

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій
(повне найменування факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалаврський
(ступінь вищої освіти)

на тему: РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ПОШУКУ ІГОР

Виконав(ла): студент(ка) 4 курсу,
групи КНТ-512

Спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

СКОВОРОДНІКОВА А.Ю.
(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.
(ПРИЗВИЩЕ та ініціали)

Рецензент ЩЕРБАКОВ В.І.
(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ комп'ютерних наук і технологій
Кафедра _____ комп'ютерних систем та мереж
Ступінь вищої освіти _____ бакалаврський
Спеціальність _____ 123 комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) _____ комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

« » 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СКОВОРОДНІКОВОЇ АНАСТАСІЇ ЮРІЇВНИ

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка комп'ютерної системи пошуку ігор
керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ С.Ю.

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

затверджені наказом закладу вищої освіти від « 14 » квітня 2026 року № 160

2. Строк подання студентом проєкту (роботи) 01.06.2026 р.

3. Вихідні дані до проєкту (роботи) опис предметної області, технології розробки

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області;

2) Проєктування комп'ютерної системи пошуку ігор;

3) Розробка комп'ютерної системи пошуку ігор;

4) Випробування комп'ютерної системи пошуку ігор.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5	СКРУПСЬКИЙ С.Ю.		
Нормоконтроль	ДЬЯЧУК Т.С.		

7. Дата видачі завдання «14» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	до 25.04.2026	
2	Формування вимог до систем	до 01.05.2026	
3	Проектування архітектури системи	до 05.05.2026	
4	Проектування бази даних	до 15.05.2026	
5	Проектування артефактів системи	до 17.05.2026	
6	Розробка користувацького інтерфейсу	до 19.05.2026	
7	Випробування комп'ютерної системи	до 23.05.2026	
8	Оформлення пояснювальної записки	до 30.05.2026	
9	Проходження нормоконтролю	до 01.06.2026	
10	Перевірка на наявність академічного плагіату	до 03.06.2026	
11	Проходження рецензування	до 05.06.2026	

Студент(ка) _____
(підпис)

Анастасія СКОВОРОДНІКОВА
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи) _____
(підпис)

Степан СКРУПСЬКИЙ
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
98 с., 30 рис., 32 джерела, 1 додаток.

СИСТЕМА ПОШУКУ ІГОР, ANDROID, JAVA, SQLITE,
КАТАЛОГІЗАЦІЯ, ВІДЕОІГРИ, МОБІЛЬНИЙ ЗАСТОСУНОК, БАЗА ДАНИХ

Предмет розробки - комп'ютерна система пошуку ігор, реалізована у вигляді Android-застосунку з локальною реляційною базою даних та гнучкою системою фільтрації. Мета роботи - підвищення зручності та ефективності пошуку ігрового контенту (десктопних, мобільних та настільних ігор) шляхом створення спеціалізованого застосунку, незалежного від алгоритмів комерційних рекомендацій та постійного підключення до мережі інтернет.

Новизна роботи полягає у поєднанні автономного зберігання структурованої інформації про ігри різних платформ у локальній базі даних мобільного пристрою з мінімалістичним інтерфейсом, що забезпечує чистий пошук без нав'язливих маркетингових компонентів.

Практична цінність роботи полягає у можливості використання розробленої системи як швидкого та надійного офлайн-довідника для індивідуального підбору ігор за заданими критеріями.

У роботі виконано аналіз існуючих систем пошуку відеоігор та виявлено обмеження, притаманні сучасним цифровим магазинам: перевантаженість інтерфейсу та комерційну зміщеність результатів. Запропоновано архітектуру на платформі Android з використанням одного Activity та фрагментної навігації. Для зберігання даних реалізовано реляційну базу на основі SQLite. Виконано проектування системи, включаючи структуру бази даних, прототип, основні екрани застосунку. Реалізовано алгоритми динамічного формування SQL-запитів для багатокритеріальної фільтрації контенту. Перевірку працездатності розробленої системи проведено шляхом випробувань базових сценаріїв пошуку.

ЗМІСТ

Вступ.....	7
1 Аналіз предметної області.....	9
1.1 Класифікація відеоігор	9
1.2 Аналіз існуючих систем пошуку відеоігор.....	15
1.3 Функціональні вимоги до системи	19
1.4 Нефункціональні вимоги до системи.....	23
1.5 Вибір технологій побудови системи	25
1.6 Вибір системи керування базою даних	27
1.7 Висновки по розділу	30
2 Проєктування комп'ютерної системи пошуку ігор.....	31
2.1 Апаратні вимоги системи	31
2.2 Проєктування архітектури системи.....	33
2.3 Проєктування варіантів використання.....	35
2.4 Проєктування бази даних	38
2.5 Прототипування системи.....	41
2.6 Проєктування компонентів системи	43
2.7 Висновки по розділу	48
3 Розробка комп'ютерної системи пошуку ігор	49
3.1 Розробка бази даних.....	49
3.2 Реалізація графічного інтерфейсу	55
3.3 Реалізація бізнес-логіки системи	63
3.4 Висновки по розділу	68

4 Випробування комп'ютерної системи пошуку ігор	69
4.1 Випробування бази даних	69
4.2 Випробування сценарію пошуку десктопної гри.....	73
4.3 Випробування сценарію пошуку мобільної гри	78
4.4 Випробування сценарію пошуку настільної гри.....	82
4.5 Висновки по розділу	84
Висновки	86
Перелік джерел посилання	87
Додаток А Слайди презентації.....	90

ВСТУП

Ринок відеоігор за останні десятиліття створив величезні обсяги медіа-контенту. Масове розповсюдження ігор на різних платформах (персональні комп'ютери, консолі, мобільні пристрої), а також зростаюча популярність настільних ігор створюють потребу в зручних інструментах їх каталогізації та пошуку. Існуючі цифрові платформи орієнтовані насамперед на продаж контенту. Їхні інтерфейси перевантажені рекламою, алгоритми пошуку часто підміняються механізмами комерційних рекомендацій. Це ускладнює користувачеві задачу чистого, об'єктивного підбору гри за заданими критеріями.

Актуальність даної дипломної роботи визначається необхідністю створення незалежної, швидкої та зручної комп'ютерної системи пошуку ігор, яка б об'єднувала різні типи ігрових проєктів в єдиній базі та надавала користувачу гнучкі інструменти фільтрації. Запропонований підхід орієнтований на мінімізацію зовнішніх мережових залежностей. Це забезпечує стабільну роботу системи в офлайн-режимі та миттєвий відгук інтерфейсу.

Метою роботи є розробка комп'ютерної системи пошуку ігор у вигляді мобільного Android-застосунку для підвищення ефективності підбору ігрового контенту за багатокритеріальними параметрами. Предметом розробки є комп'ютерна система-каталог, реалізована з використанням локальної бази даних SQLite. Створення системи забезпечує користувачу швидкий доступ до структурованої інформації без нав'язливої монетизації.

Для досягнення поставленої мети у роботі вирішено такі завдання:

- проаналізовано предметну область, існуючі платформи та принципи класифікації ігор;
- визначено функціональні та нефункціональні вимоги до системи;
- спроектовано архітектуру мобільного застосунку та нормалізовану реляційну базу даних;
- розроблено програмний інтерфейс і бізнес-логіку системи мовою Java;

- проведено комплексні випробування розробленого рішення.

В дипломі основна робота полягає в розробці мобільного клієнта, де локальна база даних виконує роль основного джерела інформації. Новизна роботи полягає у створенні універсального агрегатора, що охоплює десктопні, мобільні та настільні ігри, поєднуючи автономну роботу з можливістю зовнішніх переходів через посилання. Практичне значення отриманих результатів - це готовність розробленої системи до використання як зручного персонального довідника для геймерів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Класифікація відеоігор

Сучасні відеоігри перетворилися з просто розваги для вузького кола до глобального феномену. Відеоігри сьогодні - це не просто програмний продукт, а складні системи, що вимагають системного підходу до їхнього аналізу та каталогізації. Як зазначає Ернест Адамс у своїй роботі з геймдизайну, чітка класифікація є необхідною умовою для проектування будь-якої інформаційної системи, що взаємодіє з ігровим контентом, оскільки вона визначає очікування користувача та технічні межі реалізації [1].

Класифікація ігор - завдання складне через постійну зміну жанрів та появу гібридних форм. Проте найважливішим критерієм, який визначає архітектурні особливості розробки програмного забезпечення, є цільова апаратна платформа. У межах розроблюваної системи доцільно виділити три основні категорії: комп'ютерні (консольні) ігри, мобільні ігри та настільні ігри. Кожна з цих категорій має унікальні характеристики, що впливають на інтерфейс взаємодії з користувачем, механіки, користувацький досвід та технічні вимоги до системи пошуку.

Комп'ютерні та консольні ігри (PC/Console Games) традиційно вважаються сегментом «високих досягнень» або AAA-проектами. Вони орієнтовані на максимальне занурення користувача, використання потужних графічних процесорів та складних обчислювальних алгоритмів. Наприклад, гра Elden Ring [2] стала еталоном сучасного Action-RPG. Вона пропонує величезний відкритий світ, де кожна деталь архітектури та механіка взаємодії з оточенням вимагає значних апаратних ресурсів. У контексті пошукової системи такі ігри вимагають деталізації за платформою (PC, Xbox, PlayStation), оскільки користувачу важливо знати доступність проекту саме для його пристрою.

Іншим прикладом високобюджетного сегменту є Cyberpunk 2077 [3] від польської студії CD Projekt Red. Це проект, що поєднує в собі елементи шутера

від першої особи та глибоку рольову систему в сетингу дистопічного майбутнього. Складність таких ігор обумовлює необхідність їх класифікації в першу чергу за жанром. Також важливо - за рівнем вікового рейтингу та можливістю грати одному чи в мультиплеєр. До цієї ж категорії відноситься *Baldur's Gate 3* [4], яка повернула популярність класичним партійним рольовим іграм. Її успіх підкреслює важливість такої характеристики, як жанрова приналежність. Тут ми бачимо складне переплетення тактичних боїв та розгалужених діалогів, що змушує систему пошуку оперувати точними тегамі. Український проєкт *S.T.A.L.K.E.R. 2* [5] демонструє специфіку жанру *Survival Horror* у поєднанні з відкритим світом. Такі ігри стають центральними об'єктами в базах даних через свій культовий статус та великий обсяг супутньої інформації.

Мобільні ігри - це категорія, що призначена для запуску на мобільних пристроях (смартфони, планшети). Тут на перший план виходять зручність інтерфейсу, адаптивність дизайну, доступність, швидкість сесії та специфічні методи монетизації. Розробники мобільних застосунків вважають що, юзабіліті та адаптивність інтерфейсу є критичними для успіху в цьому сегменті [6]. Яскравим представником є *Genshin Impact* [7], яка фактично стерла межу між мобільними та стаціонарними платформами, пропонуючи консольну якість графіки на смартфонах. Пошук у цій категорії часто спирається на такі фільтри, як наявність багатокористувацького режиму та можливість гри в офлайн-режимі.

Популярним є сегмент мобільних карткових ігор, як-от *Marvel Snap* [8]. Тут механіки орієнтовані на швидкі ігрові цикли та колекціонування. Для користувача мобільної платформи пріоритетним в грі є зручність інтерфейсу, а в пошуку гри - пряме посилення на маркетплейс та популярність гри.

Третя категорія - настільні ігри (*Board Games*). Хоча вони є аналоговими за своєю природою, їхня цифровізація та роль у геймерській спільноті є істотною. Сучасні настільні ігри, такі як, наприклад, *Dune: Imperium* [9], часто мають цифрові адаптації або додатки-компаньйони. Класифікація настільних ігор суттєво відрізняється від відеоігор: тут критичними параметрами є кількість гравців, тривалість партії та складність правил. Наприклад, *Wingspan* [10]

орієнтована на широку аудиторію та має низький поріг входження, тоді як стратегії на кшталт Ark Nova [11] вимагають від гравця глибокого аналізу. Включення настільних ігор у нашу систему пошуку розширює її можливості для покриття і такого сегменту ігор та аудиторії.

Розглянемо жанри відеоігор, оскільки це є основним критерієм пошуку. Це безпосередньо впливає на ігровий процес та користувацький досвід. До одного з найбільш затребуваних напрямків сучасності належить жанр Action-RPG. Це гібридна категорія, що органічно поєднує динаміку бойових систем у реальному часі з глибокою механікою розвитку персонажа. Основний акцент тут робиться на вдосконаленні навичок героя, підборі спорядження та прийнятті сюжетних рішень, що безпосередньо впливають на стан ігрового світу. Такі проєкти вимагають від гравця як швидкої реакції, так і стратегічного планування ресурсів.

Жанр Survival Horror фокусується на створенні гнітючої атмосфери та відчуття постійної вразливості користувача. На відміну від звичайних бойовиків, тут ресурси (патрони, медикаменти) суворо обмежені, а ігрові механіки часто змушують уникати прямих зіткнень із ворогом. Жанр апелює до психологічної напруги, активно використовуючи звуковий дизайн та візуальні обмеження для посилення ефекту занурення в ігровий процес.

Стратегії вимагають від гравця насамперед аналітичного мислення та довгострокового планування. У таких іграх користувач бере на себе роль керівника, менеджера чи полководця, керуючи ресурсами, розбудовуючи складні системи та координуючи дії великих груп юнітів. Перемога зазвичай досягається завдяки правильному економічному розвитку або вибору оптимальної тактики, що робить цей жанр популярним серед поціновувачів складних логічних завдань.

Напрямок MOBA став фундаментом для сучасного кіберспорту. Основна концепція полягає у протистоянні двох команд на обмеженій локації, де кожен учасник керує одним унікальним героєм. Успіх у таких іграх залежить від злагодженої командної взаємодії, стратегічного захоплення ключових точок та вміння швидко адаптуватися до дій супротивника в умовах динамічної битви. Тобто це виключно багатокористувацькі ігри, де упор не на сюжет, а на процес.

