необхідних для роботи з обліковим записом користувача:

– LoginComponent – компонент, який відповідає за сторінку авторизацію користувачів;

– RegistrationComponent – компонент, який відповідає за реєстрацію користувачів;

е) services – містить усі сервіси, які відповідають за взаємодію клієнтської частини веб-застосування з серверною частиною, також сервіси містять допоміжні методи для коректної роботи компонентів;

ж) auth – містить interceptor, який додає токен до кожного запиту, та обробляє помилки пов'язані з авторизацією, та guard, який ставиться на певні маршрути щоб заборонити переходід неавторизованих користувачів на ці сторінки;

Кожен компонент складається з трьох основних файлів: .ts, який містить логіку компонента, .html , який створює макет розташування елементів на сторінці, та .scss , який містить в собі стилі відповідного компонента.

2.3 Безпека паролю та автентифікації

Паролі в базі даних не можна зберігати в вигляді звичайного тексту, тому що при викраденні даних з такої бази зловмисники зможуть зайти через аккаунт користувача та виконувати певні операції від його імені. Крім того, це може призвести до взлому аккаунтів цих користувачів і в інших додатках. Тому паролі потрібно зберігати у вигляді хешу, тоді при викраденні даних зловмисники не зможуть дізнатись сам пароль. Для цієї цілі в даному веб-застосунку використовуються методи з бібліотеки AspNetCore.Identity, і реєстрація та зберігання даних про користувача відбуваються захищено.

Не менш важливим процесом у веб-застосуванні процесом є автентифікація, його безпеку також необхідно забезпечити,. Після операції авторизації, сервер буде надсилати користувачу токен, який буде підтверджувати його особу. Було вирішено використовувати використовувати JSON Web Token. JSON Web Token (JWT) - це відкритий стандарт для створення токенов доступу, заснований на форматі JSON. Як правило, використовується для передачі даних для аутентифікації в клієнт-серверних додатках. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який в подальшому використовує даний токен для підтвердження своєї особи

Висновок до розділу

В даному розділі був описаний процес роботи запиту для пошуку ігор за певними фільтрами з точки зору користувача, там самого веб-застосування.

Були описані архітектурні підходи та використані рішення, а також зазначені їх переваги та необхідність в використанні. Крім того, було описане використання бази даних, показана її фізична діаграма, та описані основні таблиці, їх вміст, та відношення між таблицями.

Крім того, в другому розділі були зазначені основні проекти з яких складається серверна частина веб-застосування. Також, для кращого розуміння були описані всі класи та інтерфейси які в ній використовуються. Завдяки використанню багаторівневої архітектури можна легко розширювати та покращувати функціонал цього веб-застосування. При цьому різні рівні не залежать один від одного, так як кожен верхній рівень використовує лише API нижнього.

В кінці розділу також був розглянутий розглянутий аспект безпеки даних, а саме безпека процесів автентифікації і зберігання паролю.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис процесів тестування

Тестування є важливим етапом розробки програмного забезпечення. Воно допомагає виявити суттєві недоліки та помилки програмного коду. Тестування – це процес аналізу компонента програмного забезпечення для виявлення відмінностей між існуючими та необхідними вимогами, та оцінки функцій елемента.

3.2 Випробування програмного продукту

Тестування програмного забезпечення — проведення контролю над якістю продукту, перевірка відповідності між фактичною та запланованою поведінкою програми з використанням кінцевого набору тестів. Тестування, як процес виявлення помилок та багів не може повністю забезпечити правильність роботи програмного забезпечення у всіх можливих випадках. Тестування лише порівнює поведінку програми у різних ситуаціях зі специфікацією.

частиною (за допомогою веб-браузера).

3.2.1 Мета випробувань

Метою випробувань є визначення працездатності розробленого програмного продукту та відповідності його усім вимогам.

3.2.2 Випробування серверної частини

Для спрощення процесу серверної частини тестування до проєкту була підключена бібліотека Swashbuckle Swagger, яка надає можливість переглянути сторінку з усіма публічними адресами контролерів, а також з списком моделей які можуть служити вхідними параметрами. Корисною особливістю цією бібліотеки є те, що вона надає можливість швидко

здійснювати запити на сервер для тестування програмного продукту.

Для тестування серверної частини була створена окрема база даних, з певними початковими даними, які містили декілька сутностей всіх типів, таких як ігри, жанри, ігрові платформи, та інші.

Під час тестування було перевірено 18 методів контролерів та виконано 43 різних теста. Результати тестування:

– 36 тестів були успішно пройдені з першого разу;

– 2 тести було провалені через помилки при заповнені моделі даними, при повторному проходженні стали успішними;

– 5 тестів були провалені через відсутність перевірки на некоректні чи пусті дані моделі в декількох методах сервісу який містив бізнес-логіку пов’язану з оцінюванням ігор. Після додавання відповідних перевірок проблема була вирішена і повторне тестування було успішним;

Приклади результатів тестів наведені нижче (Рисунки 3.1 – 3.8):

Рисунок 3.1 - Відповідь на запит отримання списку ігор

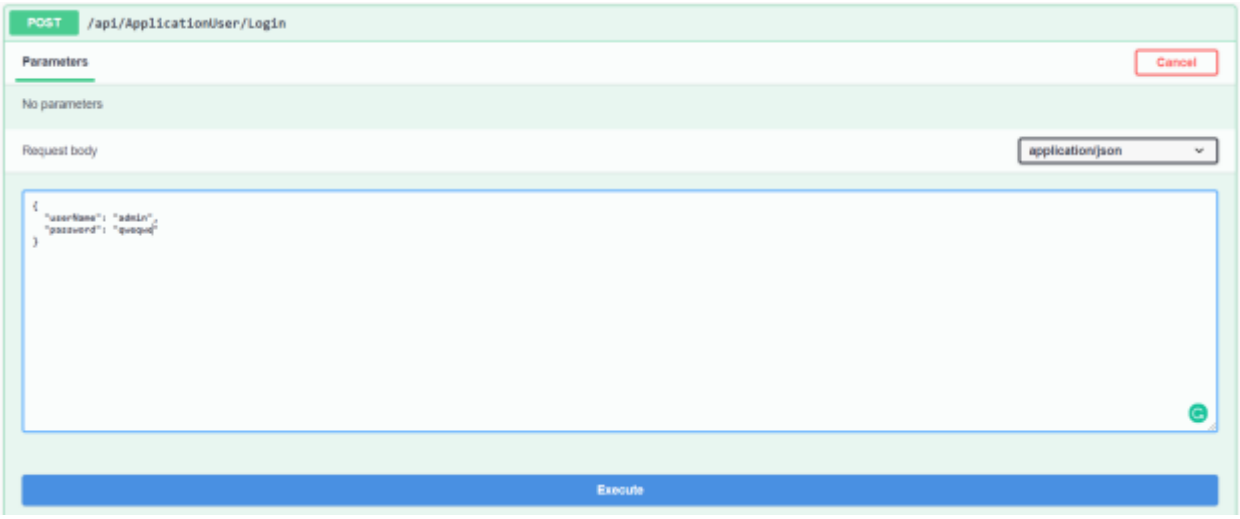


Рисунок 3.2 - Запит на вхід в обліковий запис

Рисунок 3.3 - Відповідь на запит входу в обліковий запис

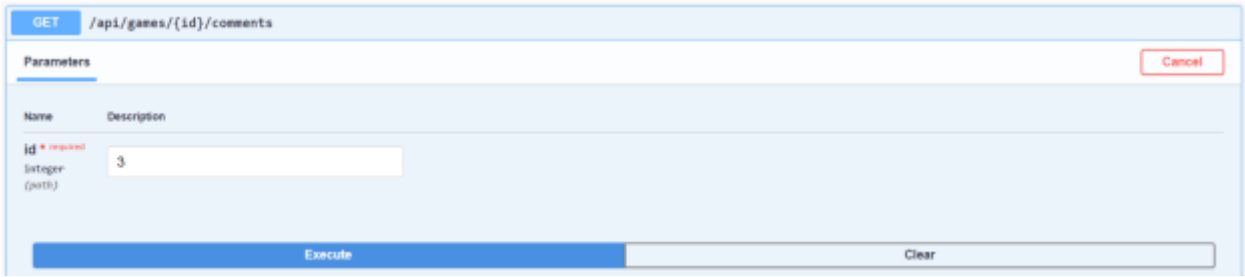


Рисунок 3.4 - Запит на отримання коментарів до гри

Рисунок 3.5 - Відповідь на запит отримання коментарів до гри

Рисунок 3.6 - Запит на отримання ігор за жанром

Рисунок 3.7 - Відповідь на запит отримання списку ігор за жанром

Рисунок 3.8 - Запит на отримання оцінки гри

Рисунок 3.9 - Відповідь на запит отримання оцінки гри

3.2.3 Випробування клієнтської та серверної частин разом

Для тестування роботи обидвох частин програмного забезпечення обрано декілька ключових сценаріїв, які були описані у вигляді тест кейсів. Тест кейси поділяються на негативні і позитивні, в залежності від очікуваного результату. Позитивні тест кейси перевіряють роботу програмного забезпечення з коректними даними, а негативні перевіряють коректність роботи у виняткових ситуаціях.

Далі наведені приклади позитивного (Таблиця 3.1) та негативного Таблиця 4.2) тест кейсів.

Таблиця 3.1 - Опис основних класів та інтерфейсів

Тест кейс	Кроки відтворення	Очікуваний результат
Реєстрація та вхід в обліковий запис	– Перейти на сторінку реєстрації, заповнити поле нікнейму унікальним значенням, ввести пароль з мінімальною довжиною 6 символів. – Натиснути кнопку реєстрації. – перейти на сторінку авторизації, дані з першого кроку та натиснути кнопку авторизації.	Після першого кроку кнопка реєстрації стає доступною. Після другого кроку зареєстрований. Після третього кроку відбувається перехід на сторінку ігор, в навігаційній панелі з’являється кнопка виходу.

Таблиця 3.2 - Опис основних класів та інтерфейсів

Тест кейс	Кроки відтворення	Очікуваний результат
Спроба відкрити сторінку адміністратора користувачем без відповідних прав	– натиснути кнопку реєстрації. – Увійти в обліковий запис адміністратора. – Перейти на сторінку адміністратора. – Скопіювати посилання на сторінку. – Вийти з облікового запису. – Вставити посилання в рядок адреси та натиснути кнопку переходу..	Користувача буде переправлено на сторінку яка містить інформацію про те, що в нього немає необхідних прав доступу.

Висновок до розділу

У даному розділі було перевірено якість розробленого програмного забезпечення. Було виконано тестування веб-застосування в цілому, а також його серверної частини, при цьому знайдено декілька незначних помилок, що підтверджують необхідність продовження тестування.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Описане веб-застосування в основному орієнтоване на операційну систему Windows 10, тому далі описані кроки розгортання для цієї системи. Для використаних фреймворків є аналоги для операційних систем Linux та Mac OS, але вони можуть вимагати додаткових налаштувань.

Далі описаний процес локального розгортання програмного забезпечення, та розгортання на віддалених серверах.

4.1.1 Розгортання серверної частини

Для розгортання серверної частини веб-застосування потрібно:

– встановити Visual Studio 2017 (і вище) з налаштуваннями для .NET та C#;

– встановити SDK .NET Core 3.0 (і вище);

– встановити Microsoft SQL Server 2010 (і вище);

– відкрити проєкт з серверною частиною в IDE Visual Studio;

– запустити проєкт GameLibrary.Web в Internete Information Services (IIS)

4.1.2 Розгортання клієнтської частини

Для розгортання клієнтської частини веб-застосування необхідно:

– встановити Node.js 10-12, пакетний менеджер npm та Angular CLI 9.0.7 (рекомендовано встановити ці версії, так як працездатність веб-застосування з іншими версіями не є гарантованою);

– встановити Visual Code для роботи з кодом програми (необов’язково);

– відкрити папку з проєктом клієнтської частини та встановити необхідні пакети за допомогою команди npm install;

– в файлі environment.ts вказати адресу api серверної частини - змінна apiBaseUrl, за цією адресою будуть надсилатись запити до сервера;

– запустити проєкт за допомогою команди ng serve в консолі;

4.2 Робота з програмним забезпеченням

Детальний опис роботи з даним веб-застосуванням наведено в документі «Керівництво користувача».

Висновок до розділу

У даному розділі був описаний процеси локального розгортання серверної та клієнтської частин даного веб-застосування, а також вказані необхідні компоненти, які повинні бути встановлені для запуску обидвох частин веб-застосування.

ВИСНОВКИ

Під час виконання даного дипломного проєкту було створено веб-застосування для пошуку та обговорення ігор, а також знаходження різної інформації про них, яке дозволяє: шукати ігри за різноманітними критеріями, такими як ігрові жанри, видавці, ігрові платформи, та інші; переглядати інформацію про ігри та їхні категорії в зручному вигляді; оцінювати на коментувати ігри; переглядати список ігрових новин; легко керувати вмістом веб-застосування адміністраторам.

У розділі «Аналіз вимог до програмного забезпечення» було досліджено предметну область, визначено основні сценарії роботи, необхідні функціональні та нефункціональні вимоги, мету та задачі розробки. Були проаналізовані наявні аналоги (їх недоліки і переваги), та знайдені основні переваги даного програмного забезпечення.

У розділі «Моделювання та конструювання програмного забезпечення» була створено структура бази даних, проведений аналіз програмного забезпечення. Крім того, була побудована архітектура як клієнтської, так і серверної частини веб-застосування. .

У розділі «Аналіз якості та тестування програмного забезпечення» були описані процеси тестування програмного продукту, також було описане випробування як тільки серверної частини веб-застосування, так і її взаємодії з клієнтською частиною. Крім того були наведені приклади тест кейсів.

У розділі «Впровадження та супровід програмного забезпечення» було описано процес розгортання веб-застосування на локальній машині.

Розроблене веб-застосування відповідає усім вимогам, забезпечує цілісність даних, має можливість для швидкого розширення через вибір ефективних підходів до проектування, є зручним у використанні як користувачами, так і адміністраторами які будуть підтримувати актуальність інформації в системі.

ПЕРЕЛІК ПОСИЛАНЬ

1) Еспозіто Д. Programming ASP.NET Core. — М.: Prentice Hall Ptr, 2018. – 1215 с.

2) Фрімен А. Angular для профессионалов. — СПб.: Пітер, 2018. — с. 800

3) Шілдт Г. Полный справочник по C#. Пер. с англ. — Видавничий Дім "Вільямс", 2014. – 752с.

4) Documentation TypeScript [Електронний ресурс]. – 2019. – Режим доступу: https://www.typescriptlang.org/

5) RxJS Tutorial [Електронний ресурс]. – 2019. – Режим доступу: http://reactivex.io/rxjs/manual/tutorial.html

6) Angular Material [Електронний ресурс]. – 2020 – Режим доступу: - https://material.angular.io/

7) Документация по .NET Core [Електронний ресурс]. – 2020 – Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/core/

8) Разработка приложений ASP.NET Core [Електронний ресурс]. – 2020 – Режим доступу: https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-3.1

9) Классификация компьютерных игр [Електронний ресурс]. – 2020 Режим доступу: https://ru.wikipedia.org/wiki/Классификация_Компьютерных

10) JWT [Електронний ресурс]. – 2020 – Режим доступу: https://ru.wikipedia.org/wiki/JSON_Web_Token

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” __ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ «ІГРОВА БІБЛІОТЕКА»

Технічне завдання

КПІ.ІП-6127.045440.02.91

“ПОГОДЖЕНО”

Керівник проекту:

_____Ю.М. Крамар

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____А.О. Шатровський

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосування «Ігрова бібліотека»

Галузь застосування:

Наведене технічне завдання поширюється на розробку веб-застосування «Ігрова бібліотека» КПІ.ІП-6127.045440, котра використовується для надання можливості перегляду ігор та призначена для спрощення перегляду та пошуку ігор за інтересами.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосування “Ігрова Бібліотека” є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для людей, які під час пошуку ігор для проведення вільного часу зважають на різні критерії відбору ігор, а також для гравців які хочуть поділитись своїми враженнями з іншими.

Метою розробки є полегшення процесу пошуку ігор за різними критеріями, такими як жанром гри, її видавцем, роком виходу, та іншими, обумовленими певними вподобаннями користувачів, а також для спрощення перегляду необхідної інформації про ігри, та для спрощення обговорення гри.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

- перегляд списку усіх ігор;
- перегляд списку усіх жанрів;
- перегляд списку усіх видавців;
- перегляд списку усіх ігрових платформ;
- перегляд списку усіх новин;
- перегляд детального опису гри;
- перегляд детального опису жанру;
- перегляд детального опису видавця;
- перегляд детального опису ігрової платформи;
- перегляд детального опису новину;
- пошук ігор за жанрами;
- пошук ігор за роком;
- пошук ігор за видавцем;
- пошук ігор за ігровими платформами;
- реєстрація облікового запису.

4.1.1.2 Для зареєстрованого користувача:

- функції, що належать звичайному користувачеві;
- вхід в обліковий запис;
- коментування ігор;
- оцінювання ігор.

4.1.1.3 Для адміністратора системи:

– всі функції, що належать зареєстрованому користувачеві;
– додавання та видалення гри ;
– додавання та видалення новини;
– додавання та видалення жанру;
– додавання та видалення новини;
– додавання та видалення видавця;
– додавання та видалення ігрової платформи.

4.1.2 Розробку виконати на платформі Windows.

4.1.3 Додаткові вимоги:

– мова клієнтського інтерфейсу – українська.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

Всі запроси, що передбачають зміну даних в системі, мають бути доступні лише для автентифікованих користувачів.

4.2.2 Передбачити захист від некоректних дій користувача.

Під час надходження запитів на оновлення та видалення певних даних, система має виконати перевірку того, що поточний користувач є автором цих даних, а інакше повернути повідомлення про помилку.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються

4.3.2 Обслуговування

Програмне забезпечення не вимагає специфічного обслуговування

4.3.3 Обслуговуючий персонал

Є необхідність в людині (адміністраторі), що створить базу з іграми та новинами, і час від часу буде її оновлювати.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору ..... Intel i3.

4.4.2.2 Об‘єм ОЗП..... 3 ГБ.

4.4.2.2 Підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем сімейств Windows 7, 8 та 10.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: HTTP запити до сервера з JSON значеннями у тілі запиту.

4.5.3 Результати повинні бути представлені в наступному форматі: Відповіді на HTTP запити у форматі JSON.

4.5.4 Програмне забезпечення повинно обробляти запити, надіслані лише з визначених адресів.

4.5.5 Програмне забезпечення повинно взаємодіяти з сервером за допомогою протоколу HTTPS.

4.5.6 Клієнтська частина повинна бути розроблена на мовах програмування TypeScript, HTML, SCSS, та за допомогою фреймворку Angular, серверна частина має бути створена з використанням мови програмування C#, та технологій ASP.Net Core, Entity Framework Core.

4.6. Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Розгорнути серверну та клієнтську частини на одному або окремих серверах.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм верхнього рівня повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 90 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2 Технічне завдання.

5.3.3 Опис програми.

5.3.4 Керівництво користувача.

5.4 Графічна частина повинна бути виконана на 3 аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки.

5.4.1 Схема структурна варіантів використання.

5.4.2 Схема бази даних.

5.4.3 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02.2020	
2.	Розробка технічного завдання	03.03.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.03.2020	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.03.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	05.04.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.04.2020	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.04.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	20.04.2020	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.04.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до підрозділу «Випробування програмного продукту».