Головоломки (Puzzle) орієнтовані на інтелектуальний виклик і розв'язання логічних задач. Вони можуть базуватися на маніпуляції фізичними об'єктами, розпізнаванні складних патернів або просторовому мисленні. Це один із найбільш універсальних жанрів, який часто не потребує високої швидкості реакції, що робить його привабливим для широкого кола користувачів.

Battle Royale - це змагальний жанр, що базується на механіці виживання великої кількості гравців на мапі, яка поступово зменшується в розмірах. Учасники починають гру без жодного спорядження і змушені швидко шукати ресурси, водночас уникаючи небезпечних зон та знищуючи опонентів. Визначальною рисою є непередбачуваність кожної сесії та висока напруга до останніх секунд матчу.

Dungeon Crawler занурює гравців у дослідження замкнених локацій, розгалужених лабіринтів або небезпечних підземель. Основою ігрового циклу є поступове відкриття мапи, подолання пасток, битви з монстрами та збір цінних предметів. Такі ігри часто використовують елементи випадкової генерації рівнів, що забезпечує високу повторюваність ігрового досвіду та постійне відчуття новизни.

У сегменті настільних ігор окреме місце посідають єврогейми. Це стратегії з акцентом на економічну складову та управління ресурсами, де роль випадковості зведена до мінімуму. Основна механіка зазвичай полягає у вибудовуванні ефективних виробничих ланцюжків, а конфлікти між гравцями мають опосередкований характер, що робить цей жанр ідеальним для інтелектуального сімейного дозвілля.

Жанр Deck-building (побудова колоди) передбачає, що формування основного інструментарію гравця відбувається безпосередньо під час партії. Кожен учасник починає з базовим набором елементів і поступово додає нові, більш потужні компоненти, створюючи унікальні комбінації. Цей жанр вимагає від гравця здатності прораховувати ймовірності та гнучко змінювати стратегію залежно від отриманих карт.

Видавці ігор виступають зв'язуючою ланкою між розробником та кінцевим

споживачем. Провідні світові компанії забезпечують повний цикл життя продукту: від фінансування та маркетингового просування до локалізації та технічної підтримки користувачів. Діяльність великих видавців дозволяє формувати високі стандарти якості та гарантувати доступність ігрового контенту. Без якісного маркетингу, просування гри та впливу на її розробку, кінцевий продукт не набуде популярності, тому роль видавців вкрай важлива. Ігри, випущені видавцями з хорошою репутацією притягують користувачів. До таких популярних видавців можна віднести CD Projekt Red, Ubisoft, GSC Game World, та інших.

Для організації безпеки користувачів та інформування батьків про характер контенту використовуються вікові рейтинги [12]. Категорія PEGI 3 вважається універсальною та вказує на повну відсутність потенційно шкідливих сцен, що робить ігри придатними для всіх вікових груп. Рейтинг PEGI 12 вже допускає наявність легкого фантастичного насильства або дещо грубішого гумору, які сприймаються підлітками як частина ігрової умовності. Маркування PEGI 16 свідчить про присутність реалістичного насильства, сцен вживання тютюну чи алкоголю, що потребує певної зрілості від гравця. Найвищий рівень, PEGI 18, призначений суворо для дорослої аудиторії, оскільки може містити сцени жорстокості, ненормативну лексику або контент, здатний викликати сильний емоційний дистрес. Ця класифікація є обов'язковим елементом при проектуванні сучасних інформаційних систем, що працюють з медіа-даними.

Окремим аспектом класифікації, що безпосередньо впливає на технічну архітектуру та користувацький досвід, є поділ на однокористувацькі (Single-player) та багатокористувацькі (Multiplayer) режими. Однокористувацькі ігри зосереджені на індивідуальному зануренні в сюжет, дозволяючи гравцеві досліджувати світ у власному темпі чи потреби в постійному підключенні до мережі. Їхньою головною перевагою є сюжет та відсутність технічних проблем, пов'язаних із затримкою інтернет-сигналу, проте вони часто мають обмежений рівень реграбельності після завершення основної кампанії. Натомість багатокористувацькі режими пропонують динамічне соціальне середовище, де

ігровий досвід щоразу змінюється залежно від дій інших людей. Це створює високий рівень змагальності та стимулює формування спільнот, але водночас робить користувача залежним від стабільності серверів розробника та якості інтернет-з'єднання.

Сучасний ринок відеоігор також чітко сегментований за апаратними платформами, кожна з яких має свою специфіку доступу до контенту та технічні особливості:

- персональний комп'ютер (PC) залишається найбільш гнучкою та відкритою платформою. Його головна особливість - мультифункціональність та можливість апаратного оновлення, що дозволяє запускати ігри з максимальною графічною якістю. Для користувача PC характерним є доступ до різноманітних цифрових магазинів (типу Steam, Epic Games Store, GOG та інші) та величезної бази модифікацій, створених спільнотою, що значно подовжує життя ігрових проєктів;

- Xbox (серії Series X та Series S) від компанії Microsoft [13] позиціонується як частина єдиної екосистеми з Windows. Ключовою перевагою платформи є сервіс Game Pass - модель підписки, що надає доступ до сотень ігор за фіксовану щомісячну плату. Технічно Xbox виділяється функцією Quick Resume, яка дозволяє миттєво перемикатися між декількома запущеними іграми, зберігаючи їх стан, що значно економить час гравця;

- PlayStation 5 (PS5) від Sony [14] робить ставку на ексклюзивність та технологічну унікальність ігрового досвіду. Платформа відома своїми високобюджетними сюжетними іграми, які доступні лише в межах цієї екосистеми. Особливістю PS5 є контролер з просунутою тактильною віддачею та адаптивними тригерами, що імітують опір (наприклад, натягування тятиви лука).

Розуміння цих відмінностей дозволяє нам при проєктуванні бази даних передбачити окремі атрибути для кожної платформи, забезпечуючи користувачеві можливість точного пошуку гри саме під його пристрій.

Таким чином, для побудови ефективної комп'ютерної системи пошуку необхідно врахувати багаторівневу структуру даних. Кожна гра має бути описана

через набір атрибутів, що включають категорію гри, пристрій (платформу), жанрову приналежність, видавця, вікові обмеження, рік випуску. Це дозволяє не тільки впорядкувати великі обсяги інформації в базі даних, а й реалізувати інтуїтивно зрозумілі фільтри для користувача.

1.2 Аналіз існуючих систем пошуку відеоігор

Для формування обґрунтованих вимог до розроблюваної системи необхідно детально проаналізувати сучасні програмні рішення, що забезпечують пошук, каталогізацію та навігацію по ігровому контенту. Існуючі системи можна спробувати класифікувати на десктопні платформи, що інтегровані з цифровими магазинами або лаунчерами, та мобільні застосунки, що орієнтовані на швидкий доступ і рекомендації.

Десктопні застосунки характеризуються високою функціональністю, глибокою інтеграцією з ігровими сервісами та орієнтацією на екосистему користувача.

Steam [15] є найбільш розвиненою системою пошуку ігор серед десктопних платформ. Пошуковий механізм дозволяє комбінувати купу параметрів: жанри, піджанри, теги, підтримка контролера, режим гри, дата релізу, рейтинг користувачів, цінові діапазони. Водночас інтерфейс перенасичений інформацією: рекламні банери, рекомендаційні блоки, акційні пропозиції, стрічки новин. Це створює візуальне навантаження та ускладнює виконання базового сценарію - швидко знайти гру за заданими критеріями. Крім того, алгоритми пошуку часто мають комерційний характер, просуваючи популярні або вигідні для платформи продукти.

Epic Games Store [16] має спрощену структуру інтерфейсу, що позитивно впливає на зручність використання. Однак пошуковий функціонал значно обмежений: фільтрація значно менш деталізована, купа реклами. Основний

акцент зроблено на вітрині продуктів і безкоштовних роздачах, що знову ж таки підкреслює комерційну спрямованість.

GOG Galaxy [17] виконує роль агрегатора ігрових бібліотек. Користувач може інтегрувати різні сервіси та здійснювати пошук у власній колекції. Проте функціональність глобального пошуку нових ігор обмежена, а інтерфейс містить значну кількість допоміжних елементів - статистика, досягнення, соціальні функції. Це знижує ефективність використання застосунку як чистого інструменту пошуку.

Ubisoft Connect [18] є прикладом вузькоспеціалізованої платформи, орієнтованої на ігри одного видавця. Пошук тут обмежений внутрішнім каталогом Ubisoft і не має універсального характеру. Інтерфейс зосереджений на новинах, подіях та внутрішній екосистемі сервісу.

Electronic Arts App [19] аналогічно обмежений власною екосистемою. Пошук працює лише в межах продуктів EA, що робить його непридатним для загального аналізу ринку ігор. Значна частина інтерфейсу зайнята рекламними блоками та підписковими сервісами.

Xbox App [20] для Windows поєднує функції бібліотеки, підписки Game Pass і пошуку. Хоча система дозволяє знаходити ігри за категоріями, вона орієнтована на підпискову модель споживання. Інтерфейс містить рекомендації, соціальні елементи та інтеграцію з акаунтом Microsoft, що відволікає від задачі пошуку.

Playnite [21] є відкритим менеджером бібліотеки ігор, який дозволяє агрегувати різні платформи. Він має гнучкі можливості фільтрації у межах локальної колекції, але не є повноцінною системою пошуку нових ігор. Його функціональність спрямована на організацію вже наявного контенту.

Таким чином, десктопні системи мають такі спільні характеристики:

- відсутність універсального охоплення різних типів ігор (мобільних, настільних);
- орієнтація на купівлю або управління бібліотекою;
- складний, перевантажений інтерфейс;
- наявність маркетингових і соціальних компонентів.

Мобільні застосунки мають іншу парадигму - вони оптимізовані під швидку взаємодію, але мають обмежену глибину функціоналу.

Google Play Store [21] є основним інструментом пошуку мобільних ігор на Android. Він пропонує базову класифікацію за категоріями, рейтингами, кількістю завантажень. Водночас система сильно залежить від алгоритмів рекомендацій, реклами та персоналізації. Результати пошуку часто зміщені у бік популярних або комерційно вигідних продуктів.

MiniReview [22] є більш спеціалізованим застосунком, що пропонує структуровані огляди, рейтинги та фільтрацію. Він дозволяє здійснювати більш точний пошук у порівнянні з офіційними магазинами, однак охоплює лише мобільні ігри та не інтегрує інші платформи.

IGDB App [23] забезпечують доступ до великої бази відеоігор з детальною інформацією: жанри, платформи, рейтинги. Проте інтерфейс таких застосунків часто складний, а функціональність не оптимізована під простий фільтраційний пошук.

GameTrack [24] використовується для відстеження ігор, створення списків (backlog, wishlist) та отримання рекомендацій. Основна функція - організація ігрового досвіду, а не пошук нових ігор за параметрами.

RAWG App [25] надають доступ до великої бази ігор із розширеною інформацією. Вони мають фільтри, але інтерфейс часто перевантажений даними, а також присутні елементи соціальної взаємодії.

GamerPower [26] орієнтований на відстеження безкоштовних роздач і акцій. Пошук як такий є вторинною функцією, а основний акцент зроблено на моніторингу пропозицій.

Мобільні застосунки для пошуку ігор мають такі спільні недоліки:

- фокус виключно на мобільному сегменті (за винятком окремих агрегаторів);
- обмежені можливості фільтрації;
- сильна залежність від рекомендаційних алгоритмів;
- орієнтація на встановлення або взаємодію з маркетплейсом;

- відсутність підтримки настільних ігор.

Проведений аналіз дозволив виділити наступні недоліки сучасних маркетплейсів:

- комерційна спрямованість. Більшість платформ інтегрують функції купівлі, підписки або встановлення. Це впливає на результати пошуку, які оптимізуються не під інформаційні потреби користувача, а під продаж;
- перевантаженість інтерфейсу. Велика кількість другорядних елементів (банери, рекомендації, соціальні функції) ускладнює швидкий доступ до потрібної інформації;
- відсутність універсальності. Десктопні застосунки працюють переважно з РС або консольними іграми, мобільні - лише з мобільними. Настільні ігри практично не представлені;
- обмежена гнучкість пошуку. Навіть у найрозвиненіших системах фільтрація часто обмежена або залежить від внутрішніх алгоритмів платформи;
- зміщення акценту з пошуку на рекомендації. Багато систем підміняють класичний пошук рекомендаційними механізмами.

Таким чином, існуючі рішення не забезпечують універсального інструменту пошуку ігор. Це обґрунтовує необхідність розробки власної комп'ютерної системи, яка:

- об'єднує в єдиній базі комп'ютерні, мобільні та настільні ігри;
- не містить функцій купівлі або комерційного просування;
- має мінімалістичний, функціонально орієнтований інтерфейс;
- реалізує гнучку систему фільтрів;
- забезпечує швидкий доступ до структурованої інформації та зовнішніх ресурсів (офіційні сайти ігор).

Такий підхід дозволяє орієнтувати нашу систему на задачах пошуку, а не на комерційну взаємодію з користувачем.

1.3 Функціональні вимоги до системи

Функціональні вимоги визначають, які саме дії повинна виконувати система та які задачі вона має вирішувати з точки зору кінцевого користувача. Мова йде про комп'ютерну систему, що включає мобільний застосунок, основна мета якого - забезпечити зручний та гнучкий пошук ігор різних типів з можливістю перегляду детальної інформації.

Перш за все система повинна забезпечувати користувачу можливість вибору категорії ігор. Передбачається, що користувач може працювати з кількома основними типами ігор - комп'ютерними (включаючи консольні), мобільними та настільними. Вибір категорії впливає на подальшу логіку роботи системи та доступний набір параметрів для пошуку.