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ «ІГРОВА БІБЛІОТЕКА»

Керівництво користувача

КПІ.ІП-6127.045440.03.34

“ПОГОДЖЕНО”

Керівник проєкту:

Ю.М.Крамар

Виконавець:

А.О.Шатровський

Нормоконтроль:

К.І. Ліщук

Київ – 2020 року

КЕРІВНИЦТВО КОРИСТУВАЧА

Загальний опис інтерфейсу

Основні елементи сторінки:

- навігаційна панель (рисунок 1 п.1);
- контент сторінки (рисунок 1 п.2).

Основний вміст сторінки завжди буде відображений в елементі Контент сторінки, також в певних частинах застосування користувач бачитиме певну інформацію в модальних вікнах.

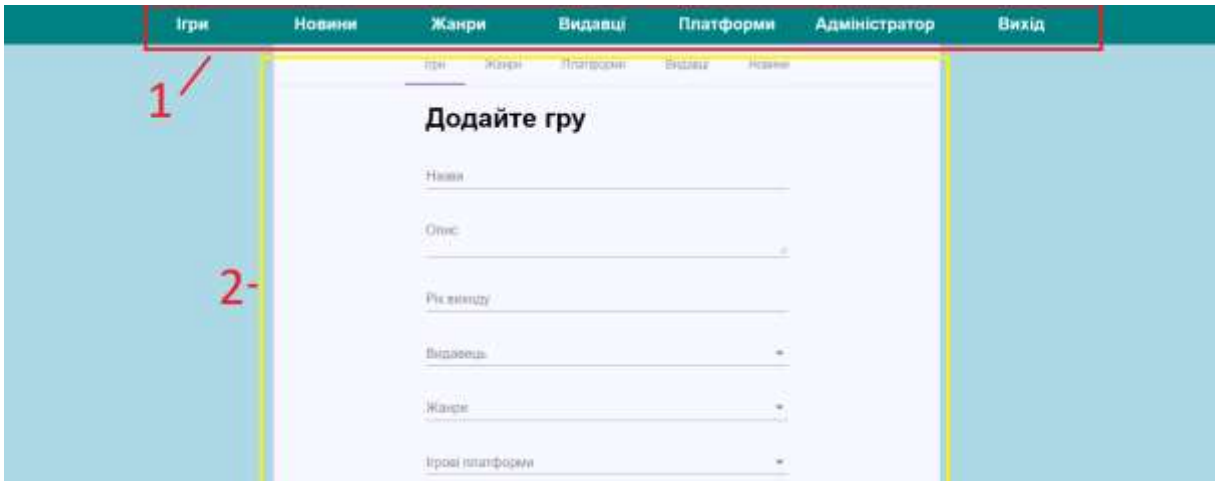


Рисунок 1 - Інтерфейс веб-застосування «Кулінарний помічник»

Навігаційна панель

Навігаційна панель використовується для переходу між сторінками веб-застосування, та для виходу користувача з облікового запису.

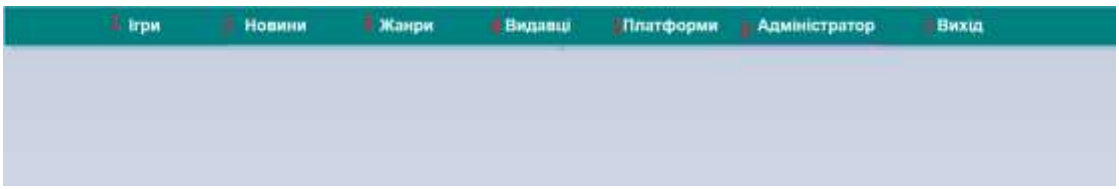


Рисунок 2 - Навігаційна панель авторизованого користувача

Рисунок 3 - Навігаційна панель неавторизованого користувача

Основні елементи навігаційної панелі (рисунки 2, 3):

- ігри;
- новини;

- жанри;
- видавці;
- платформи;
- адміністратор;
- авторизація;
- реєстрація;
- вихід.

При натисканні на певну вкладку відбувається перехід на сторінку з списком відповідних об’єктів. Також елемент навігації вибраної сторінки підсвічується.

Наприклад, при натисканні на елемент “Ігри” відбудеться перехід на сторінку з списком всіх ігор.

Особливою є кнопка Вихід – при натисканні на неї користувач вийде з облікового запису, і залишиться на тій самій сторінці, вона доступна лише для авторизованих користувачів.

Неавторизовані користувачі бачитимуть елементи “Авторизація” та “Реєстрація”, при натисканні на які вони перейдуть на відповідні сторінки.

Сторінка Реєстрації. Загальний опис

Коли новий користувач захоче створити обліковий запис, перш за все необхідно перейти на сторінку реєстрації. Зробити це можна двома шляхами:

- натиснувши кнопку Реєстрація в навігаційній панелі;
- через пряме посилання.

Опис інтерфейсу сторінки реєстрації (рисунок 4):

- поле для вводу імені користувача – нікнейму (рисунок 4 п.1);
- поле для вводу електронної пошти (рисунок 4 п.2);
- поле для вводу паролю (рисунок 4 п.3);
- кнопка реєстрації.

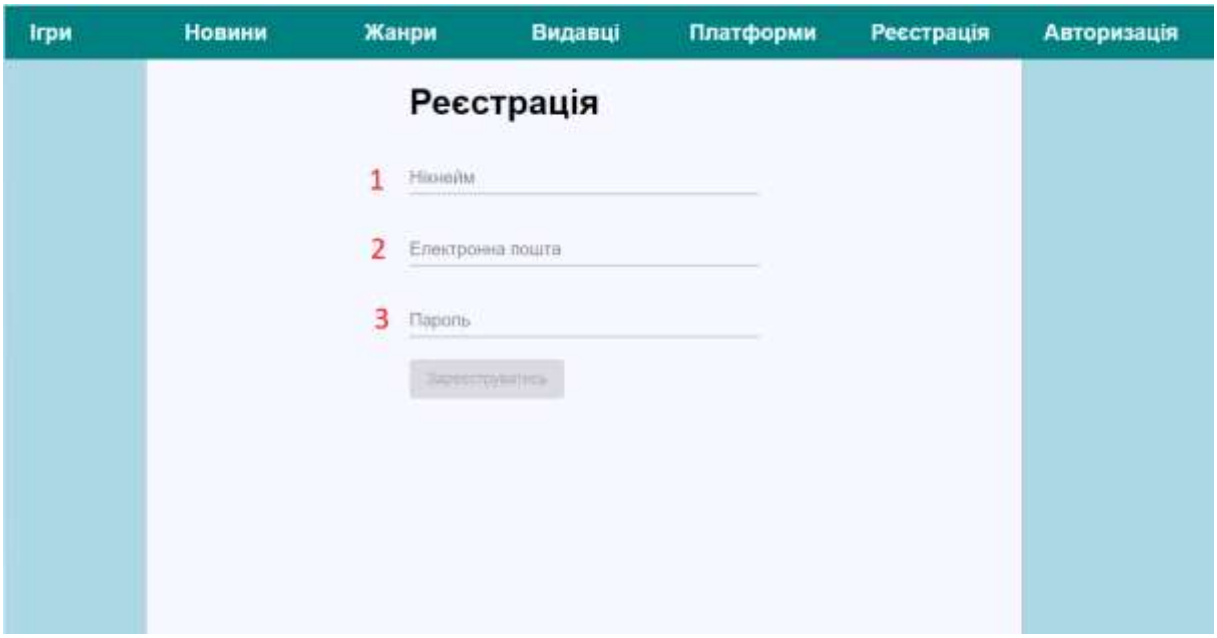


Рисунок 4 - Інтерфейс сторінки «Реєстрація»

Поля Нікнейму та паролю є обов'язковими. Також існують певні обмеження на значення для них. Мінімальна довжина імені користувача – 5 символів, мінімальна довжина паролю – 6 символів.

Ім'я користувача має бути унікальним, інакше реєстрація не пройде успішно.

Поле електронної пошти є необов'язковим.

Коли користувач заповнить коректні дані, стане доступна кнопка Зареєструватись, при натисканні на яку користувач буде зареєстрований, якщо не виникне ніяких помилок.

Сторінка Авторизації. Загальний опис

Для входу в свій обліковий запис користувачу потрібно перйти на сторінку авторизації. Зробити це можна двома шляхами:

- натиснувши кнопку Авторизація в навігаційній панелі;
- через пряме посилання.

Опис інтерфейсу сторінки реєстрації (рисунок 5):

- поле для вводу імені користувача – нікнейму (рисунок 5 п.1);
- поле для вводу паролю (рисунок 5 п.2);
- кнопка авторизації.

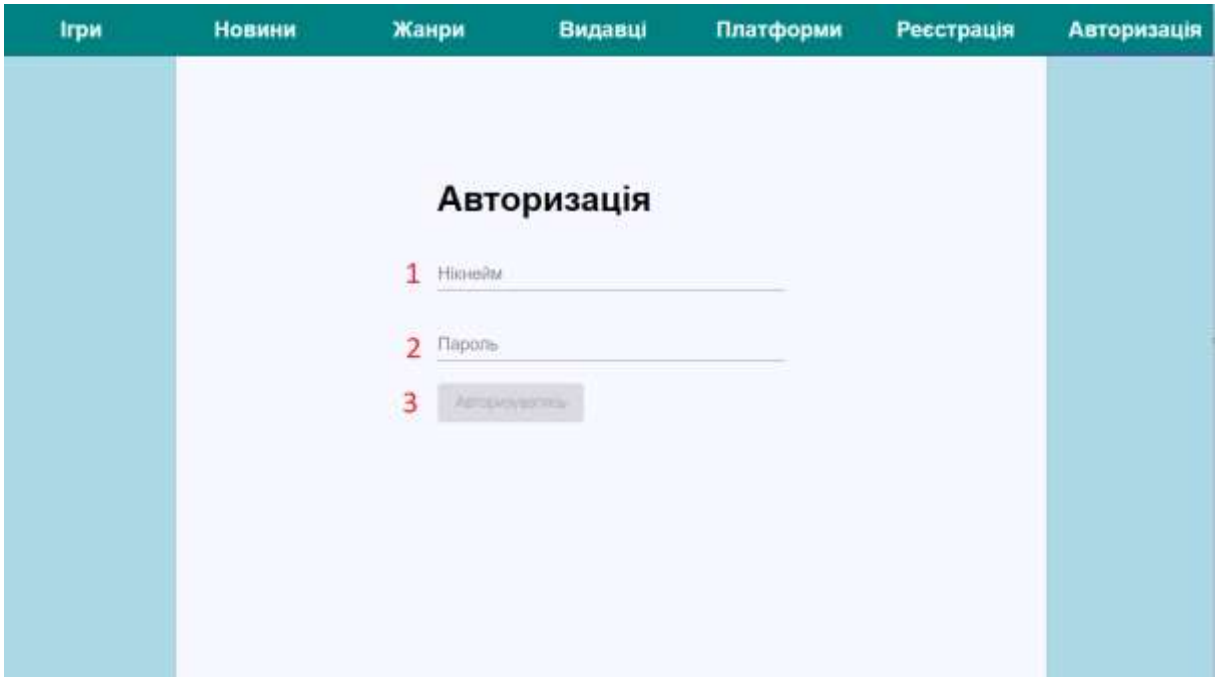


Рисунок 5 - Інтерфейс сторінки «Авторизація»

Коли користувач заповнить дані, стане доступна кнопка авторизації, при натисканні на яку буде надісланий запит на сервер, і в випадку успішної авторизації в відповіді буде повернений токен доступу, який буде зберігатись в локальному сховищі браузера. Цей токен буде дійсний один день, після чого користувачу потрібно буде пройти авторизацію повторно.

Сторінка “Адміністратор”. Загальний опис

На сторінці адміністратора наявні 5 вкладок для створення різного наповнення веб-застосування (рисунок 6).

Список елементів секції :

- ігри;
- жанри;
- платформи;
- видавці;
- новини.

При натисканні на елемент з цього списку, буде відкрита відповідна сторінка для створення сутності. Наприклад, при натисканні на “Ігри” відкриється сторінка для додавання гри.

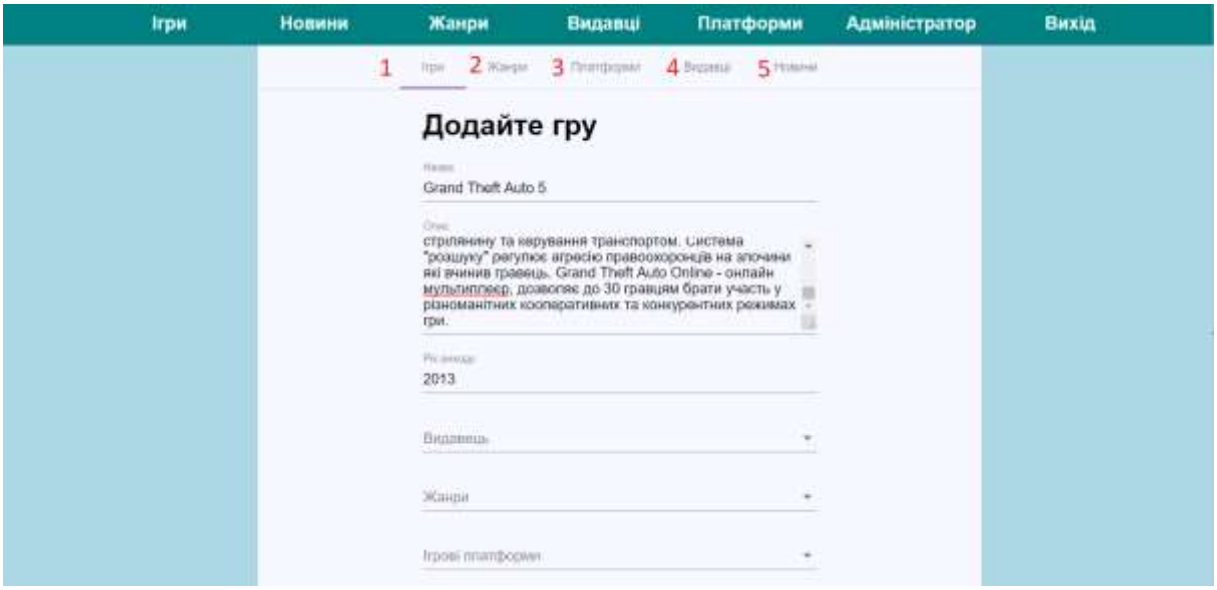


Рисунок 6 - Секція «Адміністратор»

Сторінка Адміністратора. Секція «Ігри»

Список елементів секції (рисунок 7):

- назва гри;
- опис;
- рік виходу;
- видавець;
- жанри;
- ігрові платформи;
- завантаження зображень;
- кнопка додавання гри.

Усі поля крім Назви гри не є обов’язковими для заповнення.

У полі Видавця можна вибрати одного з вже існуючих видавців в системі.

У полях жанрів і ігрових платформ можна вибрати будь-яку кількість відповідних існуючих елементів.

Рисунок 7 - Секція «Ігри»

Рисунок 8 - Вибір видавця

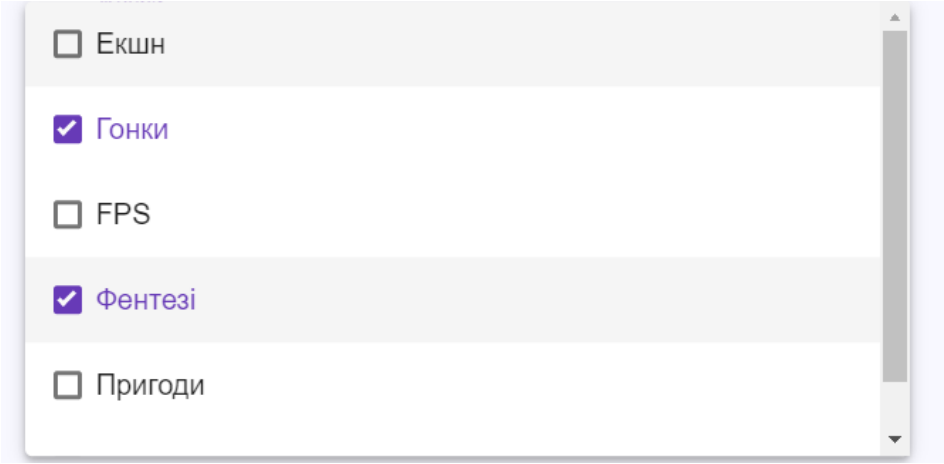


Рисунок 9 - Вибір жанрів

Сторінка Адміністратора. Секція «Жанри»

Список елементів секції (рисунок 10):

- назва жанру;
- опис;
- кнопка додавання жанру.

Лише поле назви є обов’язковими для заповнення.

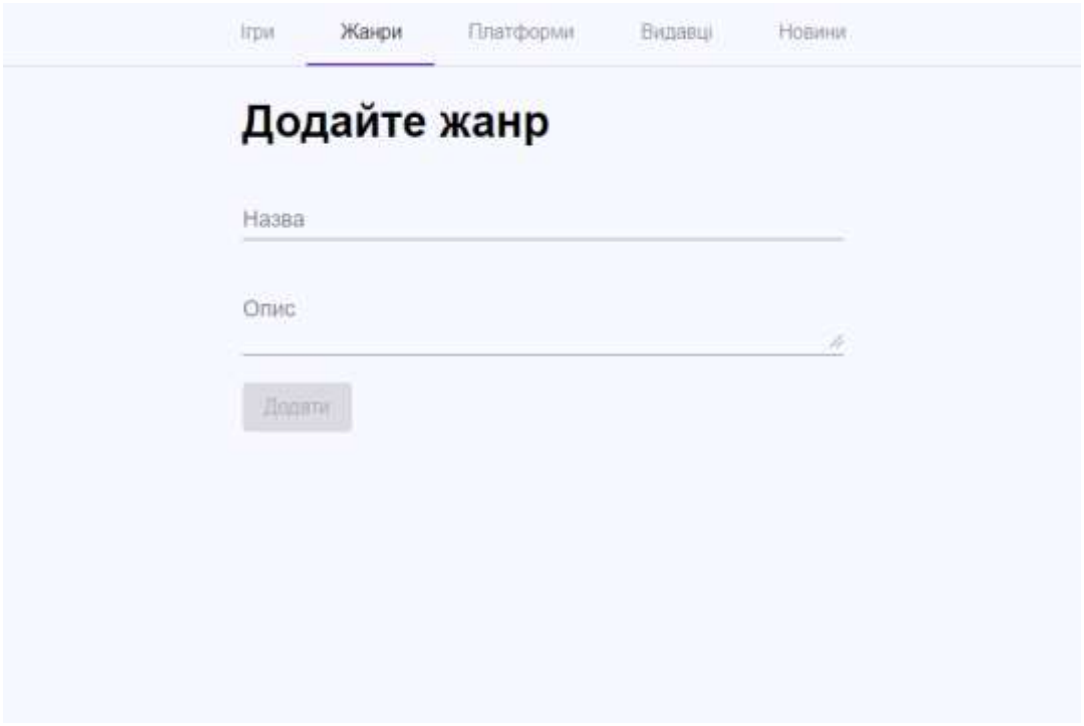


Рисунок 10 - Секція «Жанри»

Сторінка Адміністратора. Секція «Платформи»

Список елементів секції (рисунок 11):

- назва ігрової платформи;
- опис;
- кнопка додавання ігрової платформи.

Лише поле назви є обов’язковим для заповнення.

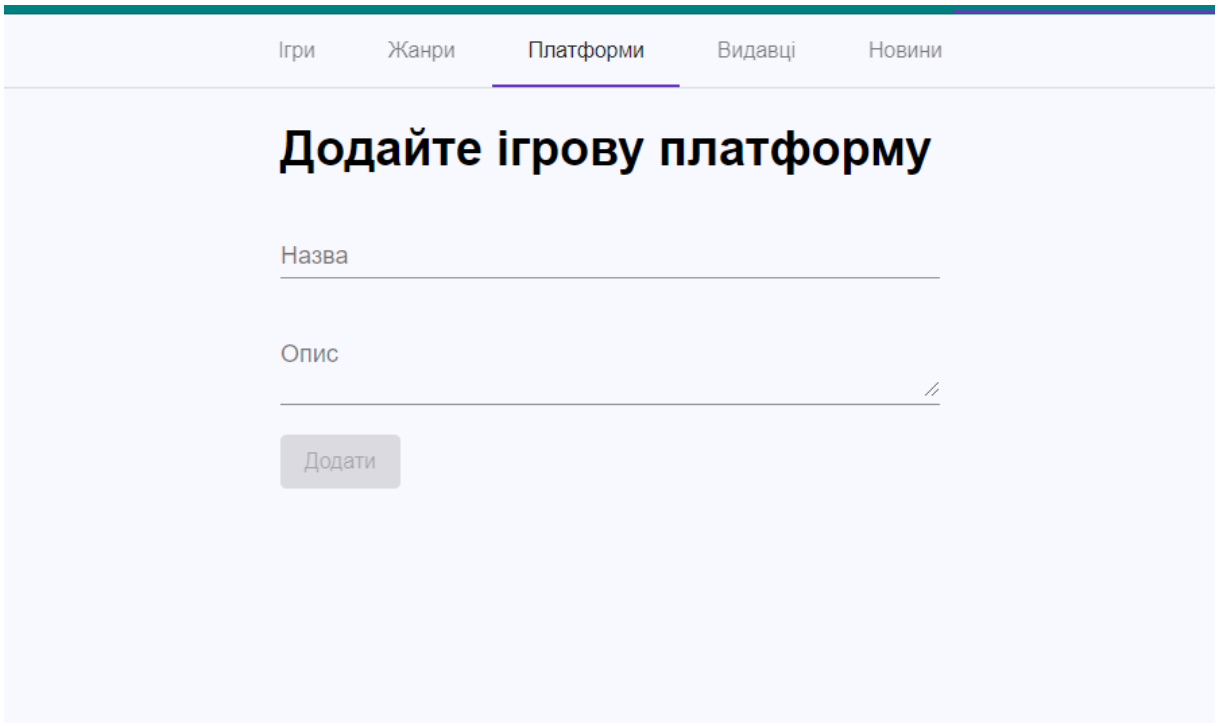


Рисунок 11 - Секція «Платформи»

Сторінка Адміністратора. Секція «Видавці»

Список елементів секції (рисунок 12):

- назва видавця;
- опис;
- кнопка додавання видавця.

Лише поле назви є обов’язковим для заповнення.

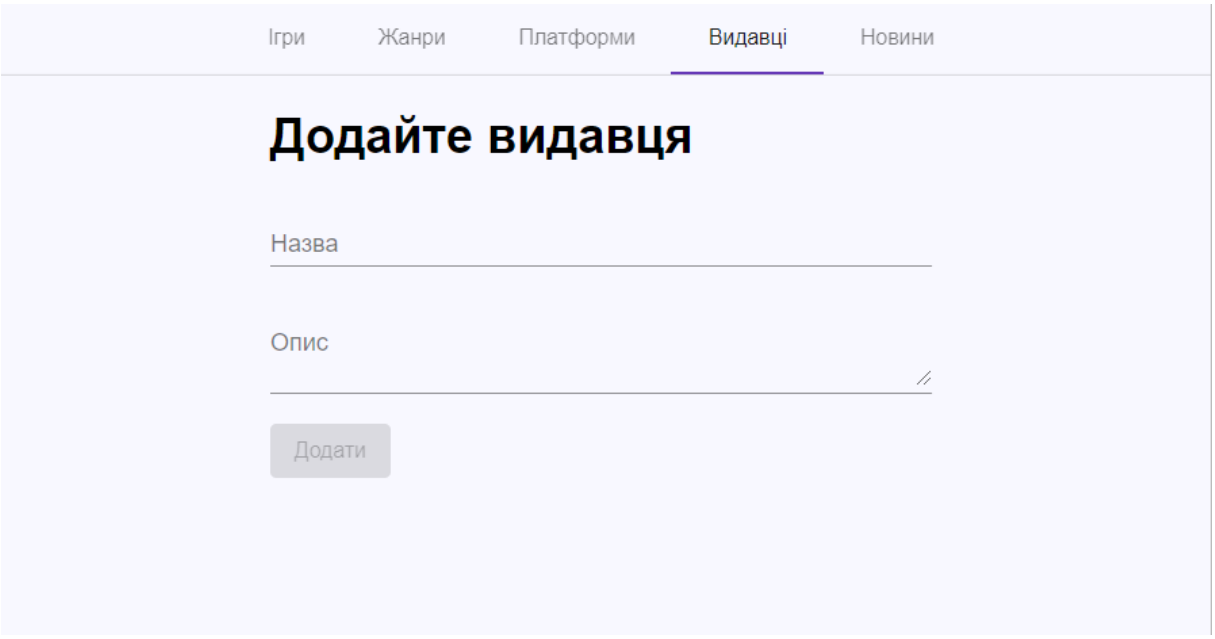


Рисунок 12 - Секція «Видавці»

Сторінка Адміністратора. Секція «Новини»

Список елементів секції (рисунок 13):

- назва новини;
- опис;
- пов’язані ігри;
- кнопка додавання новини.

Лише поле назви є обов’язковим для заповнення.

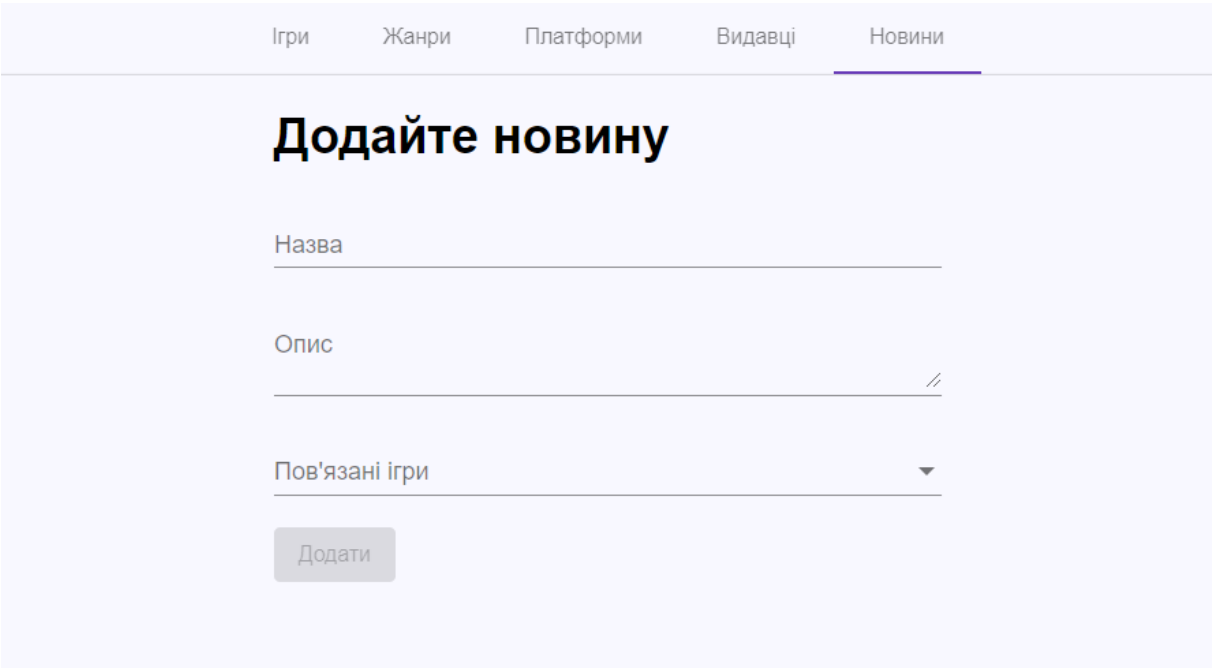


Рисунок 13 - Секція «Новини»

Сторінка ігор.

На даній сторінці представлений список всіх ігор які були додані в базу даних (рисунок 14). Вона складається з багатьох частин, таких як відображення власне списку ігор, так і частини з пошуком ігор за вподобаннями користувача.

Список елементів секції:

- розширений пошук;
- фільтрація за назвою;
- список ігрових елементів.

Рисунок 14 - Сторінка “Ігри”

При натисканні на галочку розширеного пошуку буде відкрите вікно з гнучким налаштуванням пошуку користувачем (рисунок 15).

Список елементів секції розширеного пошуку:

- мінімальний рік випуску гри;
- максимальний рік випуску гри;
- видавець;

- жанри;
- ігрові платформи;
- виключені видавці;
- виключені жанри.

Якщо якесь поле залишиться пустим, сервер не буде фільтрувати інформацію по ньому. Наприклад, якщо залишити пустим список Жанрів, в результаті при фільтрації жанри ніяк не будуть враховуватись. Аналогічна ситуація з іншими полями.

В всіх полях можна вибрати декілька варіантів одночасно, наприклад якщо в полі Жанри будуть вибрані значення Фентезі та Пригоди, в результаті будуть відображенні тільки ті ігри, які містять в собі обидва цих жанри одночасно.

При виборі якогось жанру в полі “Виключити жанри”, він стає недоступним в полі “Жанри” (рисунок 16).

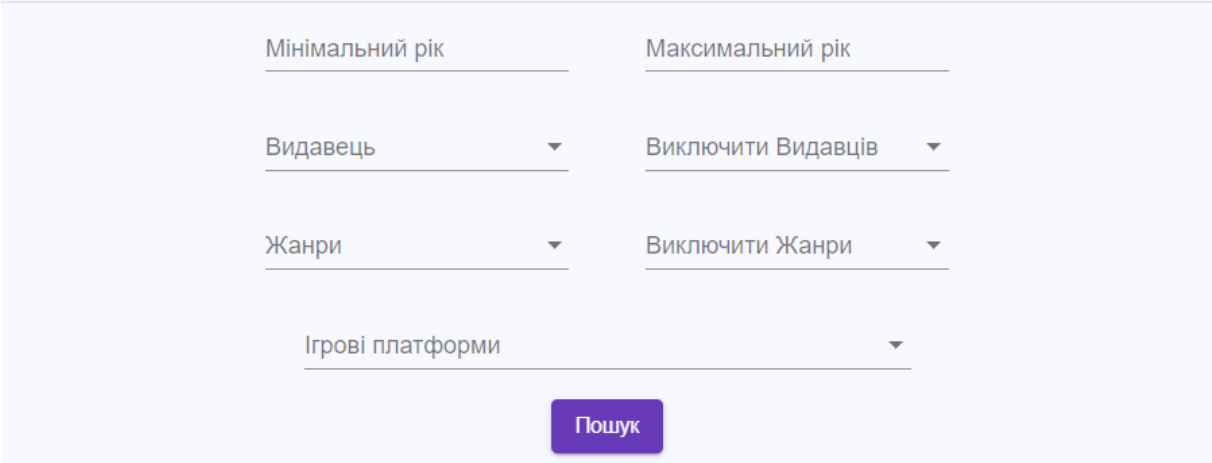


Рисунок 15 - Секція “Розширений пошук”

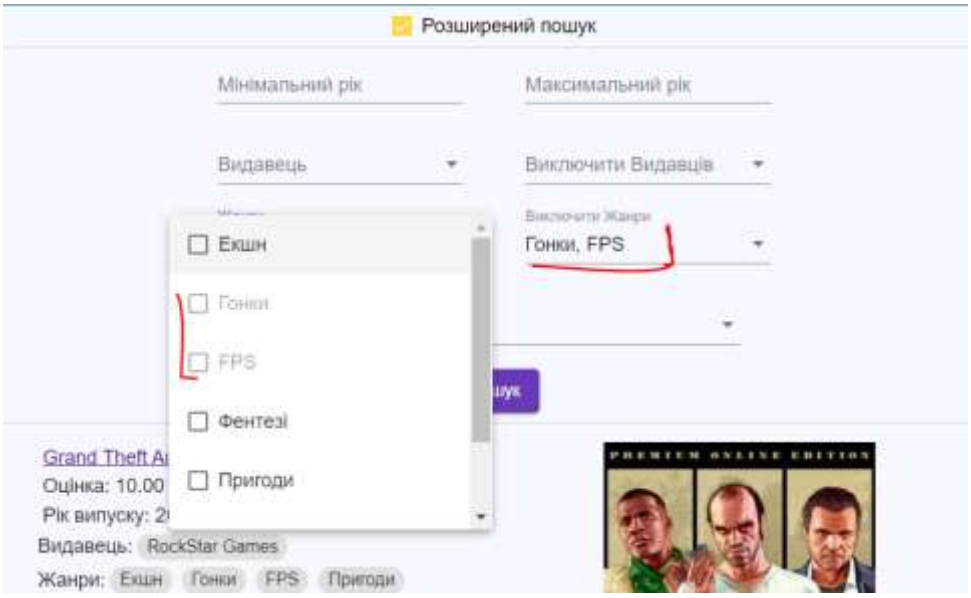


Рисунок 16 - Секція “Розширений пошук”, поле вибору жанрів

На цій сторінці всі ігри представлені в вигляді елементів списку.

Кожен елемент цього списку містить такі поля:

- назва (при натисканні на неї користувач перейде на сторінку цієї гри);
- оцінка;
- рік випуску;
- видавець (при натисканні на назву видавця користувач перейде на його сторінку);
- жанри (при натисканні на жанр користувач перейде на його сторінку);
- ігрові платформи (при натисканні на назву ігрової платформи користувач перейде на його сторінку);
- обкладинка гри.

Рисунок 17 - Секція “Ігровий елемент”

Якщо адміністратор не заповнив певну частину інформації про гру, вона буде відсутня в цьому списку.

Сторінка жанрів.

Сторінка призначена для перегляду списку усіх доступних жанрів, а також навігації на конкретний знайдений жанр (рисунок 18).

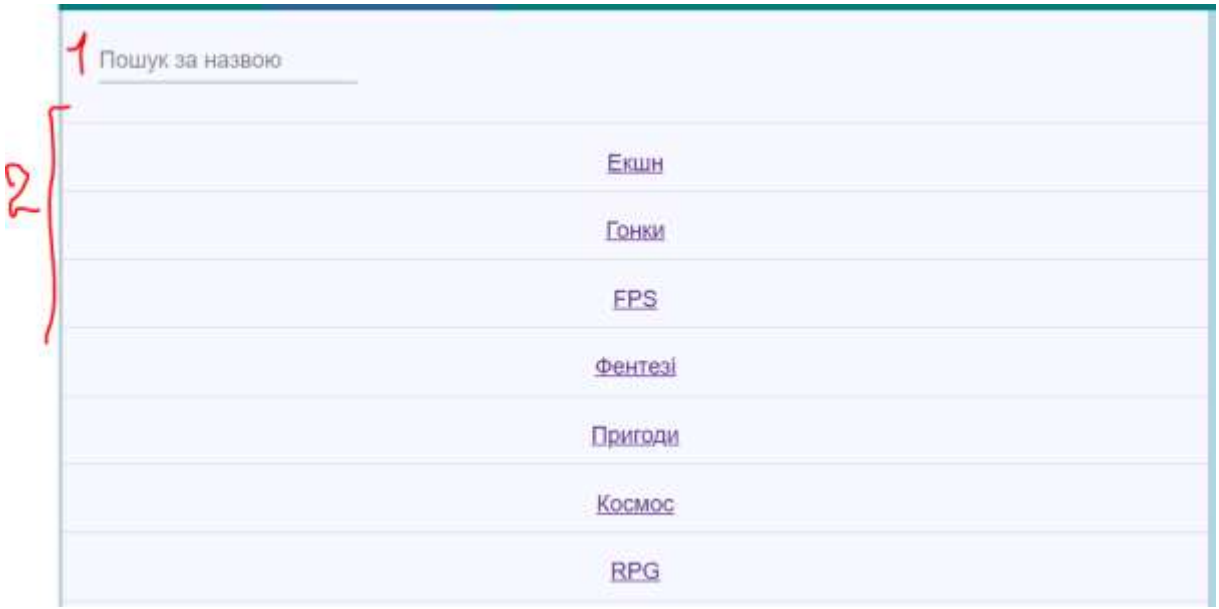


Рисунок 18 - Сторінка жанрів

Вона поділяється на наступні секції:

- пошук жанру за назвою;
- список жанрів.

При натисканні на назву певного жанру, користувач перейде на сторінку з детальною інформацією про відповідний елемент.

Сторінка видавців.

Сторінка призначена для перегляду списку усіх доступних видавців, а також навігації на конкретного знайденого видавця (рисунок 19).

Рисунок 19 - Сторінка видавців

Вона поділяється на наступні секції:

- пошук видавців за назвою;
- список видавців.

При натисканні на назву певного видавця, користувач перейде на сторінку з детальною інформацією про відповідний елемент.

Сторінка ігрових платформ.

Сторінка призначена для перегляду списку усіх доступних ігрових платформ, а також навігації на конкретну знайдену ігрову платформу (рисунок 20).

Рисунок 20 - Сторінка ігрових платформ

Вона поділяється на наступні секції:

- пошук ігрової платформи за назвою;
- список ігрових платформ.

При натисканні на назву певної ігрової платформи, користувач перейде на сторінку з детальною інформацією про відповідний елемент.

Сторінка гри.

Сторінка призначена для перегляду детальної інформації про гру, її оцінювання, а також обговорення користувачами.

Список елементів секції:

- режим редагування (рисунок 21 п.1) – доступний лише для адміністратора, при натисканні на цю кнопку з’явиться сторінка редагування гри, яка виглядає так само як і сторінка додавання гри;
- оцінка гри (рисунок 21 п.2) – зліва вказана оцінка користувача, посередині оцінка інших користувачів, і справа присутня кнопка “Оцінити”, яка відкриє модальне вікно для виставлення оцінки. Кнопка доступна лише для авторизованих користувачів;
- рік випуску (рисунок 21 п.3)
- жанри (рисунок 21 п.4);
- видавець (рисунок 21 п.5);
- ігрові платформи (рисунок 21 п.6);
- опис (рисунок 21 п.7);
- секція коментування (рисунок 21 п.8) – доступна тільки для авторизованих користувачів;
- список коментарів (рисунок 21 п.9).

Рисунок 21 - Сторінка гри

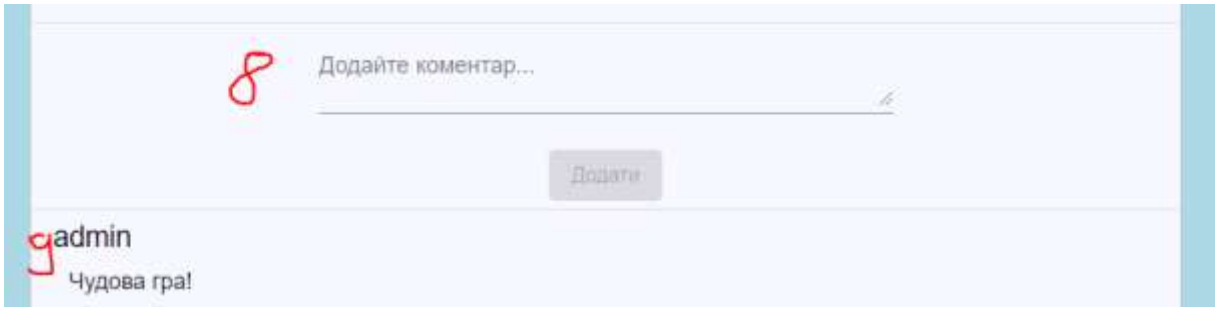


Рисунок 22 - Секція коментарів

При натисканні на назву жанру, видавця, чи ігрової платформи, буде відкрита сторінка відповідного елементу.