Важливою функцією системи є реалізація пошуку ігор за набором фільтрів. Користувач повинен мати можливість формувати запит, комбінуючи декілька критеріїв одночасно. До параметрів пошуку належать:

- жанр гри;
- видавець;
- рік випуску (з можливістю задати нижню межу);
- віковий рейтинг;
- можливість гри без підключення до мережі;
- наявність багатокористувацького режиму.

Система повинна коректно обробляти ситуації, коли користувач задає лише частину параметрів. Тобто пошук не повинен вимагати обов'язкового заповнення всіх полів. Результати мають формуватися на основі доступних критеріїв, що підвищує гнучкість і зручність використання.

Важливою функціональною вимогою є відображення результатів пошуку у вигляді списку на окремому екрані. Кожен елемент списку повинен містити мінімально необхідний набір інформації для швидкого ознайомлення: назву гри, рік випуску та зображення (постер або обкладинку). Це дозволяє користувачу

швидко зорієнтуватися серед результатів без переходу до детального перегляду кожної позиції.

Система повинна забезпечувати можливість вибору конкретної гри зі списку результатів. Після вибору користувач отримує доступ до розширеної інформації про гру. До цієї інформації мають входити:

- повна назва гри;
- текстовий опис;
- жанрова приналежність;
- видавець;
- рік випуску;
- віковий рейтинг;
- характеристики режимів гри, тобто однокористувацький,

багатокористувацький, офлайн, онлайн.

Окремою функцією є робота з зовнішніми посиланнями. Система повинна надавати користувачу можливість перейти на офіційний ресурс гри або сторінку завантаження. При цьому відкриття посилання має відбуватися у зовнішньому браузері пристрою, без реалізації власного механізму перегляду веб-сторінок. Це спрощує реалізацію та відповідає типовим сценаріям використання мобільних застосунків.

Система також повинна забезпечувати коректне відображення мультимедійного контенту, зокрема зображень ігор, які завантажуються за зовнішніми посиланнями. Важливо, щоб цей процес був прозорим для користувача і не вимагав додаткових дій з його боку.

Ще однією функціональною вимогою є забезпечення логічної цілісності даних під час пошуку. Наприклад, якщо гра не має певних характеристик (наприклад, не підтримує мультиплеєр), вона не повинна потрапляти у результати при встановленні відповідного фільтра. Аналогічно, результати мають відповідати обраній категорії і не змішувати різні типи ігор.

Система повинна обробляти ситуації, коли за заданими критеріями не знайдено жодного результату. У такому випадку користувачу має бути надане

зрозуміле повідомлення, а також можливість змінити параметри пошуку.

Система не повинна реалізовувати функціонал купівлі або встановлення ігор. Її основна роль - інформаційна: допомогти користувачу знайти гру, ознайомитися з нею та перейти на відповідний зовнішній ресурс за потреби. Це дозволяє уникнути перевантаження функціоналу та зосередитися саме на задачі пошуку.

Додатково до основного функціоналу, система повинна підтримувати відображення рекламного банера у контекстно релевантному місці - безпосередньо на екрані зі списком відібраних ігор. Банер має інтегруватися у загальний інтерфейс таким чином, щоб не порушувати логіку взаємодії користувача з контентом і не відволікати від основної задачі - підбору гри за заданими критеріями. Його поява повинна бути природною частиною інтерфейсу, а не нав'язливим елементом. Ключова вимога - відсутність примусової взаємодії. Користувач не зобов'язаний взаємодіяти з банером для продовження роботи із застосунком. Перехід за рекламним банером відбувається виключно за ініціативою користувача. У разі натискання, користувач перенаправляється до зовнішнього ресурсу (онлайн-магазину або платформи дистрибуції ігор), де він може ознайомитися з додатковими пропозиціями та здійснити покупку гри.

Функціонально це означає, що система повинна:

- забезпечувати завантаження та відображення рекламного контенту (статичного або динамічного);
- підтримувати обробку кліку по банеру з подальшим відкриттям зовнішнього веб-ресурсу;
- не блокувати основний сценарій використання застосунку незалежно від того, чи відбувся перехід за рекламою;
- зберігати стабільність інтерфейсу при відсутності або помилці завантаження банера (наприклад, у випадку відсутності мережі).

Такий підхід до монетизації суттєво відрізняється від того, як це реалізовано в більшості сучасних застосунків, зокрема Google Play Store [21] або Steam [15]. У подібних системах реклама або рекомендаційні блоки часто

інтегровані агресивніше: вони займають значну частину інтерфейсу, змішуються з основним контентом або впливають на порядок відображення результатів. Це створює ефект «зашумлення» інтерфейсу, коли користувачеві складніше швидко знайти релевантну інформацію. У деяких випадках (особливо в мобільних ігрових застосунках) використовуються повноекранні рекламні вставки або примусові перегляди перед доступом до функціоналу, що негативно впливає на користувацький досвід і може викликати відторгнення.

Запропонована мінімалістична модель має іншу логіку: реклама існує як додатковий, а не домінуючий елемент. Вона не втручається у процес фільтрації чи вибору гри, не змінює результати пошуку і не нав'язує взаємодію. Це дозволяє зберегти чистоту інтерфейсу, що важливо для застосунків, орієнтованих на швидкий підбір контенту. Монетизація вбудовується як «опціональний шар» поверх основного функціоналу: користувач отримує повноцінний досвід без жодних обмежень, але при цьому має можливість перейти до комерційного контенту, якщо це йому цікаво. Такий підхід підвищує довіру до застосунку.

Таким чином, функціональність системи будується навколо чотирьох основних сценаріїв:

- формування запиту користувачем через набір фільтрів;
- отримання та перегляд списку результатів;
- перегляд детальної інформації про вибрану гру з можливістю переходу на зовнішній ресурс;
- перехід в магазин (через рекламний банер) для купівлі гри.

Реалізація цих функцій повинна забезпечити користувачу простий та зрозумілий інструмент для пошуку ігор різних категорій без зайвих елементів, характерних для комерційних платформ.

1.4 Нефункціональні вимоги до системи

Система повинна забезпечувати стабільну і передбачувану роботу в межах типових сценаріїв використання. Будь-які дії користувача: застосування фільтрів, відкриття списку ігор, перегляд детальної інформації - мають виконуватися без відчутних затримок. У нормальних умовах відгук інтерфейсу повинен залишатися швидким, без «підвисань» чи ривків під час прокрутки або переходів. Якщо виконання операції займає більше часу (наприклад, через мережу), система повинна це явно показувати через індикатор завантаження повідомлення.

Вимоги до продуктивності та швидкодії. Продуктивність застосунку має залишатися стабільною навіть при збільшенні обсягу даних:

- фільтрація і пошук повинні виконуватися без суттєвого погіршення швидкості при рості кількості записів;
- прокрутка списків повинна бути плавною, без фризів;
- повторне відкриття вже переглянутих даних (наприклад, списку або деталей гри) має відбуватися швидше за рахунок оптимізації доступу до даних;
- завантаження зображень не повинно блокувати інтерфейс і має виконуватися у фоновому режимі.

Система також повинна ефективно працювати з ресурсами: не створювати зайвих навантажень, не виконувати непотрібних операцій і не тримати активні процеси без причини.

Вимоги до сумісності. Застосунок повинен коректно працювати на різних версіях операційної системи Android. Не допускається використання застарілих підходів, що можуть призвести до некоректної роботи на окремих версіях ОС. Інтерфейс повинен адаптуватися до різних розмірів екранів і співвідношень сторін без втрати функціональності або читабельності. Усі основні елементи мають залишатися доступними незалежно від форм-фактора пристрою.

Робота в офлайн-режимі. Система повинна забезпечувати можливість повноцінної роботи без підключення до інтернету. При цьому відсутність мережі

не повинна впливати на доступ до вже наявних даних. Основний функціонал: перегляд списків ігор, застосування фільтрів, відкриття детальної інформації - має бути доступний в офлайн-режимі без обмежень завдяки використанню бази даних прямо в смартфоні. У випадках, де потрібне мережеве підключення (наприклад, завантаження зображень або перехід за зовнішніми посиланнями), система повинна:

- коректно обробляти відсутність інтернету;
- повідомляти користувача про неможливість виконання дії;
- не допускати збоїв або аварійного завершення роботи.

Вимоги до надійності. Застосунок не повинен аварійно завершувати роботу при нестандартних ситуаціях - відсутності окремих даних, помилках завантаження або некоректних вхідних значеннях. Усі такі випадки мають оброблятися передбачувано: з відображенням заглушок, повідомлень або альтернативних сценаріїв. Система повинна зберігати свій стан при перезапуску або зміні контексту (наприклад, повернення до застосунку після згортання), щоб користувач не втрачав свій прогрес (обрані фільтри, відкритий список тощо).

Вимоги до зручності використання. Інтерфейс має бути інтуїтивно зрозумілим і не перевантаженим. Користувач повинен швидко орієнтуватися у функціоналі без додаткових пояснень. Важливо забезпечити логічну послідовність дій і однакову поведінку елементів у схожих ситуаціях. Не допускається надлишкова кількість дій для досягнення результату, тобто ключові сценарії мають виконуватися максимально просто.

Вимоги до мережевої взаємодії. Усі операції, пов'язані з мережею, повинні виконуватися асинхронно і не блокувати інтерфейс. У випадку помилок, наприклад, система повинна коректно реагувати - повідомляти користувача і дозволяти повторити спробу.

Вимоги до рекламного компонента. Рекламний банер не повинен впливати на швидкодію або стабільність застосунку. Його завантаження має відбуватися у фоновому режимі, без впливу на основний функціонал. У випадку, якщо рекламний контент недоступний, це не повинно впливати на інтерфейс - система

повинна або приховувати банер, або коректно відображати пустий стан без візуальних артефактів. Рекламу не повинна заважати взаємодії з основним контентом або маскуватися під нього. Користувач має чітко розуміти, що це окремий елемент, і взаємодія з ним є повністю добровільною.

Вимоги до енергоефективності. Застосунок не повинен виконувати зайві операції у фоновому режимі. Усі ресурсоємні процеси мають запускатися тільки за необхідності. Не допускається постійна активність, яка може призводити до зайвого споживання заряду батареї. Застосунок не має включати фоновий сервіс, який може негативно вплинути на енергонезалежність пристрою.

Таким чином, система повинна залишатися швидкою, стабільною і передбачуваною в роботі.

1.5 Вибір технологій побудови системи

Для розробки програмної частини системи обрано мову програмування Java [27] з урахуванням набутого під час навчання досвіду та особливостей платформи Android. У порівнянні з Kotlin, Java має більш явну модель роботи з потоками, пам'яттю та життєвим циклом компонентів. Kotlin, хоча і скорочує обсяг коду, фактично додає рівень абстракції (особливо через coroutines), що ускладнює відслідковування виконання в багатопоточних системах. В нашій системі, де активно використовуються асинхронні операції (завантаження даних, зображень), важливо мати явний контроль над потоками і це простіше реалізувати за допомогою Java.

Інтерфейс будується на основі Fragment API [28] (`androidx.fragment.app.Fragment`). Це дозволяє:

- ізолювати логіку UI-компонентів;
- використовувати `FragmentManager` для керування стеком викликів;
- коректно працювати з життєвим циклом (`onCreateView`, `onDestroyView`);

- зберігати стан через `onSaveInstanceState()`.

Для передачі параметрів між частинами інтерфейсу використовується сховище в `MainActivity` як головному контролері застосунку. Для відображення списків застосовується:

- `RecyclerView` (`androidx.recyclerview.widget.RecyclerView`) [28];
- `ViewHolder` pattern для оптимізації перевикористання елементів;
- `Adapter` для зв'язування даних із UI.

Це критично для продуктивності при відображенні списків. Для асинхронного завантаження і обробки зображень використовується бібліотека `Glide` [29]. Вона дозволяє здійснювати:

- асинхронне завантаження, тобто внутрішній `thread pool`;
- автоматичне кешування;
- `downsampling`, тобто зменшення розміру `bitmap` під розмір `View`;
- прив'язку до `Fragment/Activity`;
- обробку помилок.

Для багатопоточності використовується модель `Thread` для виконання фонових задач та `Handler` для взаємодії з UI-потокom [28]. Тобто запускаємо фонову операцію, наприклад, фільтрацію даних з бази, далі обробляємо результат та передаємо його в `MainThread` через `Handler`. Це забезпечує: неблокування UI `thread` та контрольований життєвий цикл задач.

Для відкриття зовнішніх посилань використовується стандартний Android механізм `Intent` (`ACTION_VIEW`) та `URI` (`https://...`) [28]. Це гарантує:

- делегування обробки посилань системі;
- відкриття в браузері, наприклад, `Google Chrome`;
- відсутність необхідності реалізовувати `WebView`, що істотно зменшує складність.

Робота з UI та ресурсами відбувається за допомогою:

- `XML layout`-файли для опису інтерфейсу;
- `FrameLayout` для гнучкого позиціонування;
- `Drawable` ресурсів;

- Strings.xml для локалізації застосунку.

Так забезпечується розділення логіки і представлення та простоту підтримки UI. Для збереження стану та уникнення повторних запитів використовуються методи [28]:

- onSaveInstanceState();
- Bundle;
- локальні змінні у Fragment.

Для обробки помилок та забезпечення стабільності застосунку використовуються звичайні try/catch для критичних операцій, null-check перед використанням даних та fallback UI [28].

Реалізація банера передбачає окремий View-компонент у layout, асинхронне завантаження контенту та обробку кліку через Intent (перехід у браузер). Цей банер не блокує UI thread, не впливає на RecyclerView performance та може бути відключений без зміни логіки системи.

Таким чином, використання Java та Android SDK є хорошим вибором для реалізації застосунку.