Сторінка жанру

Сторінка призначена для преегляду детальної інформації про жанр (рисунок 23).

Сторінка містить наступні елементи:

- режим редагування (рисунок 23 п.1) – доступний лише для адміністратора;

- назва (рисунок 23 п.2);
- опис (рисунок 23 п.3);
- пов’язані ігри (рисунок 23 п.4).

При натисканні на назву будь-якої пов’язаної гри, користувач перейде на сторінку з інформацією про цю гру

Рисунок 23 - Сторінка жанру

Сторінка видавця

Сторінка призначена для преегляду детальної інформації про видавця (рисунок 24).

Сторінка містить наступні елементи:

- режим редагування (рисунок 23 п.1) – доступний лише для адміністратора;
- назва (рисунок 23 п.2);
- опис (рисунок 23 п.3);
- пов’язані ігри (рисунок 23 п.4).

При натисканні на назву будь-якої пов’язаної гри, користувач перейде на сторінку з інформацією про цю гру

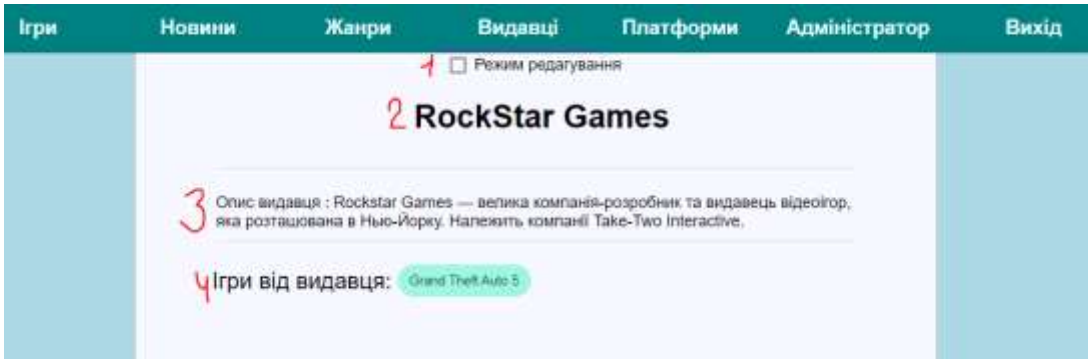


Рисунок 24 - Сторінка видавця

Сторінка ігрової платформи

Сторінка призначена для преегляду детальної інформації про ігрову платформу (рисунок 25).

Сторінка містить наступні елементи:

- режим редагування (рисунок 25 п.1) – доступний лише для адміністратора;
- назва (рисунок 25 п.2);
- опис (рисунок 25 п.3);
- пов’язані ігри (рисунок 25 п.4).

При натисканні на назву будь-якої пов’язаної гри, користувач перейде на сторінку з інформацією про цю гру

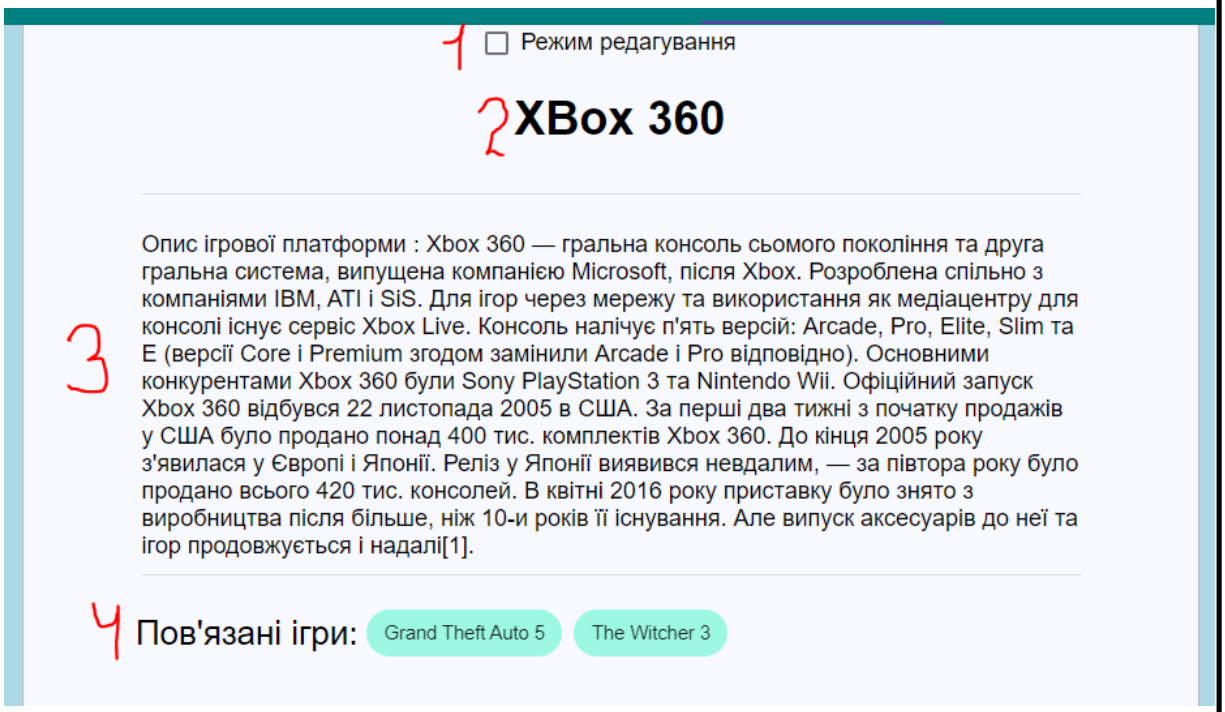


Рисунок 25 - Сторінка ігрової платформи

Сторінка гри. Секція оцінювання

Усі авторизовані користувачі можуть ставити оцінки будь-яким іграм. Для цього необхідно натиснути кнопку “Оцінити”

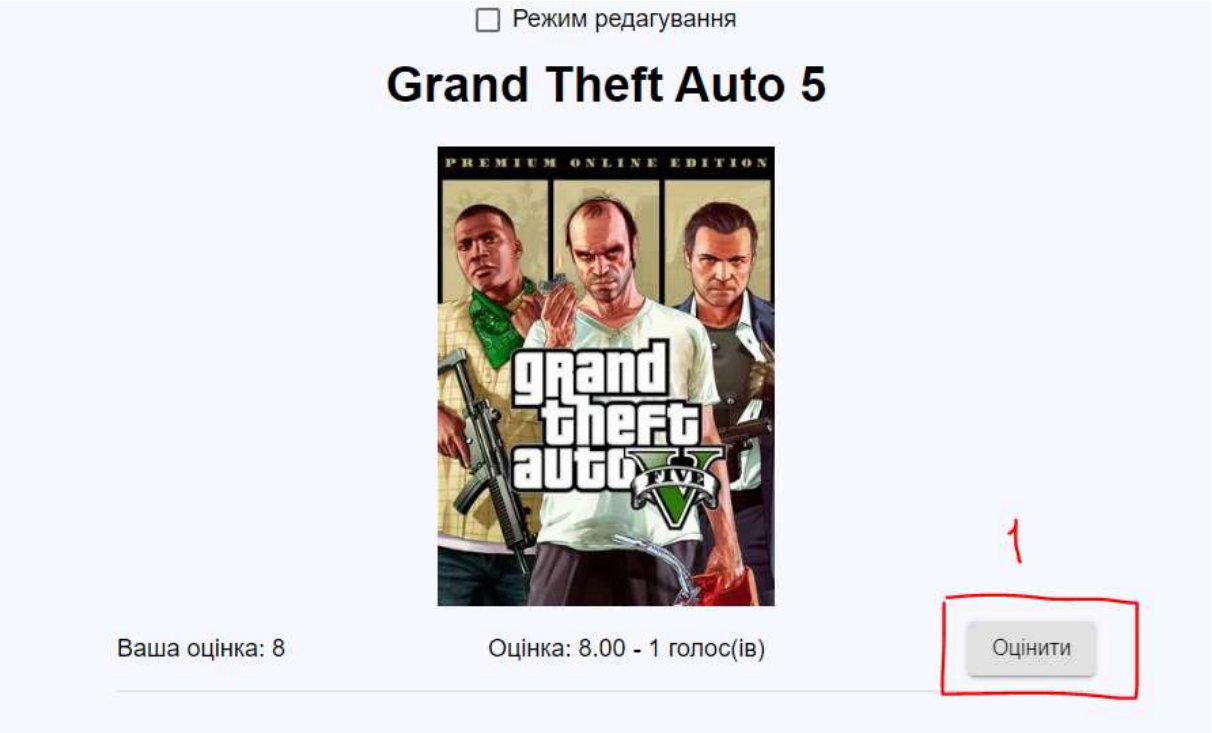


Рисунок 26 - Сторінка гри. Кнопка оцінювання

При натисканні на неї з’явиться модальне вікно (рисунок 27), в якому користувач зможе виставити оцінку від 1 до 10.

При повторному оцінювані стара оцінка перепишеться.

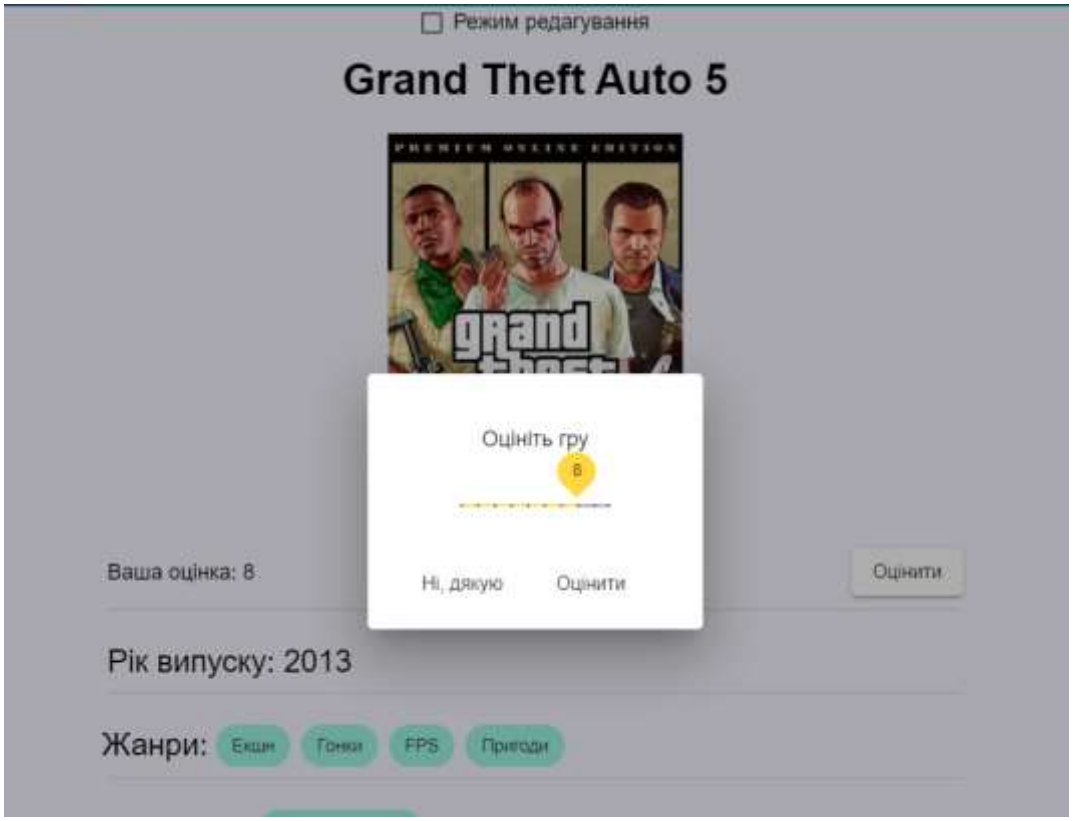


Рисунок 27 - Вікно оцінювання гри

По центру секції оцінювання буде вказана середня оцінка користувачів та кількість проголосувавших (рисунок 28).

Рисунок 28 - Сторінка гри – середня оцінка

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ «ІГРОВА БІБЛІОТЕКА»

Опис програми

КПІ.ІП-6127.045440.04.13

“ПОГОДЖЕНО”
Керівник проєкту:
Ю.М. Крамар

Нормоконтроль:
К.І. Ліщук

Виконавець:
А.О. Шатровський

Київ – 2020 року

Тексти програмного коду

Веб-застосування «Ігрова бібліотека»

(Найменування програми (документа))

DVD-R

(Вид носія даних)