1.6 Вибір системи керування базою даних

На цьому етапі важливо визначити не просто конкретну СКБД, а сам підхід до зберігання даних. У даному випадку система не передбачає складної серверної взаємодії, синхронізації між користувачами або постійного оновлення контенту з віддалених джерел. Вся логіка побудована навколо локального доступу до набору даних із можливістю швидкого фільтрування. Тому використання зовнішньої СКБД (через API або хмарні сервіси) виглядає надмірним. Зовнішні рішення додають купу складності: необхідність підтримки бекенду, обробки мережових затримок, роботи з помилками з'єднання, а також залежність від інтернету. У нашому випадку це суперечить одній із ключових вимог - можливості

повноцінної роботи офлайн. Крім того, більшість операцій у системі - це читання і фільтрація, а не часті запис або синхронізація, що підсилює доцільність локального зберігання. Отже, логічно обрати «кишенькову» систему керування базами даних, яка працює безпосередньо в межах застосунку і не потребує окремого серверного шару. У цьому класі рішень найочевиднішими кандидатами є SQLite [30] та Realm [31].

Якщо порівнювати ці підходи, то Realm пропонує більш «високорівневу» модель роботи з даними: об'єктно-орієнтований доступ, автоматичну синхронізацію об'єктів, відсутність явного SQL. Це зручно з точки зору швидкого старту, але має свою ціну - додаткова залежність, менш прозорий контроль над виконанням запитів і специфічна модель роботи з потоками. У складніших сценаріях це може створювати неочевидну поведінку. SQLite, навпаки, працює ближче до заліза: це реляційна СКБД з повноцінною підтримкою SQL, чіткою структурою таблиць і явним контролем над запитами. У контексті даного проєкту це важливо, оскільки: структура даних вже чітко нормалізована (зв'язки між категоріями, жанрами, видавцями), фільтрація напряму відображається у SQL-запити, поведінка системи легко прогнозується.

В роботі ще варто розглянути використання ORM-рівня, зокрема Room [32], який є офіційним рішенням від Google для роботи з SQLite. Room спрощує доступ до бази через анотації і генерацію коду, прибирає необхідність писати «сирий» SQL у багатьох випадках і автоматично перевіряє запити на етапі компіляції. Проте у даному випадку використання Room не дає суттєвої переваги. Причина в тому, що структура запитів досить проста і добре контролюється вручну. Додавання ORM створює додатковий рівень абстракції, який: ускладнює трасування запитів та приховує частину логіки генерації SQL. А ще з'явиться купа boilerplate-коду. Фактично, замість спрощення можна отримати зворотний ефект: зростання складності без реального виграшу.

Додатковим аргументом на користь SQLite є характер майбутніх запитів до даних. Навіть без фіксації остаточної структури бази даних вже зрозуміло, що система працюватиме з кількома взаємопов'язаними сутностями: ігри, жанри,

категорії, видавці, тощо. Це означає, що основні операції будуть включати не просто вибірку з однієї таблиці, а комбіновані запити з умовами фільтрації. Це будуть запити типу вибір ігор за категорією + жанром, додаткова фільтрація за роком (діапазон значень), врахування булевих параметрів (offline / multiplayer), підключення додаткових даних через зв'язки (наприклад, видавець або рейтинг). Тобто фактично мова йде про класичні SQL-конструкції: JOIN, WHERE, BETWEEN, комбінування умов через AND/OR. У таких сценаріях SQLite дає повний контроль над виконанням запиту і дозволяє оптимізувати його під конкретну логіку фільтрації.

Важлива перевага SQLite - можливість індексації [29]. Навіть якщо зараз обсяг даних невеликий, у перспективі можна додати індекси на поля, які будуть найчастіше використовуватись у фільтрах. Це напряму впливає на швидкодію і добре підтримується саме реляційними СКБД, такими як SQLite. У випадку з Realm або ORM-рівнем подібна оптимізація обмежена.

Структура даних передбачає наявність зовнішніх ключів і цілісності зв'язків. SQLite підтримує foreign key constraints з відповідними режимами CASCADE та SET NULL, що дозволяє гарантувати коректність даних без додаткової логіки на рівні коду. У випадку об'єктних баз або ORM це часто переноситься у програмну логіку, що підвищує ризик помилок.

SQLite дозволяє працювати з частковими вибірками, тобто отримувати тільки ті поля, які реально потрібні для відображення списку (наприклад, назва гри та рік без витягування всього запису). Це зменшує навантаження на пам'ять і прискорює відображення списків.

Таким чином, на рівні майбутньої логіки фільтрації і відображення даних система добре лягає на реляційну модель. SQLite тут виглядає як технологія, що максимально точно відповідає запитам, що будуть виконуватись у застосунку. Також SQLite повністю відповідає вимозі офлайн-роботи, не створює зайвих залежностей та добре інтегрується з Android SDK з під коробки. Таким чином, обираємо саме SQLite.

1.7 Висновки по розділу

У результаті проведеного аналізу предметної області та існуючих рішень було сформовано уявлення про задачу пошуку відеоігор і визначено основні принципи побудови майбутньої системи.

Детальний розгляд класифікації відеоігор показав, що ігровий контент має складну багаторівневу структуру. Ігри відрізняються не лише за жанрами, а й за платформами, режимами гри, видавцями та віковими обмеженнями. Це безпосередньо впливає на проєктування системи: дані не можуть бути представлені у спрощеному вигляді, а повинні зберігатися як структурований набір взаємопов'язаних даних. Відповідно, пошук у такій системі повинен базуватися на комбінуванні кількох критеріїв одночасно.

Аналіз існуючих систем показав, що сучасні рішення не орієнтовані на чистий пошук. Більшість платформ інтегрують комерційні механізми (продажі, підписки, рекомендації), що впливають на результати і ускладнюють взаємодію. Інтерфейси перевантажені, а логіка пошуку часто підміняється алгоритмами рекомендацій. Крім того, відсутня універсальність: системи або прив'язані до конкретної платформи, або не охоплюють усі типи ігор (особливо настільні). Це підтверджує доцільність розробки окремого рішення, орієнтованого саме на задачу фільтраційного пошуку.

На основі цього аналізу були сформовані функціональні вимоги. Система повинна реалізувати чіткий сценарій роботи: від вибору категорії ігор і формування запиту через набір фільтрів до отримання результатів і перегляду детальної інформації. Важливо, що пошук має бути гнучким (часткове задання параметрів) і логічно коректним (відповідність результатів заданим умовам). Окремо визначено принципову відмову від реалізації функцій купівлі, що дозволяє зосередитися саме на інформаційній складовій.

Нефункціональні вимоги уточнюють якісні характеристики системи. Важливими є висока швидкодія при роботі з локальними даними, коректна робота

в офлайн-режимі та адаптивність до різних пристроїв. Особливо важливим є акцент на офлайн-доступі, що визначає подальші архітектурні рішення (локальне зберігання даних, відсутність залежності від мережі).

Таким чином, обрані технології формують технічну основу реалізації. Використання Java забезпечує контроль над виконанням і багатопоточністю, що критично для роботи з UI та асинхронними операціями. Fragment-орієнтована архітектура дозволяє розробити компоненти інтерфейсу і керувати їх життєвим циклом. RecyclerView забезпечує ефективну роботу зі списками. Бібліотека Glide - оптимізоване завантаження зображень. Звичайна модель Thread + Handler дає можливість чітко розділити UI-потік та фонові задачі.

2 ПРОЄКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ПОШУКУ ІГОР

2.1 Апаратні вимоги системи

Апаратні вимоги визначають мінімальну та рекомендовану конфігурацію обчислювального середовища, у якому клієнтський мобільний застосунок функціонує стабільно, забезпечує прийнятний час відгуку інтерфейсу, коректну роботу локальної бази даних та мережевих запитів до сайтів з іграми. Система реалізується як Android-застосунок на Java з локальним зберіганням на SQLite, асинхронною взаємодією з мережею та обробкою графічних ресурсів, що накладає конкретні вимоги на CPU, RAM, підсистему зберігання, графіку та мережеві інтерфейси.

Обчислювальні характеристики процесора визначаються задачами, які ставить система: виконання логіки на Java з віртуалізацією ART, робота з SQLite (індекси, JOIN, фільтрація), серіалізація/десеріалізація, обробка зображень (декодування, масштабування) та мережеві виклики. Мінімально необхідним є процесор класу ARMv8-A з 4 ядрами Cortex-A53 або еквівалентними із частотою від 1.6 ГГц. Така конфігурація забезпечує прийнятний час виконання CRUD-

операцій у SQLite, а також асинхронне виконання мережових запитів без блокування UI-потoku. Рекомендованою є конфігурація 8 ядер big.LITTLE, наприклад Cortex-A55 + Cortex-A76, з частотою 2.0+ ГГц. Перевага багатоядерності проявляється при паралельній обробці запитів, кешуванні та декодуванні зображень, виконанні складніших SQL-запитів. Важливим є наявність 64-бітної архітектури, оскільки сучасні версії Android оптимізовані під ARM64 (ART), що позитивно впливає на продуктивність системи в цілому.

Оперативна пам'ять визначає здатність системи утримувати активні фрагменти, кеш зображень, буфери мережових відповідей, внутрішні структури SQLite. Коли ми згортаємо застосунок а потім повертаємо його на екран знову, то в разі браку оперативної пам'яті дані почнуть перезавантажуватися з накопичувача і користувач це легко помітить та отримає негативний досвід. Тому Мінімально допустимий обсяг - 4 ГБ. При цьому буде агресивна робота Low Memory Killer, ризик перезавантаження Activity, обмеження на кешування зображень. Рекомендований обсяг - 6 ГБ і більше. Це дозволяє використовувати in-memory кеші, уникати частоті зборки сміття, забезпечити багатозадачність.

Наша система використовує внутрішню пам'ять пристрою для зберігання SQLite-бази, кешування зображень та збереження конфігурацій застосунок. Мінімально необхідно 500 МБ вільного простору для коректної інсталяції та роботи. Рекомендовано більше 2 ГБ вільного простору. Для швидкої роботи SQLite рекомендовано накопичувач UFS 2.1+. Це дозволить швидше відкривати базу та виконувати запитів. SQLite активно використовує дискові операції тому це істотно впливає на користувацький досвід.

Застосунок системи не є графічно інтенсивним, тобто без 3D, але GPU використовується для рендерингу UI (RecyclerView та анімації), апаратного прискорення Canvas, обробки зображень. Мінімально необхідний GPU рівня Adreno 506 / Mali-G52. Рекомендовано Adreno 618+ / Mali-G57+. Головний фактор - підтримка апаратного прискорення, що зменшує навантаження на CPU та покращує плавність особливо при проскроюванні списку ігор.

Вимоги до дисплею визначаються адаптивною версткою. Мінімально

необхідна роздільна здатність 1280×720 , діагональ $\geq 4.7''$. Рекомендовано Full HD або вище та щільність ≥ 300 ppi. Це забезпечує комфортне використання застосунку системи.

Система завантажує постери з Інтернет та переходить на сторінку з сайтом гри, тому швидкісна мережа є важливою. Мінімально необхідно 3G з пропускнуою здатністю ≥ 1 Мбіт/с. Рекомендовано 4G LTE або Wi-Fi 802.11n/ac/ax.

Обчислювальні та мережеві операції впливають на енергоспоживання, тому вимоги до батареї такі: мінімально від 3000 мАгод, рекомендовано від 4000 мАгод.

Ці параметри забезпечують стабільну роботу застосунку, коректну взаємодію з SQLite та обробку мережевих запитів.

2.2 Проєктування архітектури системи

При розробці архітектури системи пошуку ігор головний акцент було зроблено на автономність та швидкість відгуку інтерфейсу. Ми свідомо відмовляємося від використання зовнішнього сервера та складних API-інтерфейсів для отримання списку ігор. Це рішення обґрунтоване в такий спосіб:

- дозволяє додатку працювати стабільно навіть при слабкому інтернет-з'єднанні оскільки вся логіка фільтрації відбувається локально;
- повністю усуває витрати на підтримку серверної інфраструктури;
- прискорює розробку системи.

Основна база даних реалізована на основі SQLite, яка є нативним і високоефективним рішенням для Android. Наповнення бази та додавання нових ігрових проєктів відбувається через оновлення самого застосунку в Google Play. Кожного разу коли виходить нова версія застосунку метод onUpgrade контролера даних додає новий контент в базу та актуалізує перелік ігор без необхідності реєстрації чи постійної синхронізації з хмарою.

Архітектура застосунку будується за принципом Single Page Application, де MainActivity виступає в ролі центрального контейнера та координатора. Це підхід, який спрощує керування життєвим циклом програми та забезпечує переходи між екранами, які для користувача виглядають безшовно ніби на одному екрані. Архітектура складається з таких рівнів:

- presentation layer (UI): Activity + Fragments;
- data layer: SQLite + Database Controller;
- model layer: POJO-класи (сутності).

Основна робота з інтерфейсом розподілена між фрагментами: від головного меню та екранів налаштування фільтрів до списку результатів і детальної картки гри. Використання фрагментів дозволяє гнучко змінювати контент у межах одного FrameLayout, зберігаючи при цьому стан інтерфейсу та забезпечуючи коректну роботу кнопки «Назад». Це наближено до спрощеної MVC-моделі без використання сторонніх фреймворків.

Логіка бази даних проектується в класі DBController, що вміщує всі SQL-запити. Він відповідає за ініціалізацію бази, вибірку даних за складними критеріями та перетворення сирих рядків із таблиць у зрозумілі для Java-коду об'єкти. Для цього спроектуємо класи моделі, які повністю мають відповідати структурі бази даних. Це забезпечить строгу типізацію та полегшує підтримку коду, тому що кожна сутність у базі має своє програмне відображення. Зв'язок між цими моделями та візуальними елементами списку має забезпечити спеціалізований адаптер, який відповідає за наповнення RecyclerView та оптимізацію використання пам'яті при гортанні великих списків.

Логіка взаємодії компонентів системи наступна: користувач відкриває категорію та переходить на відповідний Fragment, Fragment викликає Database Controller, Controller виконує SQL-запит, результат мапиться у список Game, дані передаються в Adapter, adapter відображає список, клік призводить до відкриття детального Fragment, на якому можливе завантаження зображення та відкриття URL.

Застосунок активно взаємодіє з мережею для покращення користувацького

досвіду. Оскільки вшивати важкі графічні файли у дистрибутив непрактично, то у системі реалізовано асинхронне завантаження постерів ігор за прямими посиланнями. Використання сторонніх бібліотек дозволяє кешувати ці зображення, щоб при повторному перегляді вони з'являлися миттєво. Мережевий доступ також необхідний для зовнішньої навігації: при виборі конкретної платформи додаток формує Intent і перенаправляє користувача до браузера або офіційних магазинів для PC, XBOX, Playstation та Google Play.

2.3 Проєктування варіантів використання

Користувацький досвід у нашому застосунку спроектований таким чином, щоб мінімізувати кількість зайвих дій та надати швидкий доступ до інформації про ігри. Use-case діаграма наведена на рис. 2.1.

Вся взаємодія базується на переході від загального огляду до конкретних деталей, що забезпечує простоту інтерфейсу. Перше, з чим стикається користувач - це головне меню. Воно виконує роль навігаційного пункту. Тут користувач має можливість обрати глобальну категорію ігор: десктопні, мобільні або настільні. Цей поділ є важливим, оскільки кожна категорія має свій унікальний набір параметрів фільтрації та дизайн. Користувач не перевантажений усіма можливими опціями одразу тому що система підлаштовує інтерфейс під обраний тип ігор, наприклад, параметри платформи Xbox для настільних ігор демонструватися не будуть.

Важливим етапом є процес фільтрації. Користувач отримує гнучкий інструмент для підбору гри за різними наборами параметрів: категорія, жанр, віковий рейтинг, рік випуску, видавець, доступність на платформах, режими гри: одиночний чи багатокористувацький.

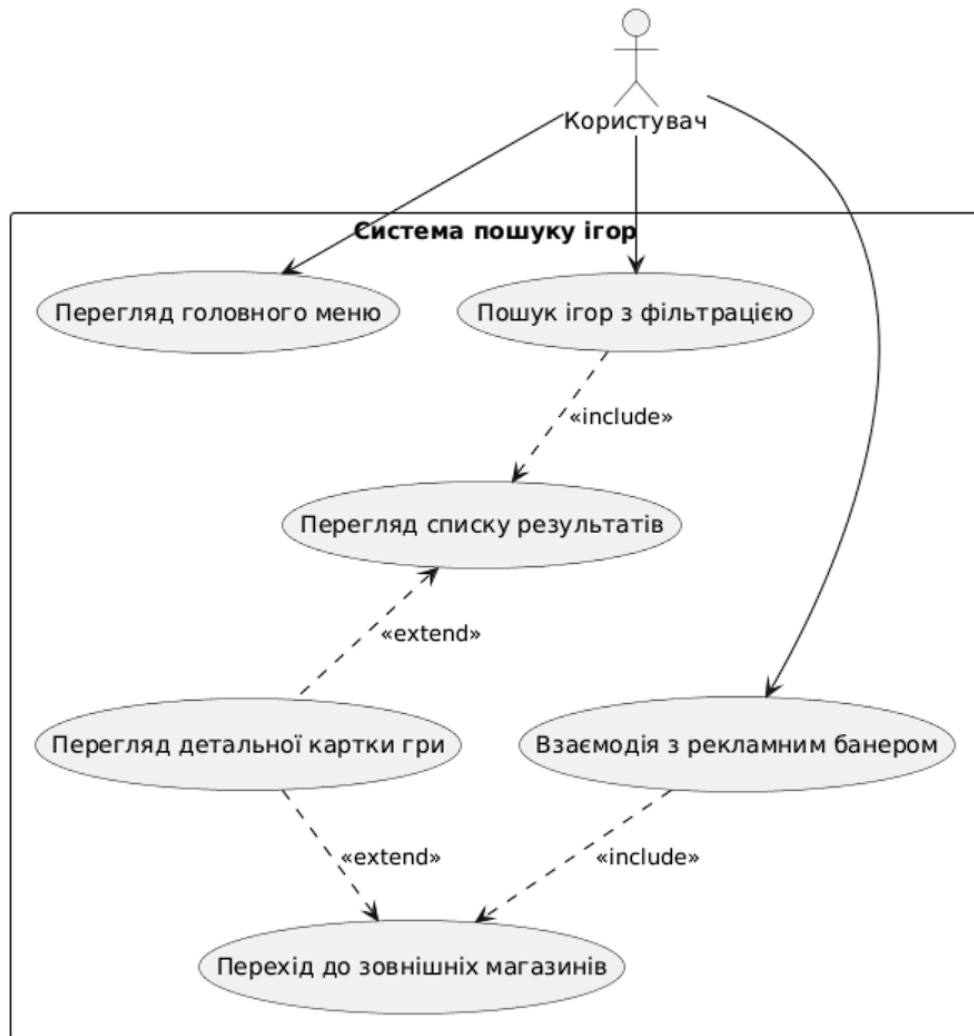


Рисунок 2.1 – Use-case діаграма застосунку

Фільтри мають бути інтерактивними: якщо користувач відмічає чекбокс, система активує відповідний випадаючий список або поле введення. Це запобігає випадковим помилкам при введенні даних. У разі, якщо користувач задав занадто суворі критерії, за якими в базі SQLite немає жодного збігу, система не просто показує пустий екран, а видає миттєве сповіщення. Це дозволяє користувачу оперативно скоригувати параметри пошуку, не виходячи з поточного фрагмента.

Після підтвердження запиту користувач переходить до динамічного списку ігор. Кожна позиція у списку - це коротка карточка гри з назвою, роком випуску та постером. Користувач може швидко прокручувати список, отримуючи візуальне підтвердження свого запиту. Тут же має бути можливість взаємодії з рекламним контентом. Рекламний банер має бути розміщено у нижній частину

екрана не перекриваючи основний контент і надаючи користувачу можливість одним натисканням перейти до партнерського магазину.

Кінцевою точкою внутрішньої навігації є екран деталей гри. Коли користувач обирає цікавий проєкт, він отримує детальну інформацію: жанр, рік, видавець, віковий рейтинг, кнопки для переходу на магазини. Весь цей контент підтягується локально з бази даних, що забезпечує миттєве відкриття сторінки. Якщо гра доступна на кількох платформах, користувач бачить відповідні кнопки-посилання. Натискання на таку кнопку активує зовнішній Intent, і система автоматично перенаправляє користувача на офіційну сторінку гри в Інтернет або у відповідний додаток магазину на пристрої. Таким чином, користувач проходить повний шлях: від бажання знайти щось цікаве до конкретного переходу до завантаження чи покупки.

В діаграмі суцільні стрілки від актора «Користувач» до варіантів використання відображають взаємодію користувача із системою. Користувач явно запускає перегляд головного меню, виконання пошуку або взаємодію з рекламним банером. Такий тип зв'язку використовується для фіксації точок входу в систему, де саме користувач є ініціатором дії. Пунктирні стрілки із стереотипом <<include>> застосовуються для моделювання обов'язкових підпроцесів, які завжди виконуються в межах основного сценарію. Це зв'язок між «Пошук ігор з фільтрацією» та «Перегляд списку результатів», оскільки будь-який пошук логічно завершується відображенням результатів. Аналогічно, взаємодія з рекламним банером включає перехід до зовнішнього ресурсу, тому використовується <<include>> відображає невід'ємність цієї дії при натисканні на банер. Зв'язки типу <<extend>> використовуються для опису додаткової поведінки, яка не є обов'язковою і виконується лише за певних умов. У застосунку це відображено у переході від «Перегляд списку результатів» до «Перегляд детальної картки гри», оскільки відкриття деталей відбувається лише у випадку, якщо користувач обирає конкретний елемент зі списку. Аналогічно, перехід до зовнішніх магазинів є розширенням сценарію перегляду деталей гри.

2.4 Проєктування бази даних

В ході проєктування було виділено основні сутності, а саме: категорії, жанри, видавці, країни, вікові рейтинги, ігри. Ці сутності було розміщено в базі даних, DBML-опис якої наведено в лістингу 2.1. З врахуванням використання SQLite булеві типи даних представлено у вигляді integer з інтерпретацією 1 як true та 0 як false. Крім того первинні ключі позначено як `_id` для коректної інтерпретації рушієм SQLite. Зв'язки між таблицями забезпечено за допомогою зовнішніх ключів. Таким чином, наша база даних знаходиться в третій нормальній формі та може бути використана в проєкті.

В основі системи стоїть таблиця Games, яка виконує роль консолідатора всієї інформації. Вона не зберігає текстові назви категорій чи жанрів безпосередньо, а використовує ідентифікатори, що забезпечує цілісність даних та дозволяє уникнути дублювання. Поля `idCategory`, `idGenre`, `idPublisher` та `idAgeRating` потрібні для реалізації логіки додатку: за ними відбувається первинна вибірка ігор у DBController, коли користувач обирає відповідні параметри у випадуючих списках.

Поля `isOffline` та `isMultiplayer` реалізовано як булеві прапорці, що дозволяє миттєво відфільтрувати ігри за типом взаємодії. Тільки на SQLite вони будуть типу integer, адже ця кишенькова СКБД не має прямої підтримки мулевих типів та трактує 1 як true та 0 як false. Блок полів `urlPC`, `urlXBOX`, `urlPS` та `urlLMobile` важливими для реалізації зовнішньої взаємодії: наявність або відсутність посилання в цих колонках не лише дозволяє користувачеві перейти в магазин, а й слугує додатковим фільтром, наприклад, для відображення тільки тих ігор, що доступні на PlayStation. Поле `imageUrl` зберігає лише шлях до файлу в Інтернет для асинхронного завантаження контенту через мережу. Тобто самі зображення постерів ігор ми в базі зберігати не будемо. ER-модель бази даних наведено на рисунку 2.2.

Лістинг 2.1 - DBML-опис бази даних

```
Table Categories {
  _id integer [primary key]
  name varchar
  description text}

Table Genres {
  _id integer [primary key]
  name varchar
  description text}

Table Publishers {
  _id integer [primary key]
  name varchar
  idCountry integer [ref: > Countries._id]
  website varchar}

Table Countries {
  _id integer [primary key]
  name varchar}

Table AgeRatings {_id integer [primary key]
  rating varchar
  description text}

Table Games {_id integer [primary key]
  title varchar
  idCategory integer [ref: > Categories._id]
  idGenre integer [ref: > Genres._id]
  description text
  imageURL varchar
  year integer
  idPublisher integer [ref: > Publishers._id]
  idAgeRating integer [ref: > AgeRatings._id]
  isOffline boolean
  isMultiplayer boolean
  urlPC varchar
  urlXBOX varchar
  urlPS varchar
  URLMobile varchar}
```



Рисунок 2.2 – ER-модель бази даних

Допоміжні сутності Categories, Genres, Publishers та AgeRatings, спроектовані з урахуванням потреби в деталізації та фільтрації ігор. Окрема таблиця Countries пов'язана з видавцями та дозволяє в майбутньому легко розширити функціонал, наприклад, додати фільтр за країною походження гри. Таблиці містять поле description, що дає можливість зберігати детальний текстовий опис без обмеження атомарності чи кількості символів. Нормалізація бази даних гарантує, що при оновленні назви якогось поля, наприклад, жанру або зміні опису видавця, зміни автоматично відобразяться для всіх пов'язаних ігор, що значно спрощує підтримку бази при випуску нових версій застосунку через Google Play.

2.5 Прототипування системи

Для того, щоб зробити прототип системи потрібно чітко зрозуміти та описати шлях користувача. Користувач взаємодіє із застосунком, починаючи з головного екрана, який виконує роль точки входу в систему та забезпечує вибір сценарію використання. Інтерфейс містить назву застосунку та три основні кнопки, що відповідають категоріям ігор: десктопні, мобільні та настільні. На цьому етапі користувач формує намір пошуку, обираючи відповідний тип ігор. Натискання будь-якої з кнопок ініціює зміну екрану та дає можливість фільтрувати ігри.

Після вибору категорії користувач потрапляє на один із трьох екранів фільтрації, структура яких адаптована під специфіку відповідного типу ігор. У випадку десктопних ігор інтерфейс надає найбільш розширений набір параметрів, включаючи жанр, видавця, рік випуску, рейтинг, а також додаткові характеристики, такі як режим гри (однокористувацький або багатокористувацький) та доступність на платформах (PC, Xbox, PlayStation). Для мобільних ігор набір фільтрів спрощений і орієнтований на жанр, видавця, рік та тип взаємодії (офлайн або мультиплеєр). Це типові сценарії використання мобільних ігор. Для настільних ігор передбачено ще більш компактний набір параметрів, який включає жанр, рік і віковий рейтинг.

Користувач має можливість застосовувати всі доступні фільтри або обирати лише частину з них, використовуючи чекбокси для активації відповідних критеріїв. Це дозволяє гнучко формувати запит до бази даних. Після налаштування параметрів користувач ініціює пошук натисканням кнопки, що призводить до звернення до шару доступу до даних. На цьому етапі формується SQL-запит із урахуванням лише активованих фільтрів, після чого виконується пошук у локальній базі SQLite. У разі відсутності результатів користувач отримує зворотний зв'язок у вигляді короткого повідомлення, що сигналізує про необхідність зміни параметрів пошуку. У випадку успішного знаходження даних

відбувається перехід до наступного екрану зі списком ігор.