39 арк, 210 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2020

СЕРВЕРНА ЧАСТИНА

```
    [Route("api/[controller]")]
    [ApiController]
    public class ApplicationUserController : ControllerBase
    {
        private IApplicationUserService _applicationUserService;
        private ApplicationSettings _applicationSettings;

        public ApplicationUserController(IApplicationUserService
applicationUserService, IOptions<ApplicationSettings> applicationSettings)
        {
            _applicationUserService = applicationUserService;
            _applicationSettings = applicationSettings.Value;
        }

        // POST: api/ApplicationUser/Register
        [HttpPost]
        [Route("Register")]
        public async Task<object> PostApplicationUser([FromBody] ApplicationUserModel
model)
        {
            model.Role = "Customer";
            try
            {
                var result = await _applicationUserService.RegisterUserAsync(model);
                if (result.Succeeded)
                {
                    return Ok(result);
                }
                else
                {
                    return BadRequest(result.Errors);
                }
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        // POST: api/ApplicationUser/Register
        [HttpPost]
        [Route("RegisterAdmin")]
        public async Task<object> PostApplicationUserAdmin([FromBody]
ApplicationUserModel model)
        {
            model.Role = "Admin";
            try
            {
                var result = await _applicationUserService.RegisterUserAsync(model);
                if (result.Succeeded)
                {
                    return Ok(result);
                }
                else
                {
                    return BadRequest(result.Errors);
                }
            }
            catch (Exception ex)
            {
                throw ex;
            }
```

```
        }

        // POST: api/ApplicationUser/Login
        [HttpPost]
        [Route("Login")]
        public async Task<IActionResult> Login([FromBody] LoginModel model)
        {
            var result = await _applicationUserService.SignInAsync(model,
_applicationSettings.JWT_Secret);

            if (result.Succeeded)
            {
                return Ok( new { token = result.Token });
            }
            else
            {
                return BadRequest(result.Message);
            }
        }
    }
[Route("api/comments")]
    [ApiController]
    public class CommentsController : ControllerBase
    {
        ICommentService commentService;
        public CommentsController(ICommentService commentService)
        {
            this.commentService = commentService;
        }

        // POST: api/comments/
        [HttpPost("games")]
        [Authorize()]
        public ActionResult AddCommentToGame([FromBody] CommentModel commentModel)
        {
            string userId = User.Claims.First(c => c.Type == "UserID").Value;
            commentModel.ApplicationUserId = userId;
            commentService.AddComment(commentModel);
            return Ok();
        }

        // POST: api/comments/{id}/comments
        [HttpPost("{id}/comments")]
        public ActionResult AddReplyToCommentById(int id, [FromBody] CommentModel
commentModel)
        {
            var comment = commentService.GetComment(id);
            if (comment == null)
            {
                return NotFound();
            }
            commentModel.ParentCommentId = id;
            commentService.AddComment(commentModel);
            return Ok();
        }

        //GET api/comments/{id}
        [HttpGet]
        public ActionResult<CommentModel> GetAllComments()
        {
            var comment = commentService.GetAllComments();
            if (comment == null)
            {
                return NotFound();
```

```
            }
            return Ok(comment);
        }

        //GET api/comments/{id}
        [HttpGet("{id}")]
        public ActionResult<CommentModel> GetComment(int id)
        {
            var comment = commentService.GetComment(id);
            if (comment == null)
            {
                return NotFound();
            }
            return Ok(comment);
        }
    }

[Route("api/games")]
    [ApiController]
    public class GamesController : ControllerBase
    {
        IGameService gameService;
        IGenreService genreService;
        IPlatformTypeService platformTypeService;
        ICommentService commentService;
        IApplicationUserService applicationUserService;

        public GamesController(IGameService gameService, IGenreService genreService,
IPlatformTypeService platformTypeService, ICommentService commentService,
                IApplicationUserService applicationUserService)
        {
            this.gameService = gameService;
            this.genreService = genreService;
            this.platformTypeService = platformTypeService;
            this.commentService = commentService;
            this.applicationUserService = applicationUserService;
        }

        [HttpPost("search")]
        public IEnumerable<GameModel> SearchGamesByModel(SearchGameModel
searchGameModel)
        {
            var res = gameService.SearchGameByManyParameters(searchGameModel);
            return res;
            //if(res.Any())
            //{
            //    return Ok(res);
            //}
        }

        // GET: api/Games
        [HttpGet]
        public IEnumerable<GameModel> GetAllGames()
        {
            return gameService.GetAllGames();
        }

        // GET: api/Games/5
        [HttpGet("{id}")]
        public ActionResult<GameModel> GetGameById(int id)
        {
            var game = gameService.GetGame(id);
            if (game == null)
            {
```

```
            return NotFound();
        }
        return Ok(game);
    }

    // POST: api/Games
    [HttpPost]
    [Authorize(Roles ="Admin,Customer")]
    public ActionResult AddOrUpdateGame([FromBody] GameModel gameModel)
    {
        gameService.AddOrUpdateGame(gameModel);
        return Ok();
    }

    // PUT: api/Games/5
    [HttpPut("{id}")]
    public ActionResult EditGame(int id, [FromBody] GameModel gameModel)
    {
        var game = gameService.GetGame(id);
        if (game == null)
        {
            return NotFound();
        }
        gameModel.Id = id;
        gameService.EditGame(gameModel);
        return Ok();
    }

    // DELETE: api/ApiWithActions/5
    [HttpDelete("{id}")]
    public ActionResult DeleteGame(int id)
    {
        var game = gameService.GetGame(id);
        if (game == null)
        {
            return NotFound();
        }
        gameService.DeleteGame(id);
        return Ok();
    }

    // GET: api/games/5/comments
    [HttpGet("{id}/comments")]
    public ActionResult<IEnumerable<CommentModel>> GetComments(int id)
    {
        var comments = commentService.GetCommentsByGameId(id);
        if (comments == null)
        {
            return NotFound();
        }
        return Ok(comments);
    }

    //GET : api/games/{id}/genres
    [HttpGet("{id}/genres")]
    public ActionResult<IEnumerable<GenreModel>> GetGenresByGameId(int id)
    {
        var game = gameService.GetGame(id);
        if (game == null)
        {
            return NotFound();
        }
        var genres = genreService.GetGenresByGameId(id);
        return Ok(genres);
```

```
        }

        //PUT : api/games/{gameid}/genres/{genreid}
        [HttpPut("{gameid}/genres/{genreid}")]
        public ActionResult AddGenreToGameById(int gameid, int genreid)
        {
            var game = gameService.GetGame(gameid);
            var genre = genreService.GetGenre(genreid);
            if (game == null || genre == null)
            {
                return NotFound();
            }
            gameService.AddGenreToGame(gameid, genreid);
            return Ok();
        }

        //PUT : api/games/{gameid}/platformtypes/{platformtypeid}
        [HttpPut("{gameid}/platformtypes/{platformtypeid}")]
        public ActionResult AddPlatformTypeToGameById(int gameid, int platformtypeid)
        {
            var game = gameService.GetGame(gameid);
            var pl = platformTypeService.GetPlatformType(platformtypeid);
            if (game == null || pl == null)
            {
                return NotFound();
            }
            gameService.AddPlatformToGame(gameid, platformtypeid);
            return Ok();
        }

        //GET : api/games/{id}/platformtypes
        [HttpGet("{id}/platformtypes")]
        public ActionResult<IEnumerable<PlatformTypeModel>> GetPlatformsByGameId(int
id)
        {
            var game = gameService.GetGame(id);
            if (game == null)
            {
                return NotFound();
            }
            var platforms = platformTypeService.GetPlatformTypesByGameId(id);
            return Ok(platforms);
        }

        // POST: api/games/{id}/comments
        [HttpPost("{id}/comments")]
        [Authorize()]
        public ActionResult AddCommentToGame(int id, [FromBody] InputCommentModel
inputCommentModel)
        {
            string userId = User.Claims.First(c => c.Type == "UserID").Value;
            //var appUser =
this.applicationUserService.GetApplicationUser(userId).Result;
            //var inputCommentModel = new InputCommentModel();
            var commentModel = new CommentModel()
            {
                Body = inputCommentModel.Body,
                ApplicationUserId = userId,
                GameId = id
            };
            commentService.AddComment(commentModel);
            return Ok(commentService.GetCommentsByGameId(id));
        }
```

```
            // POST: api/games/{id}/marks/{mark}
            [HttpPost("{id}/marks")]
            [Authorize()]
            public ActionResult AddMarkToGame(int id, [FromBody] int mark)
            {
                string userId = User.Claims.First(c => c.Type == "UserID").Value;
                var markModel = new MarkModel()
                {
                    ApplicationUserId = userId,
                    GameId = id,
                    Value = mark
                };
                var res = gameService.AddMarkToGame(markModel);
                return Ok(res);
            }

            // Get: api/games/{id}/marks
            [HttpGet("{id}/marks")]
            public ActionResult GetUserMark(int id)
            {
                //var userIdClaim = User.Claims.FirstOrDefault(c => c.Type == "UserID");
                //if(userIdClaim != nu)
                string userId = User.Claims.FirstOrDefault(c => c.Type ==
    "UserID")?.Value;
                //var appUser =
    this.applicationUserService.GetApplicationUser(userId).Result;
                var res = gameService.GetMarksInfoModel(id, userId);
                return Ok(res);
            }
        }
    [Route("api/genres")]
        [ApiController]
        public class GenresController : ControllerBase
        {
            IGenreService genreService;
            IGameService gameService;

            public GenresController(IGenreService genreService, IGameService gameService)
            {
                this.genreService = genreService;
                this.gameService = gameService;
            }

            // GET: api/Genres
            [HttpGet]
            public IEnumerable<GenreModel> GetAllGenres()
            {
                return genreService.GetAllGenres();
            }

            // GET: api/Genres/5
            [HttpGet("{id}")]
            public ActionResult<GenreModel> GetGenreById(int id)
            {
                var genre = genreService.GetGenre(id);
                if (genre == null)
                {
                    return NotFound();
                }
                return Ok(genre);
            }

            // POST: api/Genres
            [HttpPost]
```

```
        public ActionResult AddGenre([FromBody] GenreModel genreModel)
        {
            genreService.AddGenre(genreModel);
            return Ok();
        }

        // PUT: api/Genres/5
        [HttpPut("{id}")]
        public ActionResult EditGenre(int id, [FromBody] GenreModel genreModel)
        {
            var Genre = genreService.GetGenre(id);
            if (Genre == null)
            {
                return NotFound();
            }
            genreModel.Id = id;
            genreService.EditGenre(genreModel);
            return Ok();
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult DeleteGenre(int id)
        {
            var genre = genreService.GetGenre(id);
            if (genre == null)
            {
                return NotFound();
            }
            genreService.DeleteGenre(id);
            return Ok();
        }

        //GET : api/genres/{id}/games
        [HttpGet("{id}/games")]
        public ActionResult<IEnumerable<GameModel>> GetGamesByGenreId(int id)
        {
            var genre = genreService.GetGenre(id);
            if (genre == null)
            {
                return NotFound();
            }
            var games = gameService.GetGamesByGenreId(id);
            return Ok(games);
        }
    }
[Route("api/news")]
    [ApiController]
    public class NewsController : ControllerBase
    {
        INewsService newsService;
        IGameService gameService;

        public NewsController(INewsService newsService, IGameService gameService)
        {
            this.newsService = newsService;
            this.gameService = gameService;
        }

        // GET: api/News
        [HttpGet]
        public IEnumerable<NewsModel> GetAllNews()
        {
            return newsService.GetAllNews();
```

```
        }

        // GET: api/News/5
        [HttpGet("{id}")]
        public ActionResult<NewsModel> GetNewsById(int id)
        {
            var news = newsService.GetNews(id);
            if (news == null)
            {
                return NotFound();
            }
            return Ok(news);
        }

        // POST: api/News
        [HttpPost]
        public ActionResult AddNews([FromBody] NewsModel newsModel)
        {
            newsService.AddNews(newsModel);
            return Ok();
        }

        // PUT: api/News/5
        [HttpPut("{id}")]
        public ActionResult EditNews(int id, [FromBody] NewsModel newsModel)
        {
            var News = newsService.GetNews(id);
            if (News == null)
            {
                return NotFound();
            }
            newsModel.Id = id;
            newsService.EditNews(newsModel);
            return Ok();
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public ActionResult DeleteNews(int id)
        {
            var news = newsService.GetNews(id);
            if (news == null)
            {
                return NotFound();
            }
            newsService.DeleteNews(id);
            return Ok();
        }
[Route("api/platformTypes")]
    [ApiController]
    public class PlatformTypesController : ControllerBase
    {
        IGameService gameService;
        IGenreService genreService;
        IPlatformTypeService platformTypeService;
        ICommentService commentService;

        public PlatformTypesController(IGameService gameService, IGenreService
genreService, IPlatformTypeService platformTypeService, ICommentService
commentService)
        {
            this.gameService = gameService;
            this.genreService = genreService;
            this.platformTypeService = platformTypeService;
```

```
            this.commentService = commentService;
        }

        // GET: api/PlatformTypes
        [HttpGet]
        public IEnumerable<PlatformTypeModel> GetAllPlatformTypes()
        {
            return platformTypeService.GetAllPlatformTypes();
        }

        // GET: api/PlatformTypes/5
        [HttpGet("{id}")]
        public ActionResult<PlatformTypeModel> GetPlatformTypeById(int id)
        {
            var game = platformTypeService.GetPlatformType(id);
            if (game == null)
            {
                return NotFound();
            }
            return Ok(game);
        }

        // POST: api/PlatformTypes
        [HttpPost]
        public ActionResult AddPlatformType([FromBody] PlatformTypeModel gameModel)
        {
            platformTypeService.AddPlatformType(gameModel);
            return Ok();
        }

        // PUT: api/PlatformTypes/5
        [HttpPut("{id}")]
        public ActionResult EditPlatformType(int id, [FromBody] PlatformTypeModel
gameModel)
        {
            var game = platformTypeService.GetPlatformType(id);
            if (game == null)
            {
                return NotFound();
            }
            gameModel.Id = id;
            platformTypeService.EditPlatformType(gameModel);
            return Ok();
        }

        [HttpDelete("{id}")]
        public ActionResult DeletePlatformType(int id)
        {
            var game = platformTypeService.GetPlatformType(id);
            if (game == null)
            {
                return NotFound();
            }
            platformTypeService.DeletePlatformType(id);
            return Ok();
        }

        //GET : api/PlatformTypes/{id}/games
        [HttpGet("{id}/games")]
        public ActionResult<IEnumerable<GameModel>> GetGamesByPlatformTypeId(int id)
        {
            var platform = platformTypeService.GetPlatformType(id);
            if (platform == null)
            {
```

```
                return NotFound();
            }
            var games = gameService.GetGamesByPlatformId(id);
            return Ok(games);
        }

    }
[Route("api/publishers")]
    [ApiController]
    public class PublishersController : ControllerBase
    {
        IPublisherService publisherService;
        IGameService gameService;

        public PublishersController(IPublisherService publisherService, IGameService
gameService)
        {
            this.publisherService = publisherService;
            this.gameService = gameService;
        }

        // GET: api/publishers
        [HttpGet]
        public IEnumerable<PublisherModel> GetAllPublishers()
        {
            return publisherService.GetAllPublishers();
        }

        // GET: api/publishers/5
        [HttpGet("{id}")]
        public ActionResult<PublisherModel> GetPublisherById(int id)
        {
            var publisher = publisherService.GetPublisher(id);
            if (publisher == null)
            {
                return NotFound();
            }
            return Ok(publisher);
        }

        // POST: api/publishers
        [HttpPost]
        public ActionResult AddPublisher([FromBody] PublisherModel publisherModel)
        {
            publisherService.AddPublisher(publisherModel);
            return Ok();
        }

        // PUT: api/publishers/5
        [HttpPut("{id}")]
        public ActionResult EditPublisher(int id, [FromBody] PublisherModel
publisherModel)
        {
            var publisher = publisherService.GetPublisher(id);
            if (publisher == null)
            {
                return NotFound();
            }
            publisherModel.Id = id;
            publisherService.EditPublisher(publisherModel);
            return Ok();
        }

        // DELETE: api/ApiWithActions/5
```

```
        [HttpDelete("{id}")]
        public ActionResult DeletePublisher(int id)
        {
            var publisher = publisherService.GetPublisher(id);
            if (publisher == null)
            {
                return NotFound();
            }
            publisherService.DeletePublisher(id);
            return Ok();
        }

        //GET : api/publishers/{id}/games
        [HttpGet("{id}/games")]
        public ActionResult<IEnumerable<GameModel>> GetGamesByPublisherId(int id)
        {
            var publisher = publisherService.GetPublisher(id);
            if (publisher == null)
            {
                return NotFound();
            }
            var games = gameService.GetGamesByPublisherId(id);
            return Ok(games);
        }
    }

public static class WebApiDependencies
    {
        public static void AddWebApiDependencies(this IServiceCollection services)
        {
            services.AddScoped<IPlatformTypeService, PlatformTypeService>();
            services.AddScoped<IPublisherService, PublisherService>();
            services.AddScoped<ICommentService, CommentService>();
            services.AddScoped<IGenreService, GenreService>();
            services.AddScoped<IGameService, GameService>();
            services.AddScoped<IApplicationUserService, ApplicationUserService>();
            services.AddScoped<INewsService, NewsService>();

            services.Configure<IdentityOptions>(options =>
            {
                options.Password.RequireNonAlphanumeric = false;
                options.Password.RequireDigit = false;
                options.Password.RequiredLength = 6;
                options.Password.RequireUppercase = false;
            });
        }
    }
public class GameService : IGameService
    {
        IUnitOfWork db;
        IMapper mapper;

        public GameService(IUnitOfWork db, IMapper mapper)
        {
            this.db = db;
            this.mapper = mapper;
        }

        public IEnumerable<GameModel> SearchGameByManyParameters(SearchGameModel
searchGameModel)
        {
            var games = db.Games.FindAll(game =>
            {
                var res = true;
```

```
                if (searchGameModel.Genres.Count != 0)
                {
                    foreach (var searchEntity in searchGameModel.Genres)
                    {
                        if (!CheckGenre(game.GamesGenres, searchEntity.Id))
                        {
                            res = false;
                            return false;
                        }
                    }
                    foreach (var searchEntity in searchGameModel.ExcludeGenres)
                    {
                        if (CheckGenre(game.GamesGenres, searchEntity.Id))
                        {
                            res = false;
                            return false;
                        }
                    }
                    if (res == false)
                    {
                        return false;
                    }
                }

                if (searchGameModel.PlatformTypes.Count != 0)
                {
                    foreach (var searchEntity in searchGameModel.PlatformTypes)
                    {
                        if (!CheckPlatforms(game.GamesPlatforms, searchEntity.Id))
                        {
                            res = false;
                            return false;
                        }
                    }
                    if (res == false)
                    {
                        return false;
                    }
                }

                if (searchGameModel.Publisher.Id != 0 && game.PublisherId !=
searchGameModel.Publisher.Id)
                {
                    return false;
                }

                if( searchGameModel.ExcludePublishers.Count != 0)
                {
                    foreach (var searchEntity in searchGameModel.ExcludePublishers)
                    {
                        if (CheckPublisher(game.Publisher, searchEntity.Id))
                        {
                            res = false;
                            return false;
                        }
                    }
                    if (res == false)
                    {
                        return false;
                    }
                }
```

```
                if (!CheckYear(game.Year, searchGameModel.MinYear,
searchGameModel.MaxYear))
                {
                    return false;
                }

                return true;
            });
            return mapper.Map<IEnumerable<GameModel>>(games);
        }

        private bool CheckGenre(ICollection<GameGenre> ggs, int id)
        {
            return ggs.Any(gg => gg.GenreId == id);
        }

        private bool CheckPlatforms(ICollection<GamePlatform> gps, int id)
        {
            return gps.Any(gp => gp.PlatformTypeId == id);
        }

        private bool CheckPublisher(Publisher pub, int id)
        {
            return pub.Id == id;
        }

        private bool CheckYear(int gameYear, int? minYear, int? maxYear)
        {
            if (gameYear == 0)
            {
                return true;
            }
            int min = minYear != 0 ? minYear.Value : int.MinValue;
            int max = maxYear != 0 ? maxYear.Value : int.MaxValue;
            if ( max < min )
            {
                var temp = min;
                min = max;
                max = temp;
            }
            return gameYear >= min && gameYear <= max;
        }

        public void AddOrUpdateGame(GameModel gameModel)
        {
            var game = mapper.Map<Game>(gameModel);

            if(game.Publisher.Id == 0)
            {
                game.Publisher = null;
                game.PublisherId = null;
            }

            if(db.Games.GetByIdAsNoTracking(game.Id) == null)
            {
                db.Games.Add(game);
            }
            else
            {
                db.Games.Update(game);
            }

            db.Commit();
        }
```

```
        public void AddGenreToGame(int gameId, int genreId)
        {
            var game = db.Games.GetById(gameId);
            game.GamesGenres.Add(new GameGenre
            {
                GameId = gameId,
                GenreId = genreId
            });
            db.Commit();
        }

        public void AddPlatformToGame(int gameId, int platformId)
        {
            var game = db.Games.GetById(gameId);
            game.GamesPlatforms.Add(new GamePlatform
            {
                GameId = gameId,
                PlatformTypeId = platformId
            });
            db.Commit();
        }

        public void DeleteGame(int id)
        {
            db.Games.Delete(id);
            db.Commit();
        }

        public void DeleteGame(GameModel gameModel)
        {
            var game = mapper.Map<Game>(gameModel);
            db.Games.Delete(game);
            db.Commit();
        }

        public void EditGame(GameModel gameModel)
        {
            var game = mapper.Map<Game>(gameModel);
            db.Games.Update(game);
            db.Commit();
        }

        public IEnumerable<GameModel> GetAllGames()
        {
            var games = db.Games.GetAll();
            return mapper.Map<IEnumerable<GameModel>>(games);
        }

        public GameModel GetGame(int id)
        {
            var game = GetDbGame(id);
            return mapper.Map<GameModel>(game);
        }

        private Game GetDbGame(int id)
        {
            return db.Games.GetById(id);
        }

        public IEnumerable<GameModel> GetGamesByGenreId(int id)
        {
            var games = db.Games.FindAll(g => g.GamesGenres.Any(gg => gg.GenreId ==
id));
```

```
            return mapper.Map<IEnumerable<GameModel>>(games);
        }

        public IEnumerable<GameModel> GetGamesByPlatformId(int id)
        {
            var games = db.Games.FindAll(g => g.GamesPlatforms.Any(gpl =>
gpl.PlatformTypeId == id));
            return mapper.Map<IEnumerable<GameModel>>(games);
        }

        public IEnumerable<GameModel> GetGamesByPublisherId(int id)
        {
            var publisher = db.Publishers.GetById(id);
            return mapper.Map<IEnumerable<GameModel>>(publisher.Games);
        }

        public MarksInfoModel AddMarkToGame(MarkModel markModel)
        {
            var mark = mapper.Map<Mark>(markModel);
            var game = GetDbGame(markModel.GameId);
            mark.Game = game;

            var existingMark = db.Marks.GetByIds(markModel.GameId,
markModel.ApplicationUserId);

            if (existingMark != null)
            {
                db.Marks.Update(mark);
            }
            else
            {
                db.Marks.Add(mark);
            }

            db.Commit();
            return GetMarksInfoModel(markModel.GameId, markModel.ApplicationUserId);
        }

        public MarksInfoModel GetMarksInfoModel(int gameId, string userId = "")
        {
            var marks = db.Marks.FindAll(mark => mark.GameId == gameId);
            double averageMark = 0;
            int numberOfVotes = 0;

            foreach(var mark in marks)
            {
                numberOfVotes++;
                averageMark += mark.Value;
            }
            var userMark = marks.FirstOrDefault(mark => mark.ApplicationUserId ==
userId)?.Value;
            return new MarksInfoModel()
            {
                AverageMark = averageMark / numberOfVotes,
                NumberOfVotes = numberOfVotes,
                UserMark = userMark
            };
        }

        public int GetUserMark(int id, MarkModel markModel)
        {
            var res = db.Marks.GetByIds(id, markModel.ApplicationUserId);
            if (res != null)
            {
```

```
                return res.Value;
            }
            return 0;
        }
    }
public class GenreService : IGenreService
    {
        IUnitOfWork db;
        IMapper mapper;

        public GenreService(IUnitOfWork db, IMapper mapper)
        {
            this.db = db;
            this.mapper = mapper;
        }

        public void AddGenre(GenreModel genreModel)
        {
            var genre = mapper.Map<Genre>(genreModel);
            db.Genres.Add(genre);
            db.Commit();
        }

        public void DeleteGenre(int id)
        {
            db.Genres.Delete(id);
            db.Commit();
        }

        public void DeleteGenre(GenreModel genreModel)
        {
            var genre = mapper.Map<Genre>(genreModel);
            db.Genres.Delete(mapper.Map<Genre>(genre));
            db.Commit();
        }

        public void EditGenre(GenreModel genreModel)
        {
            var genre = mapper.Map<Genre>(genreModel);
            db.Genres.Update(genre);
            db.Commit();
        }

        public IEnumerable<GenreModel> GetAllGenres()
        {
            var genres = db.Genres.GetAll();
            return mapper.Map<IEnumerable<GenreModel>>(genres);
        }

        public GenreModel GetGenre(int id)
        {
            var genre = db.Genres.GetById(id);
            return mapper.Map<GenreModel>(genre);
        }

        public IEnumerable<GenreModel> GetGenresByGameId(int id)
        {
            var genres = db.Genres.FindAll(g => g.GamesGenres.Any(gg => gg.GameId ==
id));
            return mapper.Map<IEnumerable<GenreModel>>(genres);
        }
    }
public interface IRepository <T> where T : class
    {
```

```
        IEnumerable<T> GetAll();

        T GetById(int id);

        T GetByIdAsNoTracking(int id);

        IEnumerable<T> FindAll(Func<T,bool> where);

        void Add(T item);

        void Update(T item);

        void Delete(T item);

        void Delete(int id);
    }

public interface IUnitOfWork
    {
        IRepository<PlatformType> PlatformTypes { get; }
        IRepository<Publisher> Publishers { get; }
        IRepository<Comment> Comments { get; }
        IRepository<Genre> Genres { get; }
        IRepository<Game> Games { get; }
        IRepository<News> News { get; }
        ApplicationUserRepository Users { get; }
        MarkRepository Marks { get; }
        void Commit();
    }
class GameConfiguration : IEntityTypeConfiguration<Game>
    {
        public void Configure(EntityTypeBuilder<Game> builder)
        {
            builder.HasKey(g => g.Id);
            builder.HasOne(g => g.Publisher).WithMany(p => p.Games).HasForeignKey(g =>
g.PublisherId);

            builder.HasIndex(g => g.Name).IsUnique(true);
        }
    }
public class Game
    {
        public Game()
        {
            Comments = new List<Comment>();
            GamesGenres = new List<GameGenre>();
            GamesPlatforms = new List<GamePlatform>();
            Marks = new List<Mark>();
            Images = new List<Image>();
        }

        public int Id { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public bool IsDeleted { get; set; }

        public int Year { get; set; }

        public int? PublisherId { get; set; }

        public virtual Publisher Publisher { get; set; }
```

```
        public virtual ICollection<Image> Images { get; set; }

        public virtual IEnumerable<Comment> Comments { get; set; }

        public virtual ICollection<GameGenre> GamesGenres { get; set; }

        public virtual ICollection<GamePlatform> GamesPlatforms { get; set; }

        public virtual ICollection<GameNews> GamesNews { get; set; }

        public virtual ICollection<Mark> Marks { get; set; }
    }
public class Genre
    {
        public Genre()
        {
            GamesGenres = new List<GameGenre>();
        }

        public int Id { get; set; }

        public string Name { get; set; }

        public bool IsDeleted { get; set; }

        public string Description { get; set; }

        public int? ParentGenreId { get; set; }

        public virtual Genre ParentGenre { get; set; }

        public virtual ICollection<GameGenre> GamesGenres { get; set; }

    }

public class PlatformType
    {
        public PlatformType()
        {
            GamesPlatforms = new List<GamePlatform>();
        }

        public int Id { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public bool IsDeleted { get; set; }

        public virtual ICollection<GamePlatform> GamesPlatforms { get; set; }
    }

public class NewsService : INewsService
    {
        IUnitOfWork db;
        IMapper mapper;

        public NewsService(IUnitOfWork db, IMapper mapper)
        {
            this.db = db;
            this.mapper = mapper;
        }
```

```
        public void AddNews(NewsModel genreModel)
        {
            var genre = mapper.Map<News>(genreModel);
            db.News.Add(genre);
            db.Commit();
        }

        public void DeleteNews(int id)
        {
            db.News.Delete(id);
            db.Commit();
        }

        public void DeleteNews(NewsModel genreModel)
        {
            var genre = mapper.Map<News>(genreModel);
            db.News.Delete(mapper.Map<News>(genre));
            db.Commit();
        }

        public void EditNews(NewsModel genreModel)
        {
            var genre = mapper.Map<News>(genreModel);
            db.News.Update(genre);
            db.Commit();
        }

        public IEnumerable<NewsModel> GetAllNews()
        {
            var news = db.News.GetAll();
            return mapper.Map<IEnumerable<NewsModel>>(news);
        }

        public NewsModel GetNews(int id)
        {
            var genre = db.News.GetById(id);
            return mapper.Map<NewsModel>(genre);
        }

        public IEnumerable<NewsModel> GetNewsByGameId(int id)
        {
            var news = db.News.FindAll(g => g.GamesNews.Any(gg => gg.GameId == id));
            return mapper.Map<IEnumerable<NewsModel>>(news);
        }
    }
public class GenreService : IGenreService
    {
        IUnitOfWork db;
        IMapper mapper;

        public GenreService(IUnitOfWork db, IMapper mapper)
        {
            this.db = db;
            this.mapper = mapper;
        }

        public void AddGenre(GenreModel genreModel)
        {
            var genre = mapper.Map<Genre>(genreModel);
            db.Genres.Add(genre);
            db.Commit();
        }
```

```
        public void DeleteGenre(int id)
        {
            db.Genres.Delete(id);
            db.Commit();
        }

        public void DeleteGenre(GenreModel genreModel)
        {
            var genre = mapper.Map<Genre>(genreModel);
            db.Genres.Delete(mapper.Map<Genre>(genre));
            db.Commit();
        }

        public void EditGenre(GenreModel genreModel)
        {
            var genre = mapper.Map<Genre>(genreModel);
            db.Genres.Update(genre);
            db.Commit();
        }

        public IEnumerable<GenreModel> GetAllGenres()
        {
            var genres = db.Genres.GetAll();
            return mapper.Map<IEnumerable<GenreModel>>(genres);
        }

        public GenreModel GetGenre(int id)
        {
            var genre = db.Genres.GetById(id);
            return mapper.Map<GenreModel>(genre);
        }

        public IEnumerable<GenreModel> GetGenresByGameId(int id)
        {
            var genres = db.Genres.FindAll(g => g.GamesGenres.Any(gg => gg.GameId ==
id));
            return mapper.Map<IEnumerable<GenreModel>>(genres);
        }
    }
public class PlatformTypeService : IPlatformTypeService
    {
        IUnitOfWork db;
        IMapper mapper;

        public PlatformTypeService(IUnitOfWork db, IMapper mapper)
        {
            this.db = db;
            this.mapper = mapper;
        }

        public void AddPlatformType(PlatformTypeModel platformTypeModel)
        {
            var pl = mapper.Map<PlatformType>(platformTypeModel);
            db.PlatformTypes.Add(pl);
            db.Commit();
        }

        public void DeletePlatformType(int id)
        {
            db.PlatformTypes.Delete(id);
            db.Commit();
        }

        public void DeletePlatformType(PlatformTypeModel platformTypeModel)
```

```
        {
            var pl = mapper.Map<PlatformType>(platformTypeModel);
            db.PlatformTypes.Delete(pl);
            db.Commit();
        }

        public void EditPlatformType(PlatformTypeModel platformTypeModel)
        {
            var pl = mapper.Map<PlatformType>(platformTypeModel);
            db.PlatformTypes.Update(pl);
            db.Commit();
        }

        public IEnumerable<PlatformTypeModel> GetAllPlatformTypes()
        {
            var pl = db.PlatformTypes.GetAll();
            return mapper.Map<IEnumerable<PlatformTypeModel>>(pl);
        }

        public PlatformTypeModel GetPlatformType(int id)
        {
            var pl = db.PlatformTypes.GetById(id);
            return mapper.Map<PlatformTypeModel>(pl);
        }

        public IEnumerable<PlatformTypeModel> GetPlatformTypesByGameId(int id)
        {
            var pls = db.PlatformTypes.FindAll(pl => pl.GamesPlatforms.Any(gpl =>
gpl.GameId == id));
            return mapper.Map<IEnumerable<PlatformTypeModel>>(pls);
        }
    }
public static class BusinessLogicDependencies
    {
        public static void AddBusinessLogicDependencies(this IServiceCollection
services)
        {
            services.AddScoped<IUnitOfWork, UnitOfWork>();
            //services.AddScoped<IMapper, Mapper>();
        }
    }

public class GameModel
    {
        public GameModel()
        {
            Genres = new List<GenreModel>();
            PlatformTypes = new List<PlatformTypeModel>();
            Images = new List<ImageModel>();
        }
        public int Id { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public bool IsDeleted { get; set; }

        public int Year { get; set; }

        public List<ImageModel> Images { get; set; }

        public MarksInfoModel MarkInfo { get; set; }
```

```
        public int NumberOfMarks { get; set; }

        public PublisherModel Publisher { get; set; }

        public List<GenreModel> Genres { get; set; }

        public List<PlatformTypeModel> PlatformTypes { get; set; }
    }

public class GenreModel
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public bool IsDeleted { get; set; }

        public string Description { get; set; }

        public int? ParentGenreId { get; set; }

        public virtual GenreModel ParentGenre { get; set; }
    }

public class ImageModel
    {
        public int Id { get; set; }

        public string Title { get; set; }

        public byte[] Data { get; set; }
    }

public class MarksInfoModel
    {
        public int NumberOfVotes { get; set; }

        public double AverageMark { get; set; }

        public int? UserMark { get; set; }
    }
```