Екран результатів представляє собою уніфікований інтерфейс незалежно від обраної категорії. Тільки колір інтерфейсу має співпадати з кольором екрану відповідної категорії ігор. Це забезпечує з'єднання всіх трьох гілок навігації в одну точку. Дані відображаються у вигляді вертикального списку, де кожен елемент містить зображення гри, її назву та рік випуску. Це дозволяє виконати швидкий перегляд результатів і оперативно оцінити релевантність знайдених варіантів. У нижній частині екрана має розташуватися інтерактивний рекламний банер, який є додатковим елементом монетизації. Взаємодія з банером є опціональною і не впливає на основний сценарій використання: у разі натискання має викликатися неявний Intent, що відкриває зовнішній браузер із відповідною сторінкою, після чого користувач може повернутися до застосунку стандартними засобами операційної системи. Основна дія на цьому кроці полягає у виборі конкретної гри зі списку. Натискання на елемент ініціює перехід до екрану з деталями про обрану гру. Цей екран містить розширений набір даних, включаючи зображення, назву, жанр, рік випуску, видавця, рейтинг та текстовий опис. Така деталізація дозволяє користувачу прийняти рішення щодо подальших дій. Додатково передбачено блок із кнопками, що відповідають платформам розповсюдження гри. Видимість цих кнопок визначається динамічно залежно від доступності гри на відповідних платформах. Тобто якщо, наприклад, гра доступна тільки на PC та XBOX, то кнопка PS буде прихована. Якщо гра доступна в Google Play Store, тоді з'явиться відповідна кнопка.

Натискання на кнопку платформи призводить до виходу за межі застосунку через зовнішній браузер. Залежно від типу платформи це може бути перехід до сторінки гри в цифровому магазині або відкриття веб-сторінки у браузері. Таким чином, фінальний екран виконує роль точки прийняття рішення.

З кожного екрану можна повернутися на попередній натискаючи назад. Всі переходи мають супроводжуватися анімацією. Так користувацький досвід буде значно приємнішим. Прототип системи наведено на рис. 2.3. Він чітко демонструє екрани, елементи інтерфейсу та дії, що призводять до змін екрану.

Загалом, користувацький шлях дає чітку послідовність переходів та налаштування фільтрів, після чого всі сценарії сходяться до перегляду результатів і деталізації вибраного об'єкта. Така структура забезпечує баланс між гнучкістю пошуку та простотою навігації, тому графічний інтерфейс будемо верстати чітко у відповідності до цієї структури та прототипу (рис. 2.3).

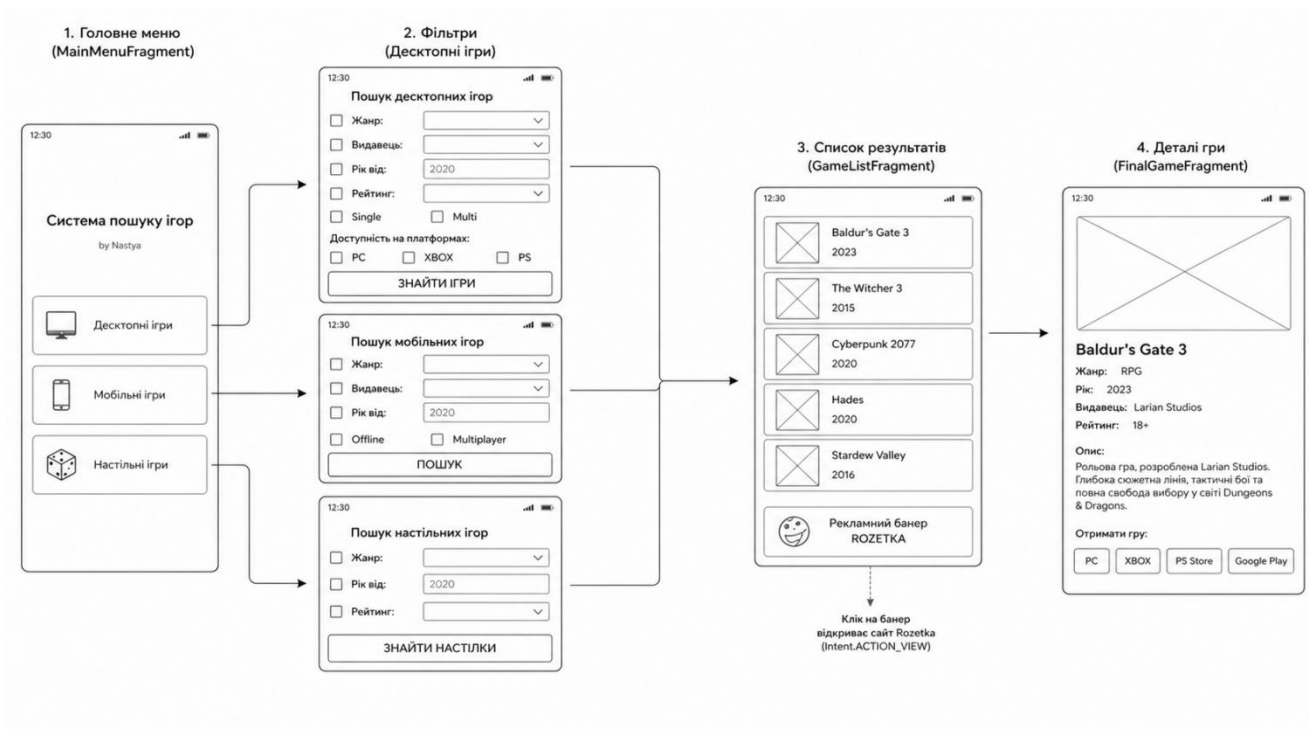


Рисунок 2.3 – Прототип системи

2.6 Проєктування компонентів системи

Маючи прототип системи та чіткий опис варіантів використання можна переходити до проєктування компонентів системи. Розпочнемо з діаграми класів, яку наведено на рис. 2.4. MainActivity це головне вікно, що ініціалізує DBController та керує життєвим циклом усіх фрагментів. Це головний контролер та плейсхолдер, на якому змінюються екрани. Фрагменти керують інтерфейсом та навігацією. MainMenuFragment - це стартовий екран, який відповідає за вибір

глобальної категорії ігор: десктопні, мобільні, настільні. `ComputerGameFragment`, `MobileGameFragment`, `BoardGameFragment` – це фрагменти-фільтри. Кожен має свою унікальну логіку збору даних зі `Spinner`-ів та `CheckBox`-ів для формування запиту в базу. `GameListFragment`: Відображає результати пошуку у вигляді списку та містить логіку рекламного банера. `GameDetailsFragment`: Картка конкретної гри. Отримує об'єкт `Game` і відображає його деталі, завантажуючи зображення через `Glide`. Робота з даними виконується завдяки класу `DBController`. Він виконує SQL-запити, фільтрацію та конвертацію рядків бази даних у Java-об'єкти. `GameAdapter` - це головний адаптер для `RecyclerView`, тобто для коректного відображення списку фільтрованих ігор. Він поєднує дані зі списку об'єктів `Game` із візуальними елементами картки в `GameListFragment`. Класи моделі повністю повторюють структуру таблиць бази даних:

- `Game` - основний об'єкт гри (назва, опис, рік, посилання, id зв'язків);
- `Genre` - жанри ігор (`Action`, `Strategy` тощо);
- `Publisher` – описує видавців ігор;
- `Country` - країни, що пов'язані з видавцями ігор;
- `Category` - глобальні типи ігор;
- `AgeRating` - вікові обмеження та рекомендації до аудиторії.

Структура полів класів моделі повністю відповідає структурі в базі даних.

Для створення діаграми послідовностей змін екранів (рис. 2.5) сплануємо шляхи до зміни фрагментів на екрані. Після запуску застосунку ініціалізується головна активність `MainActivity`, яка створює екземпляр `DBController` та одразу встановлює стартовий екран у вигляді `MainMenuFragment` через `replaceFragment`. Далі вся навігація будується через взаємодію користувача з елементами інтерфейсу головного меню. Натискання кнопки «Десктопні» у `MainMenuFragment` ініціює виклик методу `replaceFragment` у `MainActivity` з передачею `ComputerGameFragment`, після чого поточний інтерфейс замінюється на екран фільтрації комп'ютерних ігор. Аналогічно, вибір кнопки «Мобільні» призводить до переходу на `MobileGameFragment`, а кнопка «Настільні» - до `BoardGameFragment`, кожен з яких відповідає окремій категорії ігор та має

власний набір фільтрів.

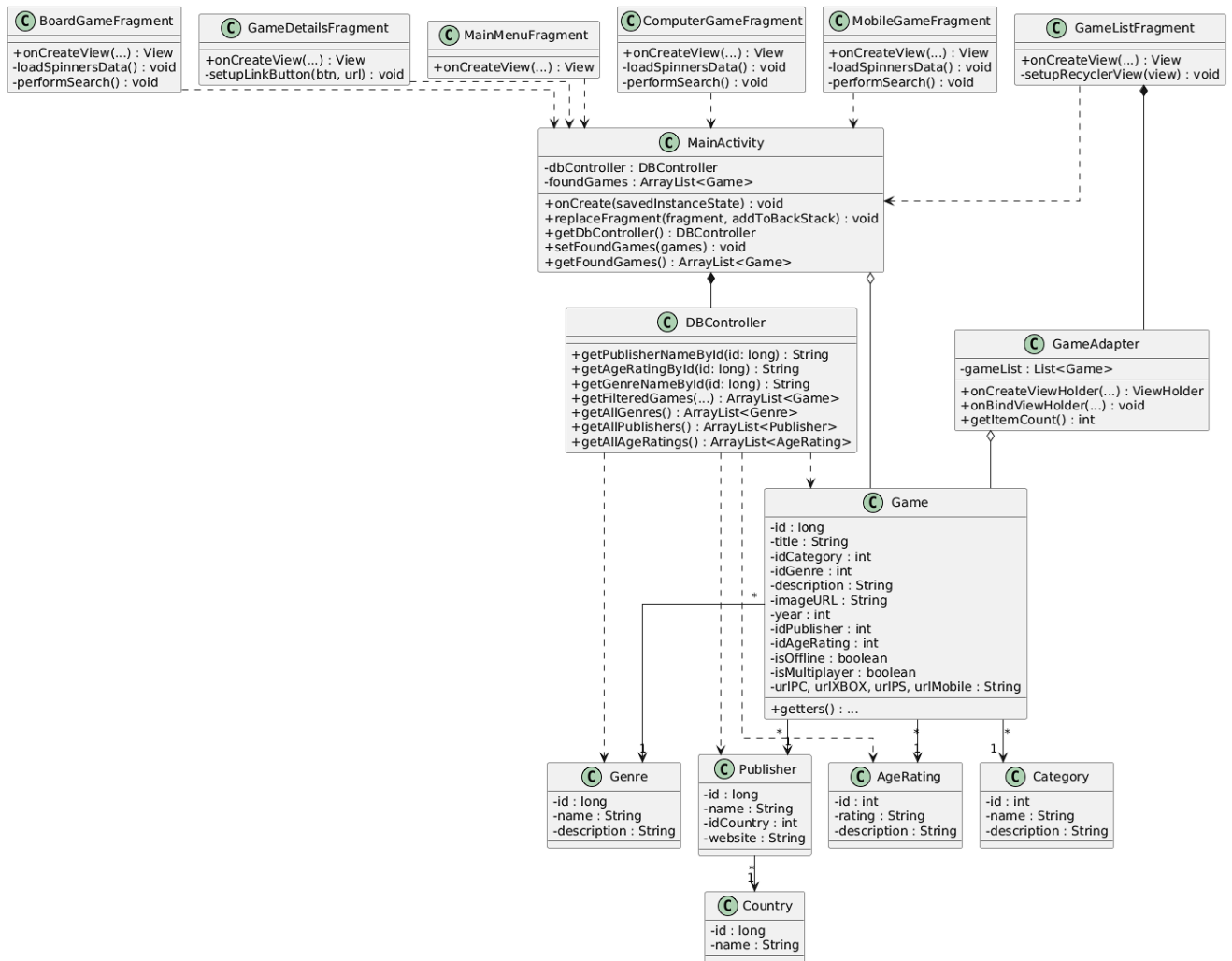


Рисунок 2.4 – Діаграма класів системи

У кожному з цих фрагментів користувач задає параметри пошуку та натискає відповідну кнопку пошуку, що запускає формування запиту до бази даних через DBController. Отриманий список ігор зберігається у MainActivity через setFoundGames, після чого відбувається програмний перехід на GameListFragment. Таким чином GameListFragment завжди відкривається як результат виконання пошуку незалежно від того, з якого саме категорійного фрагмента він був викликаний.

Після відображення списку ігор користувач може взаємодіяти з елементами RecyclerView. Вибір конкретної гри ініціює перехід до GameDetailsFragment, який створюється в MainActivity і отримує об'єкт гри через Bundle. У цьому фрагменті

відображається повна інформація про вибраний запис, включно з описом, роком, жанром, видавцем та рейтингом, які додатково підтягуються з бази даних через DBController.

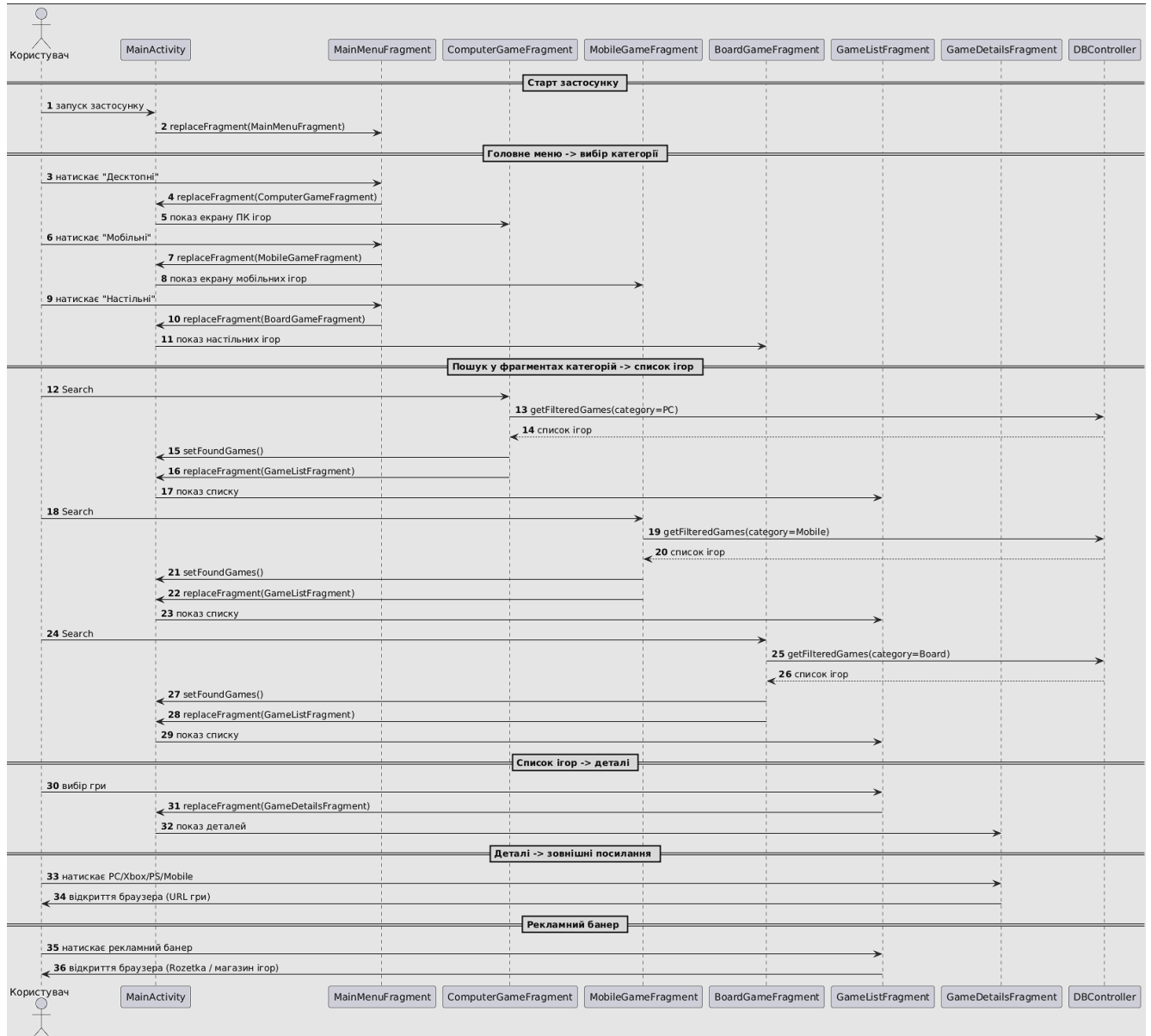


Рисунок 2.5 – Діаграма послідовності зміни екранів

Окремо організовується механізм зовнішніх переходів із GameDetailsFragment: натискання кнопок платформ (PC, Xbox, PlayStation або Mobile) відкриває відповідні URL у зовнішньому браузері через Intent, не змінюючи поточний стек фрагментів. Аналогічний принцип використовується для рекламного банера, який розміщений у GameListFragment: при натисканні на

нього виконується відкриття зовнішнього вебсайту магазину ігор у браузері.

Діаграма діяльності пошуку гри наведена на рис. 2.6.

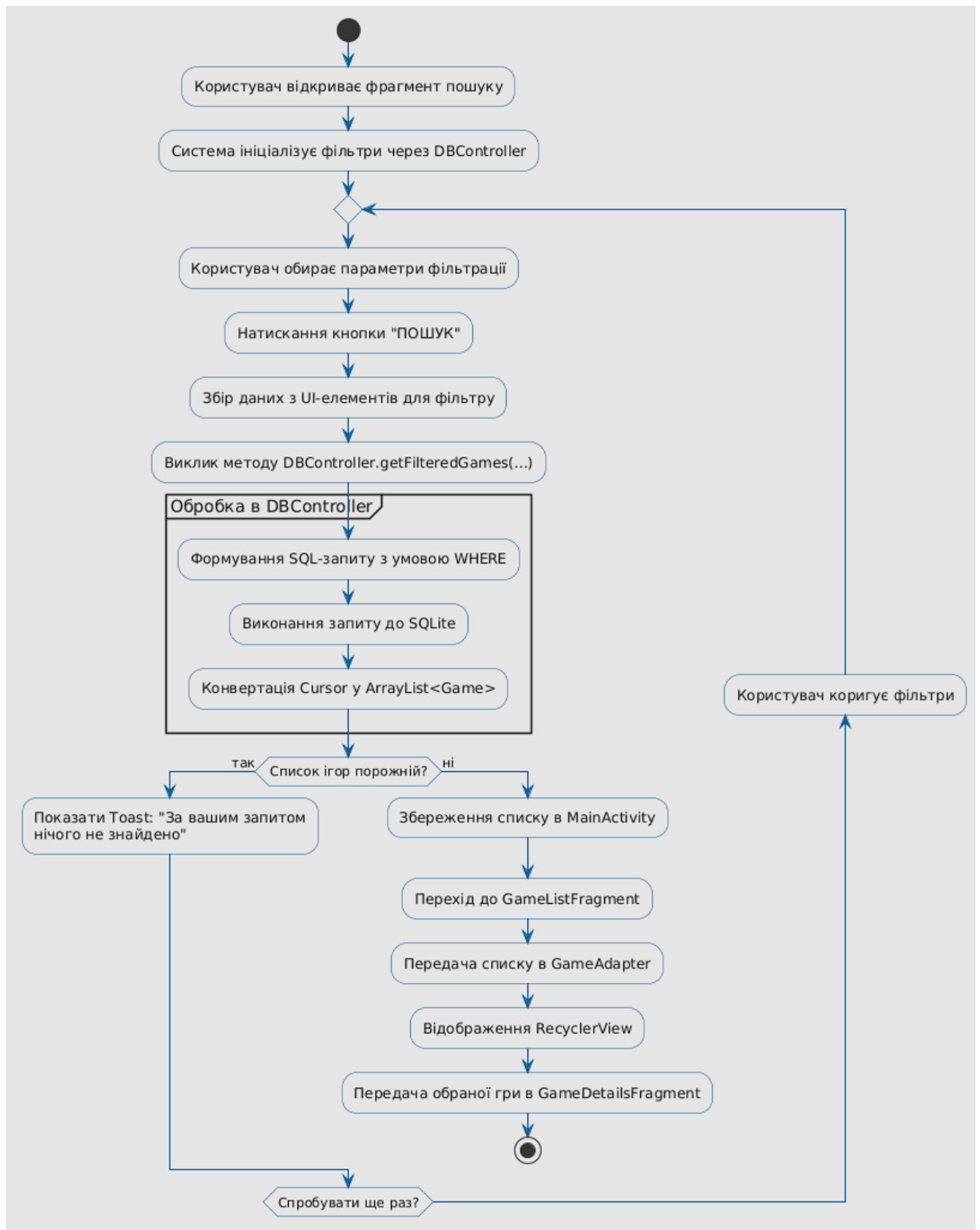


Рисунок 2.6 – Діаграма діяльності пошуку гри

Після відкриття фрагмента пошуку відбувається початкова ініціалізація стану інтерфейсу - система через DBController підтягує доступні значення для фільтрів (жанри, рейтинги, видавці, тощо). Це формує дані в елементах інтерфейсу для подальшого запиту без додаткових звернень під час кожної ітерації. Цикл моделює поведінку користувача як послідовність спроб. На кожній ітерації користувач задає або змінює параметри фільтрації, після чого ініціює пошук. Тут UI виступає як джерело параметрів. Далі управління переходить у шар доступу до даних. У межах DBController відбувається динамічне формування SQL-запиту - конструюється WHERE-частина залежно від обраних фільтрів. Запит в базу не є статичним, а адаптується під конкретну комбінацію параметрів та фільтрів. Після виконання запиту результати, отримані у вигляді Cursor, зберігаються в колекції об'єктів Game, що відповідає моделі предметної області і може напряму використовуватися у UI. Якщо вибірка порожня, система не переходить до іншого екрана, а сигналізує про це локально через Toast. Це рішення зберігає контекст і дозволяє користувачу швидко скоригувати параметри без додаткової навігації до порожнього екрану з необхідністю повертатися назад і знову вказувати фільтри. У випадку наявності результатів відбувається передача даних на рівень Activity, яка є контейнером стану. Збережений список використовується для ініціалізації GameListFragment. Передача в адаптер забезпечує зв'язування даних із RecyclerView, після чого відбувається рендеринг списку. Вибір гри зі списку передає її в GameDetailsFragment. Завершення процесу на цьому етапі означає вихід із пошуку гри.

2.7 Висновки по розділу

В розділі виконано повний проєктування комп'ютерної системи пошуку ігор - від визначення технічних обмежень до деталізації внутрішніх компонентів та варіантів взаємодії. Сформульовані апаратні вимоги дозволили зафіксувати

середовище виконання застосунку системи та обґрунтувати мінімальні й рекомендовані характеристики пристроїв. У процесі проєктування архітектури прийнято рішення щодо відмови від серверної частини на користь повністю локальної обробки даних. Такий підхід дозволив мінімізувати залежність від мережі та спростити систему. Архітектура застосунку організована за чітким поділом на presentation, data та model рівні, де MainActivity виконує роль координатора, а DBController інкапсулює всю логіку доступу до даних.

Опис варіантів використання продемонстрував орієнтацію системи на мінімальну кількість дій користувача для фільтрації, перегляду результатів і деталізації гри.

Проєктування бази даних забезпечило структуроване зберігання інформації з дотриманням принципів нормалізації. Використання зовнішніх ключів та розділення сутностей дозволяє уникнути дублювання даних і спрощує подальше розширення системи. Центральна таблиця Games виступає агрегатором, що пов'язує всі довідкові сутності.

Прототипування дозволило формалізувати інтерфейс і послідовність дій користувача ще до реалізації. Це дало змогу узгодити структуру екранів та логіку навігації.

Проєктування компонентів системи завершило формування цілісної моделі застосунку. Визначено ролі всіх класів та механізми їх взаємодії. UML-діаграми підтверджують узгодженість прийнятих рішень.

З РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ПОШУКУ ІГОР

3.1 Розробка бази даних

Почнемо реалізацію застосунку системи з бази даних. Для цього створюємо клас DBController, який спадкує клас SQLiteOpenHelper з пакету android.database.sqlite. Через DBController проходить ініціалізація структури бази,

наповнення бази первинним контентом, операції читання даних. Це дозволяє ізолювати SQL-логіку від UI-рівня. При створенні бази даних у методі onCreate (лістинг 3.1) виконується повний DDL-сценарій.

Лістинг 3.1 – Реалізація методу onCreate

```
public void onCreate(SQLiteDatabase db) {
    db.execSQL("PRAGMA foreign_keys = ON;");
    db.execSQL("CREATE TABLE Countries (_id INTEGER PRIMARY KEY
AUTOINCREMENT, name VARCHAR(100) NOT NULL);");
    db.execSQL("CREATE TABLE Categories (_id INTEGER PRIMARY KEY
AUTOINCREMENT, name VARCHAR(50) NOT NULL, description TEXT);");
    db.execSQL("CREATE TABLE Genres (_id INTEGER PRIMARY KEY
AUTOINCREMENT, name VARCHAR(50) NOT NULL, description TEXT);");
    db.execSQL("CREATE TABLE AgeRatings (_id INTEGER PRIMARY KEY
AUTOINCREMENT, rating VARCHAR(20) NOT NULL, description TEXT);");
    db.execSQL("CREATE TABLE Publishers (_id INTEGER PRIMARY KEY
AUTOINCREMENT, name VARCHAR(100) NOT NULL, idCountry INTEGER,
website VARCHAR(255), FOREIGN KEY (idCountry) REFERENCES
Countries(_id) ON DELETE SET NULL);");
    db.execSQL("CREATE TABLE Games (_id INTEGER PRIMARY KEY
AUTOINCREMENT, title VARCHAR(255) NOT NULL, idCategory INTEGER,
idGenre INTEGER, description TEXT, imageURL TEXT, year INTEGER,
idPublisher INTEGER, idRating INTEGER, isOffline INTEGER DEFAULT 1,
isMultiplayer INTEGER DEFAULT 0, urlPC TEXT, urlXBOX TEXT, urlPS
TEXT, urlMobile TEXT, FOREIGN KEY (idCategory) REFERENCES
Categories(_id) ON DELETE CASCADE, FOREIGN KEY (idGenre) REFERENCES
Genres(_id) ON DELETE SET NULL, FOREIGN KEY (idPublisher) REFERENCES
Publishers(_id) ON DELETE SET NULL, FOREIGN KEY (idRating)
REFERENCES AgeRatings(_id) ON DELETE SET NULL);");
    fillInitialData(db);
}
```

Спочатку примусово вмикається підтримка зовнішніх ключів через PRAGMA foreign_keys = ON, що важливо для забезпечення цілісності даних. Далі створюються таблиці Countries, Categories, Genres, AgeRatings. Таблиця Publishers із зовнішнім ключем на країну видавця. Центральна таблиця Games зберігає основну інформацію та містить декілька зовнішніх ключів, що формують зв'язки з іншими таблицями. Використання ON DELETE CASCADE та ON DELETE SET NULL визначає поведінку при видаленні пов'язаних записів і дозволяє уникнути «битих» посилань. Наприклад, при видаленні видавця Ubisoft, в зв'язаних таблицях має проставитися NULL у відповідному полі, а не видалятися всі записи

ігор, пов'язані з цим видавцем.

Після створення структури таблиць викликається метод `fillInitialData` (лістинг 3.2), де база наповнюється початковими даними. Дані в таблиці вставляються за допомогою інструкції SQL `INSERT`.