КЛІЄНТСЬКА ЧАСТИНА

```
@Injectable({
  providedIn: 'root'
})
export class CommentsService {

  constructor(private http: HttpClient) { }

  getAllComments(gameId: number) {
    const url = environment.apiBaseUrl +
'games/' + gameId + '/comments';
    return
this.http.get<InputCommentModel[]>(url);
  }

  addCommentToGame(id: number, model:
InputCommentModel) {
    const url = environment.apiBaseUrl +
'games/' + id + '/comments';
    return
this.http.post<InputCommentModel[]>(url,
model);
  }

}
export class GamesService {

  constructor(private http: HttpClient) { }

  getAllGames() {
    const url = environment.apiBaseUrl +
'games';
    return this.http.get<GameModel[]>(url);
  }

  getGenresByGameId(id: number) {
    const url = environment.apiBaseUrl +
'games/' + id + '/genres';
    return this.http.get<GenreModel[]>(url);
  }

  getGameById(id: number) {
    const url = environment.apiBaseUrl +
'games/' + id;
    return this.http.get<GameModel>(url);
  }

  deleteGameById(id: number) {
    const url = environment.apiBaseUrl +
'games/' + id;
    return this.http.delete<GameModel>(url);
  }

  addOrUpdateGame(model: GameModel) {
    const url = environment.apiBaseUrl +
'games';
    return this.http.post(url, model);
  }

  searchGames(model: SearchGameModel):
Observable<GameModel[]> {
    const url = environment.apiBaseUrl +
'games/search';
    return this.http.post<GameModel[]>(url,
model);
  }

  voteByGameId(id: number, mark: number):
Observable<MarkInfoModel> {
    const url = environment.apiBaseUrl +
'games/' + id + '/marks';
    return this.http.post<MarkInfoModel>(url,
mark);
  }

  getMarkInfoByGameId(id: number):
Observable<MarkInfoModel> {
    const url = environment.apiBaseUrl +
'games/' + id + '/marks';
    return this.http.get<MarkInfoModel>(url);
  }
export class GenresService {

  constructor(private http: HttpClient) { }

  getAllGenres() {
    const url = environment.apiBaseUrl +
'Genres';
    return this.http.get<GenreModel[]>(url);
  }

  getGamesByGenreId(id: number) {
    const url = environment.apiBaseUrl +
'Genres/' + id + '/games';
    return this.http.get<GameModel[]>(url);
  }
```

```
  getGenreById(id: number) {
    const url = environment.apiBaseUrl +
'Genres/' + id;
    return this.http.get<GenreModel>(url);
  }

  addGenre(model: GenreModel) {
    const url = environment.apiBaseUrl +
'Genres';
    return this.http.post(url, model);
  }
}

export class NavService {

  constructor(private router: Router) { }

  navigateToPage(page: string, id: number) {
    this.router.navigate([page, id]);
  }
}
export class NewsService {

  constructor(private http: HttpClient) { }

  getAllNews() {
    const url = environment.apiBaseUrl + 'News';
    return this.http.get<NewsModel[]>(url);
  }

  getGamesByNewsId(id: number) {
    const url = environment.apiBaseUrl + 'News/' +
id + '/Games';
    return this.http.get<GameModel[]>(url);
  }

  getNewsById(id: number) {
    const url = environment.apiBaseUrl + 'News/' +
id;
    return this.http.get<NewsModel>(url);
  }

  addNews(model: NewsModel) {
    const url = environment.apiBaseUrl + 'News';
    return this.http.post(url, model);
  }
}
export class PlatformsService {

  constructor(private http: HttpClient) { }

  getAllPlatforms() {
    const url = environment.apiBaseUrl +
'PlatformTypes';
    return
this.http.get<PlatformTypeModel[]>(url);
  }

  getGamesByPlatformId(id: number) {
    const url = environment.apiBaseUrl +
'PlatformTypes/' + id + '/games';
    return this.http.get<GameModel[]>(url);
  }

  getPlatformById(id: number) {
    const url = environment.apiBaseUrl +
'PlatformTypes/' + id;
    return this.http.get<PlatformTypeModel>(url);
  }

  addPlatform(model: PlatformTypeModel) {
    const url = environment.apiBaseUrl +
'PlatformTypes';
    return this.http.post(url, model);
  }
}
export class PublishersService {

  constructor(private http: HttpClient) { }

  getAllPublishers() {
    const url = environment.apiBaseUrl +
'publishers';
    return this.http.get<PublisherModel[]>(url);
  }

  getGamesByPublisherId(id: number) {
    const url = environment.apiBaseUrl +
'publishers/' + id + '/games';
    return this.http.get<GameModel[]>(url);
  }

  getPublisherById(id: number) {
    const url = environment.apiBaseUrl +
'publishers/' + id;
    return this.http.get<PublisherModel>(url);
  }
```

```
  addPublisher(model: PublisherModel) {
    const url = environment.apiBaseUrl +
'publishers';
    return this.http.post(url, model);
  }
}
export class UserService {

  constructor(private http: HttpClient) { }

  register(model: RegistrationModel, asAdmin =
false) {
    let url = environment.apiBaseUrl +
'ApplicationUser/Register';
    url = asAdmin ? url + 'Admin' : url;
    return this.http.post(url, model);
  }

  login(model: LoginModel) {
    const url = environment.apiBaseUrl +
'ApplicationUser/Login';
    return this.http.post(url, model);
  }

  isAuthenticated(): boolean {
    return !!localStorage.getItem('token');
  }

  logOut() {
    localStorage.removeItem('token');
  }

  roleMatch(allowedRoles): boolean {
    let isMatch = false;
    if (!this.isAuthenticated()){
      return false;
    }
    const payload =
JSON.parse(window.atob(localStorage.getItem('to
ken').split('.')[1]));
    const userRole = payload.role;
    allowedRoles.forEach( role => {
      if (userRole === role) {
        isMatch = true;
        return false;
      }
    });
    return isMatch;
  }
}
export class AdminGameComponent implements
OnInit, OnDestroy {

  allGenres: GenreModel[];
  allPlatforms: PlatformTypeModel[];
  allPublishers: PublisherModel[];
  images: ImageModel[] = [];

  @Input() pageName = 'Додайте гру';
  @Input() gameForEdit: GameModel;
  @Output() gameWasUpdated = new
EventEmitter();

  private subscriptions = new Subscription();

  formModel = this.fb.group({
    Name: ['', Validators.required],
    Year: [''],
    Description: [''],
    Genres: [''],
    Publisher: [''],
    Platforms: [''],
  });

  constructor(private fb: FormBuilder, private
gameService: GamesService,
              private genreService: GenresService,
private publisherService: PublishersService,
              private platformService:
PlatformsService) { }

  ngOnInit(): void {

this.subscriptions.add(this.genreService.getAllGe
nres().subscribe( genres => {
      this.allGenres = genres;
    }));

this.subscriptions.add(this.platformService.getAll
Platforms().subscribe( pl => {
      this.allPlatforms = pl;
    }));

this.subscriptions.add(this.publisherService.getAll
Publishers().subscribe( pub => {
      this.allPublishers = pub;
    }));
```

```
    if (this.gameForEdit) {

this.formModel.controls.Name.setValue(this.gam
eForEdit.name || '');

this.formModel.controls.Year.setValue(this.game
ForEdit.year || null);

this.formModel.controls.Description.setValue(this
.gameForEdit.description || '');

this.formModel.controls.Genres.setValue(this.ga
meForEdit.genres || []);

this.formModel.controls.Platforms.setValue(this.g
ameForEdit.platformTypes || []);

this.formModel.controls.Publisher.setValue(this.g
ameForEdit.publisher || {});
      this.formModel.patchValue({
        Genres: this.gameForEdit.genres
      });
    }
  }

  AddOrUpdateGame() {
    const fmv = this.formModel.value;

    const gameModel = {
      id: this.gameForEdit?.id || 0,
      name: fmv.Name,
      year: +fmv.Year,
      description: fmv.Description,
      publisher: fmv.Publisher || {},
      genres: fmv.Genres ? fmv.Genres : [],
      platformTypes: fmv.Platforms ?
fmv.Platforms : [],
      images: this.images
    } as GameModel;



this.gameService.addOrUpdateGame(gameModel
).subscribe(() => {
      this.gameWasUpdated.emit();
    });
    this.formModel.reset();
  }

  ngOnDestroy(): void {
    this.subscriptions.unsubscribe();
  }

}

export class AdminGenreComponent implements
OnInit {

  formModel = this.fb.group({
    Name: ['', Validators.required],
    Description: [''],
  });

  constructor(private fb: FormBuilder, private
genreService: GenresService) { }

  ngOnInit(): void {
  }

  AddGenre() {
    const fmv = this.formModel.value;
    const genre = {
      name: fmv.Name,
      description: fmv.Description,
    } as GenreModel;
    this.genreService.addGenre(genre).subscribe();
  }

}

export class AdminNewsComponent implements
OnInit {

  formModel = this.fb.group({
    Name: ['', Validators.required],
    Body: [''],
    Games: ['']
  });

  allGames: GameModel[];

  constructor(private fb: FormBuilder, private
newsService: NewsService, private gamesService:
GamesService) { }

  ngOnInit(): void {
```

```
this.gamesService.getAllGames().subscribe(game
s => {
     this.allGames = games;
   })
  }

  AddNews() {
   const fmv = this.formModel.value;
   const model = {
    name: fmv.Name,
    body: fmv.Body,
    games: fmv.Games
   } as NewsModel;
   this.newsService.addNews(model).subscribe();
  }

}
export class AdminPlatformComponent
implements OnInit {

  formModel = this.fb.group({
   Name: ['', Validators.required],
   Description: [''],
  });

  constructor(private fb: FormBuilder, private
platformService: PlatformsService) { }

  ngOnInit(): void {
  }

  AddPlatform() {
   const fmv = this.formModel.value;
   const model = {
    name: fmv.Name,
    description: fmv.Description,
   } as PlatformTypeModel;

this.platformService.addPlatform(model).subscrib
e();
  }
}

export class AuthInterceptor implements
HttpInterceptor {

  constructor(private router: Router) { }

  intercept(req: HttpRequest<any>, next:
HttpHandler): Observable<HttpEvent<any>> {
   const token = localStorage.getItem('token');
   if (token) {
    const clonedReq = req.clone({
     headers: req.headers.set('Authorization',
'Bearer ' + token)
    });
    return next.handle(clonedReq).pipe(
     tap(
      succ => {},
      err => {
       if (err.status === 401) {
        localStorage.removeItem('token');
        this.router.navigateByUrl('user/login');
       }
       else if (err.status === 403) {
        this.router.navigateByUrl('/forbidden');
       }
      }
     )
    );
   }
   else {
    return next.handle(req.clone());
   }
  }

}

export class AuthGuard implements CanActivate
{

  constructor(private router: Router, private
userService: UserService) { }

  canActivate(
   next: ActivatedRouteSnapshot,
   state: RouterStateSnapshot): boolean {
    if (localStorage.getItem('token') != null) {
     const roles = next.data.permittedRoles as
Array<string>;
      if (roles) {
       if (this.userService.roleMatch(roles)) {
        return true;
       }
       else {
        this.router.navigate(['/forbidden']);
        return false;
```

```
        }
       }
       return true;
      }
      else {
       this.router.navigate(['games']);
      }
  }

}

export class CommentsComponent implements
OnInit {

  allComments: InputCommentModel[];
  private subscriptions = new Subscription();

  @Input() pageId: number;

  formModel = this.fb.group({
    Body: ['', Validators.required],
  });


  constructor(private commentService:
CommentsService, private fb: FormBuilder) { }

  ngOnInit(): void {

this.subscriptions.add(this.commentService.getAll
Comments(this.pageId).subscribe(comments => {
      this.allComments = comments;
    }));
  }

  AddComment(): void {
    const model = {
      body: this.formModel.value.Body,
    } as InputCommentModel;



this.commentService.addCommentToGame(this.p
ageId, model).subscribe(comments => {
      this.allComments = comments;
    });
  }

}
```

```
@Component({
  selector: 'app-file-upload',
  templateUrl: './file-upload.component.html',
  styleUrls: ['./file-upload.component.scss'],
    animations: [
        trigger('fadeInOut', [
            state('in', style({ opacity: 100 })),
            transition('* => void', [
                animate(300, style({ opacity: 0 }))
            ])
        ])
    ]
})
export class FileUploadComponent implements
OnInit {

  @Input() text = 'Upload';

  @Input() accept = 'image/*';

  @Input() images: ImageModel[];

  constructor(private http: HttpClient) { }

  ngOnInit() {
  }

  onClick() {
    const fileUpload =
document.getElementById('fileUpload') as
HTMLInputElement;
    fileUpload.onchange = () => {
      for (let index = 0; index <
fileUpload.files.length; index++) {
        const file = fileUpload.files[index];
        const reader = new FileReader();
        reader.readAsDataURL(file);
        reader.onload = () => {
          this.images.push({data:
reader.result.slice(reader.result.toString().indexOf
(',') + 1).toString(), title: file.name});
        };
      }
    };
    fileUpload.click();
  }

}
```

```
export class GamesComponent implements
OnInit, AfterViewInit {

  allGames: GameModel[];
  filteredGames: GameModel[];
  filterString: string;
  @ViewChild('flexSearch')
fleaxSearchCheckbox;

  get isFlexSearch(): boolean {
    return
this.fleaxSearchCheckbox?._inputElement?.native
Element?.checked;
  }

  constructor(private gamesService:
GamesService, private nav: NavService,
              private cdr: ChangeDetectorRef) { }

  ngOnInit(): void {
    this.gamesService.getAllGames().subscribe(
games => {
      this.allGames = games;
      this.filteredGames = games;
    });
  }

  ngAfterViewInit() {
    this.cdr.detectChanges();
  }

  filterGames(filter: string) {
    this.filteredGames = this.allGames.filter( game
=> {
      return
game.name.toLowerCase().startsWith(filter.toLo
werCase());
    });
    console.log(this.filteredGames);
  }

  onGameSearch(games: GameModel[]) {
    this.filteredGames = games;
  }

  navigateToPage(page: string, id: number) {
    this.nav.navigateToPage(page, id);
  }
}

export class SearchGamesComponent implements
OnInit, OnDestroy {

  @Output() gamesWereFiltered = new
EventEmitter<GameModel[]>();

  allGenres: GenreModel[];
  allPlatforms: PlatformTypeModel[];
  allPublishers: PublisherModel[];

  private subscriptions = new Subscription();

  formModel = this.fb.group({
    Name: [''],
    MinYear: [''],
    MaxYear: [''],
    Genres: [''],
    ExcludeGenres: [''],
    Publisher: [''],
    ExcludePublishers: [''],
    Platforms: [''],
  });

  constructor(private fb: FormBuilder, private
gameService: GamesService,
              private genreService: GenresService,
private publisherService: PublishersService,
              private platformService:
PlatformsService) { }

  ngOnInit(): void {

this.subscriptions.add(this.genreService.getAllGe
nres().subscribe( genres => {
      this.allGenres = genres;
    }));

this.subscriptions.add(this.platformService.getAll
Platforms().subscribe( pl => {
      this.allPlatforms = pl;
    }));

this.subscriptions.add(this.publisherService.getAll
Publishers().subscribe( pub => {
      this.allPublishers = pub;
    }));
  }
```

```
  SearchGames() {
    const fmv = this.formModel.value;
    const searchGameModel = {
      name: fmv.Name,
      minYear: +fmv.MinYear,
      maxYear: +fmv.MaxYear,
      publisher: fmv.Publisher ? fmv.Publisher : {},
      excludePublishers: fmv.ExcludePublishers ?
fmv.ExcludePublishers : [],
      genres: fmv.Genres ? fmv.Genres : [],
      excludeGenres: fmv.ExcludeGenres ?
fmv.ExcludeGenres : [],
      platformTypes: fmv.Platforms ?
fmv.Platforms : [],
    } as SearchGameModel;


this.gameService.searchGames(searchGameMode
l).subscribe(games => {
      this.gamesWereFiltered.emit(games);
    });
  }

  disableIncludeGenre(genre: GenreModel):
boolean {
    return (this.formModel.value.ExcludeGenres as
GenreModel[]).includes(genre);
  }

  disableExcludeGenre(genre: GenreModel):
boolean {
    return (this.formModel.value.Genres as
GenreModel[]).includes(genre);
  }

  disableIncludePublisher(pub: PublisherModel):
boolean {
    return
(this.formModel.value.ExcludePublishers as
PublisherModel[]).includes(pub);
  }

  disableExcludePublisher(pub: PublisherModel):
boolean {
    return (this.formModel.value.Publisher as
PublisherModel) === pub;
  }

  ngOnDestroy(): void {
    this.subscriptions.unsubscribe();
  }

}

export class MarkDialogComponent {

  constructor(
    public dialogRef:
MatDialogRef<MarkDialogComponent>,
    @Inject(MAT_DIALOG_DATA) public
markValue: number) {}

  onNoClick(): void {
    this.dialogRef.close(false);
  }
}

export class MarkComponent implements OnInit
{

  @Input() markInfo: MarkInfoModel;
  @Input() showVoteSection = false;
  @Output() userVoted = new
EventEmitter<number>();
  public userMark = 0;

  constructor(private dialog: MatDialog, public
userService: UserService) { }

  ngOnInit(): void {
  }

  openMarkDialog(): void {
    const dialogRef =
this.dialog.open(MarkDialogComponent, {
      width: '250px',
      data: this.userMark
    });

    dialogRef.afterClosed().subscribe(result => {
      if (result) {
        this.userMark = result;
        this.userVoted.emit(this.userMark);
      }
    });
  }
```

```
}

export class GameListItemComponent
implements OnInit {

  @Input() game: GameModel;

  constructor(private nav: NavService) { }

  ngOnInit(): void {
  }

  navigateToPage(page: string, id: number) {
    this.nav.navigateToPage(page, id);
  }

}

export class GameComponent implements OnInit,
OnDestroy, AfterViewInit {

  gameId: number;
  game: GameModel;
  genres: GenreModel[];

  @ViewChild('editMode') editModeCheckbox;

  get isEditMode(): boolean {
    return
this.editModeCheckbox?._inputElement?.nativeEl
ement?.checked;
  }

  private subscriptions = new Subscription();

  constructor(private gamesService:
GamesService, private route: ActivatedRoute,
private cdr: ChangeDetectorRef,
            private router: Router, private
navService: NavService, public userService:
UserService) { }

  ngOnInit(): void {
    this.gameId = this.route.snapshot.params.id;

this.subscriptions.add(this.gamesService.getGame
ById(this.gameId).subscribe(res => {
      this.game = res;
      if (this.userService.isAuthenticated()) {
        this.getMarkInfo();
      }
    }));

this.subscriptions.add(this.gamesService.getGenre
sByGameId(this.gameId).subscribe(res => {
      this.genres = res;
    }));

  }

  ngAfterViewInit() {
    this.cdr.detectChanges();
  }

  ngOnDestroy(): void {
    this.subscriptions.unsubscribe();
  }

  navigateToPage(page: string, id: number) {
    this.navService.navigateToPage(page, id);
  }

  voteByGameId(mark: number) {
    this.gamesService.voteByGameId(this.gameId,
mark).subscribe(result => {
      this.game.markInfo = result;
    });
  }

  getMarkInfo() {

this.gamesService.getMarkInfoByGameId(this.ga
meId).subscribe(result => {
      this.game.markInfo = result;
    });
  }

  onGameUpdate() {

this.subscriptions.add(this.gamesService.getGame
ById(this.gameId).subscribe(res => {
      this.game = res;

this.editModeCheckbox._inputElement.nativeEle
ment.checked = false;
    }));
  }
```

```
}

export class GenresComponent implements OnInit
{

  allGenres: GenreModel[];
  filteredGenres: GenreModel[];
  filterString: string;

  constructor(private genreService: GenresService,
private nav: NavService) { }

  ngOnInit(): void {
    this.genreService.getAllGenres().subscribe(
genres => {
      this.allGenres = genres;
      this.filteredGenres = genres;
    });
  }

  filterGenres(filter: string) {
    this.filteredGenres = this.allGenres.filter( news
=> {
      return
news.name.toLowerCase().startsWith(filter.toLow
erCase());
    });
    console.log(this.filteredGenres);
  }

  navigateToPage(page: string, id: number) {
    this.nav.navigateToPage(page, id);
  }

}

Component({
  selector: 'app-genre',
  templateUrl: './genre.component.html',
  styleUrls: ['./genre.component.scss']
})
export class GenreComponent implements OnInit,
OnDestroy {

  genreId: number;
  genre: GenreModel;
  games: GameModel[];
  private subscriptions = new Subscription();

  constructor(private genresService:
GenresService, private route: ActivatedRoute,
          private navService: NavService) { }


  ngOnInit(): void {
    this.genreId = this.route.snapshot.params.id;

this.subscriptions.add(this.genresService.getGenre
ById(this.genreId).subscribe(res => {
      this.genre = res;
    }));

this.subscriptions.add(this.genresService.getGame
sByGenreId(this.genreId).subscribe(res => {
      this.games = res;
    }));
  }

  ngOnDestroy(): void {
    this.subscriptions.unsubscribe();
  }

  navigateToPage(page: string, id: number) {
    this.navService.navigateToPage(page, id);
  }

}

export class ImageComponent implements OnInit
{

  @Input() image: ImageModel;
  @Input() maxWidth: string;
  @Input() maxHeight: string;
  @Input() width: string;
  @Input() height: string;

  constructor() { }

  ngOnInit(): void {
  }

}

export interface GameModel {
  id: number;
```

```
  name: string;
  description: string;
  year: number;
  isDeleted: boolean;
  publisher: PublisherModel;
  genres: GenreModel[];
  platformTypes: PlatformTypeModel[];
  markInfo: MarkInfoModel;
  images: ImageModel[];
}

export interface GenreModel {
  id: number;
  name: string;
  description: string;
  isDeleted: boolean;
}

export interface ImageModel {
  id?: number;
  title?: string;
  data: string;
}

export interface InputCommentModel {
  id: number;
  userName: string;
  body: string;
}

export interface MarkInfoModel {
  numberOfVotes: number;
  averageMark: number;
  userMark: number;
}

export interface NewsModel {
  id: number;
  name: string;
  body: string;
  isDeleted: boolean;
  games: GameModel[];
}

export interface PlatformTypeModel {
  id: number;
  name: string;
  description: string;
  isDeleted: boolean;
}

export interface PublisherModel {
  id: number;
  name: string;
  description: string;
  isDeleted: boolean;
}

export interface SearchGameModel {
  minYear: number;
  maxYear: number;
  publisher: PublisherModel;
  genres: GenreModel[];
  platformTypes: PlatformTypeModel[];
  excludePublishers: PublisherModel[];
  excludeGenres: GenreModel[];
}

export interface LoginModel {
  userName: string;
  password: string;
}

export interface RegistrationModel {
  userName: string;
  email: string;
  password: string;
}

export class NewsComponent implements OnInit
{

  allNews: NewsModel[];
  filteredNews: NewsModel[];
  filterString: string;

  constructor(private newsService: NewsService,
private nav: NavService) { }

  ngOnInit(): void {
    this.newsService.getAllNews().subscribe( news
=> {
      this.allNews = news;
      this.filteredNews = news;
    });
  }

  filterNews(filter: string) {
```

```
    this.filteredNews = this.allNews.filter( game =>
{
      return
game.name.toLowerCase().startsWith(filter.toLo
werCase());
    });
    console.log(this.filteredNews);
  }

  navigateToPage(page: string, id: number) {
    this.nav.navigateToPage(page, id);
  }

}

export class SingleNewsComponent implements
OnInit, OnDestroy {

  singleNewsId: number;
  singleNews: NewsModel;

  private subscriptions = new Subscription();

  constructor(private newsService: NewsService,
private route: ActivatedRoute,
              private navService: NavService) { }


  ngOnInit(): void {
    this.singleNewsId =
this.route.snapshot.params.id;

this.subscriptions.add(this.newsService.getNewsB
yId(this.singleNewsId).subscribe(res => {
      this.singleNews = res;
    }));
  }

  ngOnDestroy(): void {
    this.subscriptions.unsubscribe();
  }

  navigateToPage(page: string, id: number) {
    this.navService.navigateToPage(page, id);
  }

}
```

admin-game.component.html

```
<h1>{{pageName}}</h1>

<form class="form" [formGroup]="formModel">
    <mat-form-field class="full-width">
      <mat-label>Назва</mat-label>
      <input matInput
formControlName="Name">
    </mat-form-field>
    <mat-form-field class="full-width">
      <mat-label>Опис</mat-label>
      <textarea matInput
formControlName="Description"></textarea>
    </mat-form-field>
    <mat-form-field class="full-width">
      <mat-label>Рік виходу</mat-label>
      <input matInput formControlName="Year">
  </mat-form-field>
    <mat-form-field>
      <mat-label>Видавець</mat-label>
      <mat-select formControlName="Publisher">
        <mat-option>Невідомий</mat-option>
        <mat-option *ngFor="let pub of
allPublishers" [value]="pub">
          {{pub.name}}
        </mat-option>
      </mat-select>
    </mat-form-field>
    <mat-form-field>
      <mat-label>Жанри</mat-label>
      <mat-select formControlName="Genres"
multiple>
        <mat-option *ngFor="let genre of allGenres"
[value]="genre">{{genre.name}}</mat-option>
      </mat-select>
    </mat-form-field>
    <mat-form-field>
      <mat-label>Ігрові платформи</mat-label>
      <mat-select formControlName="Platforms"
multiple>
        <mat-option *ngFor="let platform of
allPlatforms"
[value]="platform">{{platform.name}}</mat-
option>
      </mat-select>
    </mat-form-field>
      <app-file-upload [images]="images"></app-
file-upload>
    <div>
```

```
      <button mat-raised-button color="primary"
type="submit" (click)="AddOrUpdateGame()"
[disabled]="!formModel.valid">Додати</button>
    </div>
</form>
```

Admin-genre.component.html

```
<h1>Додайте жанр</h1>

<form class="form" [formGroup]="formModel">
    <mat-form-field class="full-width">
        <mat-label>Назва</mat-label>
        <input matInput
formControlName="Name">
    </mat-form-field>
    <mat-form-field class="full-width">
        <mat-label>Опис</mat-label>
        <textarea matInput
formControlName="Description"></textarea>
    </mat-form-field>
    <div>
      <button mat-raised-button color="primary"
type="submit" (click)="AddGenre()"
[disabled]="!formModel.valid">Додати</button>
    </div>
</form>
```

Admin.component.html

```
<nav class="navigation-wrapper" mat-tab-nav-bar
mat-align-tabs="center">
  <a mat-tab-link routerLink="games" class="tab"
     routerLinkActive #rla0="routerLinkActive"
     [active]="rla0.isActive">
    Ігри
  </a>
  <a mat-tab-link routerLink="genres" class="tab"
     routerLinkActive #rla1="routerLinkActive"
     [active]="rla1.isActive">
    Жанри
  </a>
  <a mat-tab-link routerLink="platforms"
class="tab"
     routerLinkActive #rla2="routerLinkActive"
     [active]="rla2.isActive">
    Платформи
  </a>
  <a mat-tab-link routerLink="publishers"
class="tab"
     routerLinkActive #rla3="routerLinkActive"
     [active]="rla3.isActive">
    Видавці
  </a>
  <a mat-tab-link routerLink="news" class="tab"
     routerLinkActive #rla4="routerLinkActive"
     [active]="rla4.isActive">
    Новини
  </a>
</nav>
<div class="content-wrapper">
  <router-outlet></router-outlet>
</div>
```

Navigation.component.html

```
<nav class="navigation-wrapper" mat-tab-nav-bar
mat-align-tabs="center">
  <a mat-tab-link routerLink="games"
class="navigation-link"
     routerLinkActive #rla0="routerLinkActive"
     [active]="rla0.isActive">
    Ігри
  </a>
   <a mat-tab-link routerLink="news"
class="navigation-link"
     routerLinkActive #rla21="routerLinkActive"
     [active]="rla21.isActive">
    Новини
  </a>
   <a mat-tab-link routerLink="genres"
class="navigation-link"
     routerLinkActive #rla22="routerLinkActive"
     [active]="rla22.isActive">
    Жанри
  </a>
   <a mat-tab-link routerLink="publishers"
class="navigation-link"
     routerLinkActive #rla23="routerLinkActive"
     [active]="rla23.isActive">
    Видавці
  </a>
   <a mat-tab-link routerLink="platforms"
class="navigation-link"
     routerLinkActive #rla24="routerLinkActive"
     [active]="rla24.isActive">
    Платформи
  </a>
  <a mat-tab-link routerLink="admin"
class="navigation-link"
     routerLinkActive #rla1="routerLinkActive"
     *ngIf="userService.roleMatch(['Admin'])"
```

```
    [active]="rla1.isActive">
    Адміністратор
  </a>
  <a mat-tab-link routerLink="user/registration"
class="navigation-link"
    routerLinkActive #rla2="routerLinkActive"
    *ngIf="!userService.isAuthenticated()"
    [active]="rla2.isActive">
    Реєстрація
  </a>
  <a mat-tab-link routerLink="user/login"
class="navigation-link"
    routerLinkActive #rla3="routerLinkActive"
    *ngIf="!userService.isAuthenticated()"
    [active]="rla3.isActive">
    Авторизація
  </a>
  <a mat-tab-link routerLink="games"
class="navigation-link"
    routerLinkActive #rla4="routerLinkActive"
    *ngIf="userService.isAuthenticated()"
    [active]="rla4.isActive"
    (click)="userService.logOut()">
    Вихід
  </a>
</nav>
```

Games.component.html

```
<div>
  <div class="checkbox-wrapper">
    <mat-checkbox #flexSearch>Розширений
пошук</mat-checkbox>
  </div>

  <mat-divider></mat-divider>
  <ng-container *ngIf="!isFlexSearch">
    <mat-form-field class="search-wrapper">
      <mat-label>Пошук за назвою</mat-label>
      <input matInput placeholder="Введіть
назву..." #filterInput
(change)=filterGames(filterInput.value)>
    </mat-form-field>
  </ng-container>
  <ng-container *ngIf="isFlexSearch">
    <app-search-games
(gamesWereFiltered)="onGameSearch($event)">
</app-search-games>
  </ng-container>
</div>

<mat-list role="list">
  <mat-divider style="width: 100%;"></mat-
divider>
  <ng-container *ngFor="let game of
filteredGames">
    <app-game-list-item [game]="game"></app-
game-list-item>
    <mat-divider style="width: 100%;"></mat-
divider>
  </ng-container>
</mat-list>
```

Game.component.html

```
<div class="checkbox-wrapper">
  <mat-checkbox #editMode>Режим
редагування</mat-checkbox>
</div>

<ng-container *ngIf="!isEditMode">
  <div class="header-wrapper">
    <h1 class="header">{{game?.name}}</h1>
  </div>

  <div class="content-wrapper">
    <app-image class="main-image-wrapper"
      [image]="game?.images[0]"
      [maxWidth]="'100%'"
      [maxHeight]="'300px'"></app-image>
    <app-mark [markInfo]="game?.markInfo"
[showVoteSection]="true"
(userVoted)="voteByGameId($event)"></app-
mark>
    <mat-divider class="divider"></mat-divider>
    <ng-container>
      <span class="section-name"
*ngIf="game?.year">Рік випуску:
{{game.year}}</span>
      <mat-divider class="divider"></mat-divider>
    </ng-container>
    <ng-container *ngIf="game?.genres">
      <mat-chip-list>
        <span class="section-name">Жанри:
</span>
        <mat-chip class="chip" *ngFor="let genre of
game?.genres"
(click)="navigateToPage('genres',genre.id)">{{ge
nre?.name}}</mat-chip>
      </mat-chip-list>
      <mat-divider class="divider"></mat-divider>
```

```
    </ng-container>
    <ng-container *ngIf="game?.publisher">
      <mat-chip-list>
        <span class="section-name">Видавець:
</span>
        <mat-chip class="chip"
(click)="navigateToPage('publishers',
game.publisher.id)">{{game.publisher?.name}}</
mat-chip>
      </mat-chip-list>
      <mat-divider class="divider"></mat-divider>
    </ng-container>
    <ng-container *ngIf="game?.platformTypes">
      <mat-chip-list>
        <span class="section-name">Ігрові
платформи: </span>
        <mat-chip class="chip" *ngFor="let pl of
game.platformTypes"
(click)="navigateToPage('platforms',pl.id)">{{pl.
name}}</mat-chip>
      </mat-chip-list>
      <mat-divider class="divider"></mat-divider>
    </ng-container>
    <span class="description">
{{game?.description}} </span>
  </div>
  <app-comments [pageId]="gameId"></app-
comments>
</ng-container>

<ng-container *ngIf="isEditMode">
  <app-admin-game [gameForEdit]="game"
[pageName]="'Редагування гри'"
(gameWasUpdated)="onGameUpdate()"></app-
admin-game>
</ng-container>
```

Game-list-item.component.html

```
<div class="component-wrapper">
  <mat-list-item role="listitem" class="game-item-
wrapper">
    <mat-list role="list">
      <mat-list-item class="game-description-item"
role="listitem">
        <a [routerLink]="game.id"
routerLinkActive="active">{{game.name}}</a>
      </mat-list-item>
      <mat-list-item class="game-description-item"
role="listitem">
        <app-mark
[markInfo]="game.markInfo"></app-mark>
      </mat-list-item>
      <mat-list-item class="game-description-item"
*ngIf="game?.year">
        <span >Рік випуску: {{game.year}}</span>
      </mat-list-item>
      <mat-list-item class="game-description-item"
role="listitem"
        *ngIf="game?.publisher">
        <mat-chip-list>
          <span class="section-name">Видавець:
</span>
          <mat-chip class="chip"
(click)="navigateToPage('publishers',
game.publisher.id)">{{game.publisher?.name}}</
mat-chip>
        </mat-chip-list>
      </mat-list-item>
      <mat-list-item class="game-description-item"
role="listitem"
      *ngIf="game?.genres.length != 0">
        <mat-chip-list>
          <span class="section-name">Жанри:
</span>
          <mat-chip class="chip" *ngFor="let genre
of game?.genres"
(click)="navigateToPage('genres',genre.id)">{{ge
nre?.name}}</mat-chip>
        </mat-chip-list>
      </mat-list-item>
      <mat-list-item class="game-description-item"
role="listitem"
      *ngIf="game?.platformTypes.length != 0">
        <mat-chip-list>
          <span class="section-name">Ігрові
платформи: </span>
          <mat-chip class="chip" *ngFor="let pl of
game.platformTypes"
(click)="navigateToPage('platforms',pl.id)">{{pl.
name}}</mat-chip>
        </mat-chip-list>
      </mat-list-item>
    </mat-list>
  </mat-list-item>
  <app-image class="main-image-wrapper"
    [image]="game?.images[0]"
    [maxWidth]="'100%'"
    [maxHeight]="'300px'"></app-image>
```

```
</div>
```

games.scss

```
.game-item {
  height: fit-content !important;
  margin: 5px 0px 5px 0px;
}

.game-description-item {
  height: fit-content !important;
  margin: 5px 0px 5px 0px;
}

:host {
  width: 100%;
}
.invalid-border {
  border-color: red;
}

.form {
  min-width: 250px;
  max-width: 500px;
  width: 100%;
  display: flex;
  flex-direction: column;
  align-items: center;
  margin: auto;
}

.full-width {
  width: 80%;
}

.half-width {
  width: 40%;
  margin: auto;
}

.group-wrapper {
  display: flex;
  flex-direction: row;
  justify-content: center;
  width: 100%;
}

:host {
  width: 60%;
  min-width: 300px;
  max-width: 435px;
}

.search-wrapper {
  margin: 5px 0 0 25px;
}

.chip {
  padding: 5px 8px !important;
  border-radius: 13px !important;
  min-height: 22px !important;
}

.checkbox-wrapper {
  display: flex;
  justify-content: center;
  margin: 5px 0 5px 0;
}
```

Game-list-item.component.scss

```
.component-wrapper {
  display: flex;
  height: fit-content !important;
  margin: 5px 0px 5px 0px;
}

.game-item-wrapper {
  width: 55%;
  height: 100%;
  // height: fit-content !important;
  // margin: 5px 0px 5px 0px;
}

.game-description-item {
  height: fit-content !important;
  margin: 5px 0px 5px 0px;
}

:host {
  width: 100%;
}

.search-wrapper {
  margin: 5px 0 0 25px;
}

.chip {
  padding: 5px 8px !important;
```

```
  border-radius: 13px !important;
  min-height: 22px !important;
}

.checkbox-wrapper {
  display: flex;
  justify-content: center;
  margin: 5px 0 5px 0;
}

.main-image {
  max-width: 100%;
  max-height: 300px;
}

.main-image-wrapper {
  display: flex;
  width: 40%;
  justify-content: center;
  margin: 10px 0 10px 0;
}
```

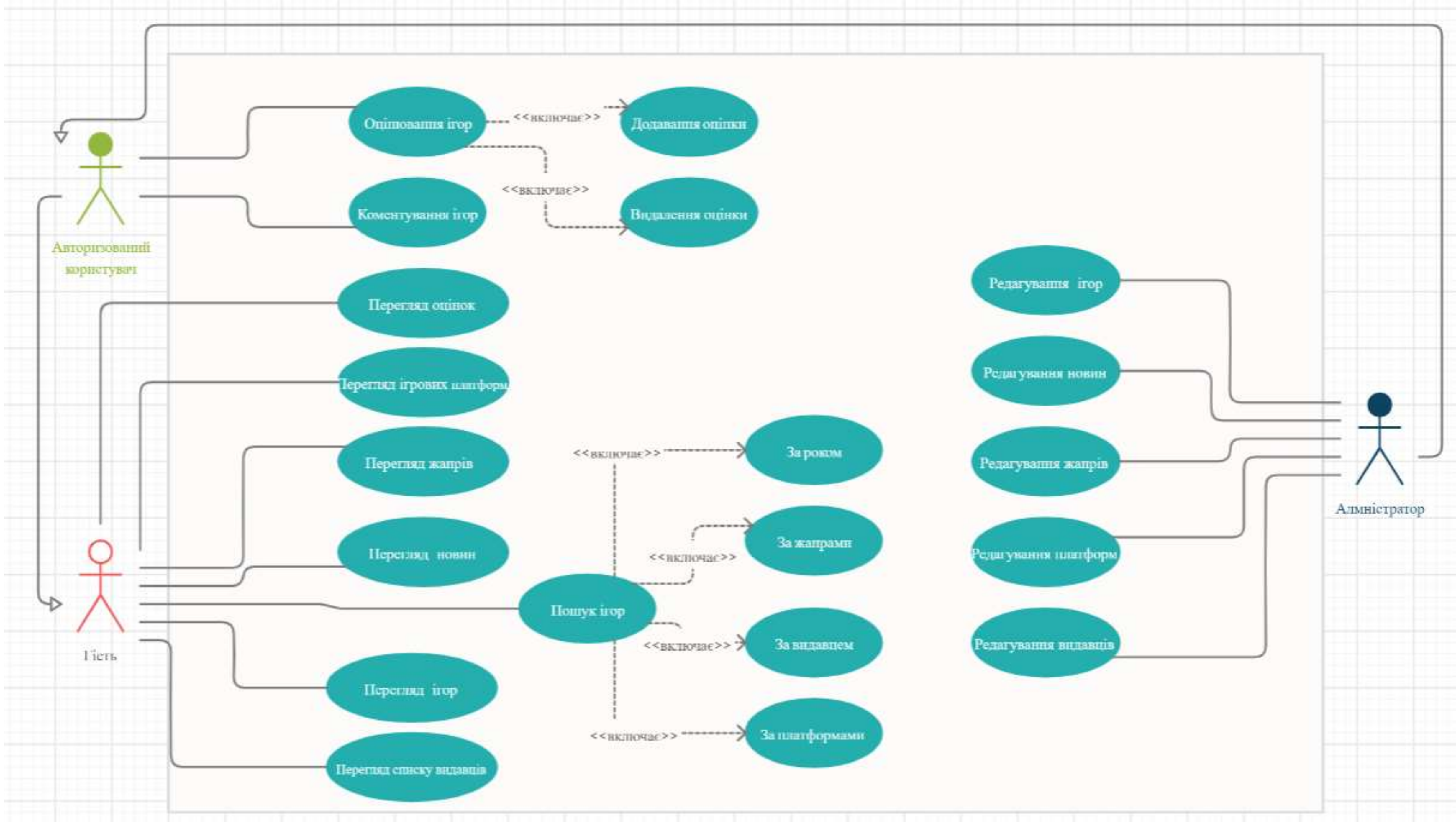
File-upload.component.scss

```
ul,
li {
    margin: 0;
    padding: 0;
    list-style: none;
}

#file-label {
    display: inline-flex;
    vertical-align: middle;
    font-size: 12px;
    line-height: 18px;
}

#file-label mat-icon {
    font-size: 18px;
    text-align: center;
}

#file-label a {
    cursor: pointer;
}
```

Admin-genre.component.scss

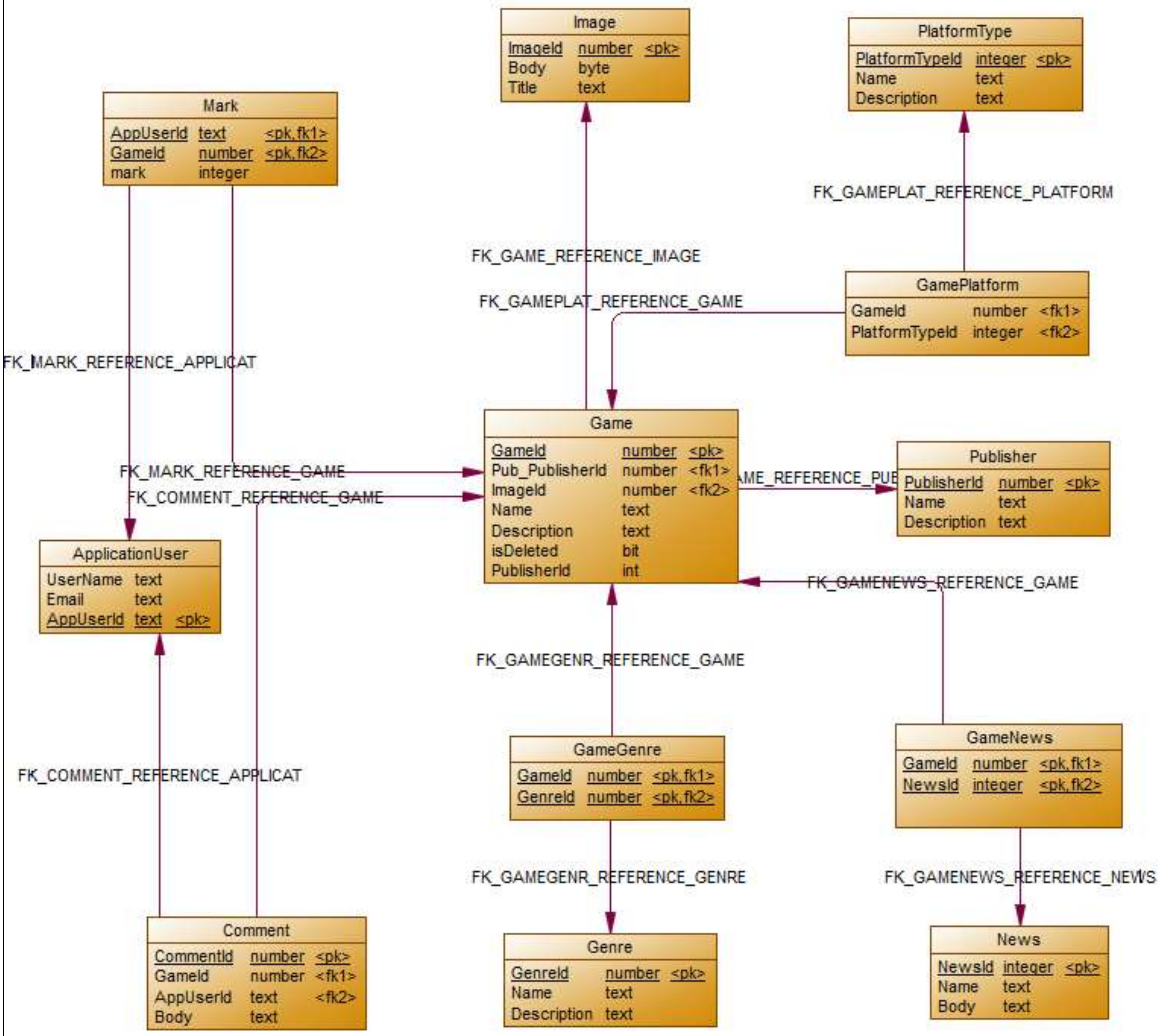
```
.invalid-border {
  border-color: red;
}

.form {
  min-width: 250px;
  max-width: 500px;
  width: 100%;
  display: flex;
  flex-direction: column;
}

.full-width {
  width: 100%;
}

:host {
  width: 60%;
  min-width: 300px;
  max-width: 435px;
}
```

КПІ.ІП-6127.045440.05.СС

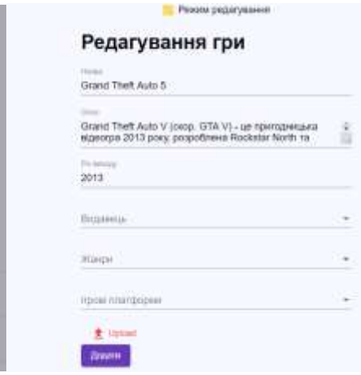


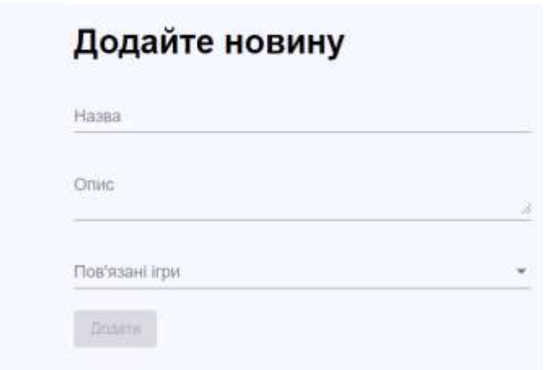
					КПІ.ІП-6127.045440.05.СС		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		ШатровськийА.О.			Схема структурна варіантів використань	Літ.	Маса	Масш.
Перевірив		Крамар Ю.М.					
Т. Контр.						Аркуш 1	Аркушів 1
					Веб-застосування «Ігрова бібліотека»	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61	
Н. Контр.		Ліщук К.І.					
Затверд.							

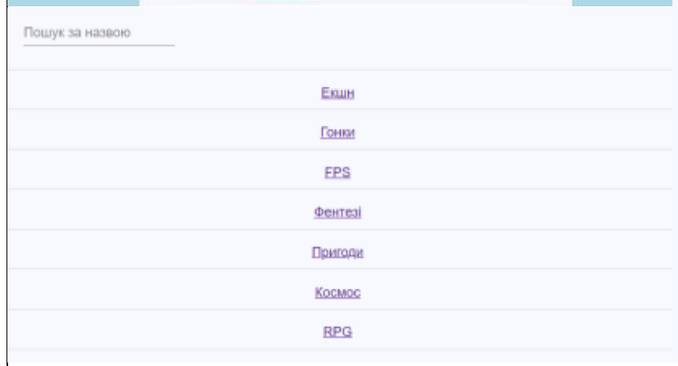
КПІ.ІП6127.045440.06.СБД

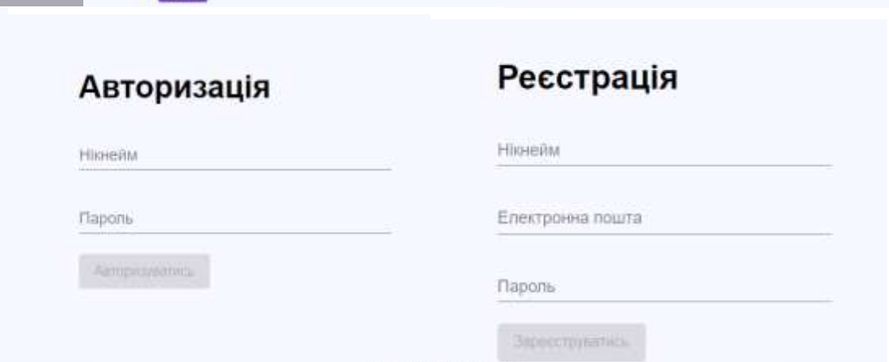


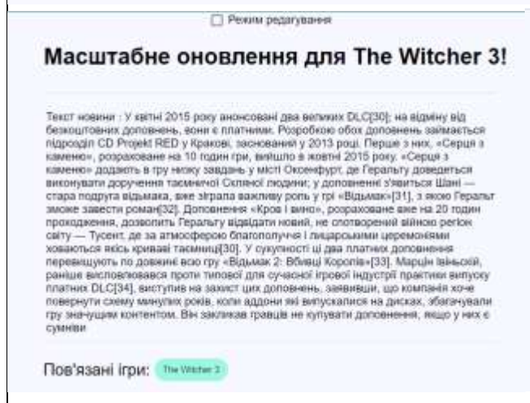
Зм.	Арк.	№ докум.	Підпис	Дата	КПІ.ІП-6127.045440.06.СБД			
Розробив		Шатровський А.О.			Схема бази даних	Літ.	Маса	Масш.
Перевірив		Крамар.Ю.М.						
Т. Контр.						Аркуш 1	Аркушів 1	
					Веб-застосування «Ігрова бібліотека»	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61		
Н. Контр.		Ліщук К.І.						
Затверд.								











					КПІ.ІП-6127.045440.07.КЕ	
Зм.	Арк.	№ докум.	Підпис	Дата	Креслення вигляду екранних форм	Літ. / Маса / Масш.
Розробив		ШатровськийА.О.				
Перевірив		Крамар Ю.М.				Аркуш 1 / Аркушів 1
Т. Контр.						
Н. Контр.		Ліщук К.І.			Веб-застосування «Кулінарний помічник»	КПІ ім. Ігоря Сікорського Кафедра АСОІУ група ІП-61
Затверд.						