Лістинг 3.2 – Фрагмент методу `fillInitialData`

```
private void fillInitialData(SQLiteDatabase db) {
    db.execSQL("INSERT INTO Categories (name, description) VALUES " +
        "('Computer Game', 'Ігри для персональних комп'ютерів та ігрових\n" +
        "консолей (AAA/AA сегмент)'), " + "('Mobile Game', 'Сучасні мобільні\n" +
        "проекти для платформи Android'), " + "('Board Game', 'Популярні\n" +
        "настільні ігри з високим рейтингом BGG');");
    db.execSQL("INSERT INTO AgeRatings (rating, description) VALUES " +
        "('PEGI 3', 'Для будь-якого віку'), " + "('PEGI 12', 'Від 12 років:\n" +
        "помірне насильство'), " + "('PEGI 16', 'Від 16 років: реалістичне\n" +
        "насильство'), " + "('PEGI 18', 'Тільки для дорослих: жорсткий\n" +
        "контент');");
    insertGame(db, "Cyberpunk 2077", 1, 1, "Пригодницька рольова гра,\n" +
        "дія якої відбувається в мегаполісі Найт-Сіті.",
        "https://cdna.artstation.com/p/assets/images/images/033/037/886/medi\n" +
        "um/artur-tarnowski-malemain.jpg?1608208334",
        2020, 1, 4, 1, 0, "https://www.cyberpunk.net/buy",
        "https://www.xbox.com/games/cyberpunk-2077",
        "https://www.playstation.com/games/cyberpunk-2077/", null);...}
```

Дані ігор вставляються в таблицю через `ContentValues`, що забезпечує правильне співставлення полів і знижує ризик помилок при великій кількості параметрів. Наприклад, якщо певна гра не має підтримки Playstation, тоді при вставці туди підставиться `null` і кількість параметрів та порядок параметрів вставки не зламається.

Отримання даних реалізоване через набір методів читання. Базовий варіант - це метод `getAllGames` (лістинг 3.3), який виконує простий `SELECT *` по таблиці з іграми та послідовно проходить `Cursor`, формуючи список об'єктів. Цей

список повертається у вигляді колекції `ArrayList`. Метод `mapGame` перетворює один рядок `Cursor` у об'єкт `Game`, який додається в колекцію. Тобто він створює нову гру опитуючи кожне поле курсору за допомогою `getColumnIndexOrThrow`. При зміні структури таблиці завдяки цьому методу потенційна помилка буде виявлена одразу, а не призведе до тихого некоректного заповнення полів.

Лістинг 3.3 – Метод `getAllGames`

```
public ArrayList<Game> getAllGames() {
    ArrayList<Game> list = new ArrayList<>();
    SQLiteDatabase db = getReadableDatabase();
    Cursor cursor = db.rawQuery("SELECT * FROM Games", null);
    if (cursor.moveToFirst()) {
        do {
            list.add(mapGame(cursor));
        } while (cursor.moveToNext());
    }
    cursor.close();
    return list;
}
```

Аналогічно побудовані методи `getAllGenres`, `getAllPublishers`, `getAllAgeRatings` та інші - вони читають відповідні таблиці в форматі `SELECT *` та повертають дані у вигляді колекцій, які потім використовуються для заповнення елементів графічного інтерфейсу для того, щоб користувач міг виконувати фільтрацію по наявних іграх. Тобто користувач буде бачити тільки ті, наприклад, жанри ігор, які наявні в базі. На мій погляд, так буде краще, щоб не створювати ті фільтри, які будуть призводити до безрезультатного пошуку.

Основна логіка пошуку реалізована в методі `getFilteredGames` (лістинг 3.4). В цьому методі динамічно формується SQL-запит залежно від вибраних користувачем фільтрів. Використовується `StringBuilder` для побудови `WHERE`-частини та окремий список аргументів. Якщо певний фільтр не активний (в метод потрібно передати значення `-1`), він просто не додається в умову. Це дозволяє

уникнути великої кількості жорстко закодованих варіантів запитів і робить систему масштабованою, тому що додавання нового фільтра не ламає існуючу логіку. Використовуються параметризовані запити з ?, що захищає від SQL-ін'єкцій навіть у локальному середовищі.

Лістинг 3.4 – Метод `getFilteredGames`

```
public ArrayList<Game> getFilteredGames(long catId, long genId, long
pubId, long ratId, int offline, int multi) {
    ArrayList<Game> list = new ArrayList<>();
    SQLiteDatabase db = getReadableDatabase();
    StringBuilder query = new StringBuilder("SELECT * FROM Games");
    List<String> conditions = new ArrayList<>();
    List<String> args = new ArrayList<>();
    if (catId != -1) { conditions.add("idCategory = ?");
args.add(String.valueOf(catId)); }
    if (genId != -1) { conditions.add("idGenre = ?");
args.add(String.valueOf(genId)); }
    if (pubId != -1) { conditions.add("idPublisher = ?");
args.add(String.valueOf(pubId)); }
    if (ratId != -1) { conditions.add("idRating = ?");
args.add(String.valueOf(ratId)); }
    if (offline != -1) { conditions.add("isOffline = ?");
args.add(String.valueOf(offline)); }
    if (multi != -1) { conditions.add("isMultiplayer = ?");
args.add(String.valueOf(multi)); }
    if (!conditions.isEmpty()) {
query.append(" WHERE ").append(TextUtils.join(" AND ",conditions));}
    Cursor cursor = db.rawQuery(query.toString(),
args.toArray(new String[0]));
    if (cursor.moveToFirst()) {do {list.add(mapGame(cursor));
        } while (cursor.moveToNext());
    }
    cursor.close();
    return list;}
```

Для наповнення інтерфейсу також знадобляться методи, що виконують вибірку назви по ідентифікатору: `getCategoryNameById` (лістинг 3.5), `getGenreNameById`, `getPublisherNameById` та `getAgeRatingById`.

Лістинг 3.5 – Метод `getCategoryNameById`

```
public String getCategoryNameById(long id) {
    SQLiteDatabase db = getReadableDatabase();
    Cursor cursor = db.rawQuery("SELECT name FROM Categories
WHERE _id = ?", new String[]{String.valueOf(id)});
    String name = "";
    if (cursor.moveToFirst()) {
        name = cursor.getString(0);
    }
    cursor.close();
    return name;
}
```

Ці методи використовуються для перетворення ідентифікаторів у читабельні значення на етапі відображення деталей гри.

Модель даних застосунку представлена класами, що чітко відповідають таблицям в базі: `Category`, `Genre`, `AgeRating`, `Publisher`, `Country`, `Game`. Фрагмент коду класу `Game` (без конструкторів, геттерів та сеттерів) наведено в лістингу 3.6. Цей клас реалізує інтерфейс `Serializable`. Це інтерфейс-маркер без жодного методу, але він дає системі зборки проекту інструкцію про необхідність збирати цей клас особливим чином для того, щоб Android міг прокидувати об'єкт `Game` між фрагментами. Тобто коли користувач обере гру зі списку та захоче відкрити деталі гри чи можна було зробити через `Bundle` фрагментів. Інші класи моделі створено аналогічно з чіткою відповідністю полів тільки без необхідності реалізації інтерфейсу `Serializable`.

Таким чином, `DBController` вирішує кілька задач: створення та ініціалізація структури бази, контроль цілісності через зовнішні ключі, вставка даних, динамічне формування запитів для фільтрації, перетворення табличних даних у об'єктну модель. Весь інший код застосунку працює вже не з SQL, а з готовими

Java-об'єктами, що суттєво спрощує логіку. Використання локальної кишенькової бази даних має істотно підвищити продуктивність застосунку в порівнянні з варіантом доступу до зовнішнього сервера.

Лістинг 3.6 – Фрагмент коду класу Game

```
public class Game implements Serializable {
    private long id;
    private String title;
    private long idCategory;
    private long idGenre;
    private String description;
    private String imageURL;
    private int year;
    private long idPublisher;
    private long idAgeRating;
    private int isOffline;
    private int isMultiplayer;
    private String urlPC;
    private String urlXBOX;
    private String urlPS;
    private String urlMobile;
    ...}
```

3.2 Реалізація графічного інтерфейсу

Розробка графічного інтерфейсу реалізована з використанням XML-моделі Android, де кожен екран представлений окремим layout-файлом. Логіка взаємодії користувача з графічним інтерфейсом винесена у відповідні фрагменти згідно прототипу, представленого на рис. 2.3. XML-код головного меню застосунку показано в лістингу 3.7. Воно побудоване на основі контейнера `LinearLayout` з вертикальною орієнтацією. Центрування елементів виконується через

android:gravity="center". Це фокусує користувача на основних діях. Фон застосунку градієнтний та відрізняється в залежності від обраної категорії ігор. Для цього ми в папці drawable створюємо три файли з відповідними градієнтами:

- gradient_computer.xml;
- gradient_mobile.xml;
- gradient_board.xml.

В такий спосіб ми можемо для десктопних ігор показати синій градієнт, для мобільних зелений, а для настільних – коричневий. Наприклад, для синього кольору налаштовуємо атрибути: `<gradient android:startColor="#1A237E" android:endColor="#3949AB" android:angle="45"/>`.

Текстові елементи виконують роль навігації. Великий заголовок задає контекст застосунку. Підпис меншого розміру є другорядним і не відволікає увагу. Основна взаємодія реалізована через три кнопки однакового розміру, які відповідають категоріям ігор. Розміри кнопок однакові. Кожна кнопка є тригером переходу до відповідного фрагмента фільтрації ігор обраної категорії. Обробники натискання кнопок прописуються прямо у фрагментах, пов'язаних з цією версткою. Фрагменти завантажують динамічно через FragmentManager. Таким чином, UI відображає модель use-case (рис. 2.1) без проміжних станів.

Екран фільтрації ігор реалізує вибір варіантів користувачем для динамічного формування запиту. Базовий контейнер тут також LinearLayout, але інтерфейс значно складніший. Кожен фільтр представлений парою CheckBox + елемент введення (Spinner або EditText). Наявність CheckBox дозволяє відокремити «поле задане» від «поле ігнорується». Це істотно спрощує логіку формування SQL-запиту в DBController. Фрагмент коду верстки екрану фільтрації десктопних ігор наведено в лістингу 3.8.

Використання Spinner для жанру, видавця та рейтингу означає, що ці дані нормалізовані і підтягуються з бази, а не вводяться вручну. Це виключає некоректні значення. Для року виходу гри застосовано EditText з числовим типом вводу. Це обмежує введення лише числовими значеннями і зменшує потребу у валідації.

Лістинг 3.7 – Верстка головного меню

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="20dp"
    android:background="@drawable/gradient_computer">
    <TextView
        android:text="Система пошуку ігор"
        android:textSize="32sp" android:textColor="#FFFFFF"
        android:textStyle="bold" />
    <TextView
        android:text="by Nastya"
        android:textSize="14sp" android:textColor="#CCCCCC"
        android:layout_marginBottom="40dp" />
    <Button
        android:id="@+id/btnPC"
        android:layout_width="match_parent"
        android:layout_height="80dp"
        android:text="Десктопні ігри"
        android:layout_margin="10dp" />
    <Button
        android:id="@+id/btnMobile"
        android:layout_width="match_parent"
        android:layout_height="80dp"
        android:text="Мобільні ігри"
        android:layout_margin="10dp" />
    <Button
        android:id="@+id/btnBoard"
        android:layout_width="match_parent"
        android:layout_height="80dp"
        android:text="Настільні ігри"
        android:layout_margin="10dp" />
</LinearLayout>
```

Окремий блок чекбоксів Single/Multi реалізує булеву фільтрацію без додаткових елементів введення. Аналогічно побудований блок платформ (PC, XBOX, PS), який відповідає множинному вибору і трансліюється у відповідні умови WHERE. Кнопка пошуку є фінальним тригером, який виконує збір стану всіх UI-компонентів. Таким чином, layout безпосередньо відображає структуру майбутнього SQL-запиту.

Лістинг 3.8 – Фрагмент коду верстки екрану фільтрації десктопних ігор

...

```
<TextView
    android:text="Пошук десктопних ігор"
    android:textColor="#FFFFFF"
    android:textSize="22sp"
    android:textStyle="bold"
    android:layout_marginBottom="20dp" />
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="center_vertical"
    android:minHeight="48dp">
    <CheckBox
        android:id="@+id/checkGenre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Жанр:" android:textColor="#FFF"/>
    <Spinner
        android:id="@+id/spinnerGenre"
        android:layout_width="match_parent"
        android:layout_height="48dp" />
</LinearLayout>
```

...

Відображення результатів фільтрації реалізоване на окремому екрані через

RecyclerView, який ефективно працює зі списками. Важливим компонентом тут виступає GameAdapter (лістинг 3.9), який реалізує патерн ViewHolder. Це дозволяє зменшити кількість викликів findViewById і підвищує продуктивність при прокрутці. Кожен елемент списку демонструє постер гри, назву та рік випуску. Користувач може обрати будь-яку гру та перейти на наступний екран з деталями конкретної гри.

Метод onCreateViewHolder відповідає за зв'язок layout одного елемента списку (item_game), тобто створення View на основі XML. Метод onBindViewHolder виконує прив'язку даних: назва гри і рік встановлюються у TextView, а зображення підвантажується через бібліотеку Glide. Використання Glide є важливим, оскільки воно виконує асинхронне завантаження, кешує зображення, запобігає блокуванню UI-потoku. Завантажувати зображення без використання бібліотеки істотно знизило би продуктивність застосунку та могло би заблокувати UI-поток в залежності від швидкості інтернету.

Обробник кліку реалізований через інтерфейс OnItemClickListener. Це дозволяє делегувати навігацію у Fragment без зв'язування адаптера з конкретною реалізацією.

Анімації покращують користувацький досвід переходів між екранами. Використаємо комбінацію alpha і translate типів анімації:

- поява екрана реалізується як одночасний зсув з правого краю і поступове збільшення прозорості;
- зникнення фрагменту - зворотний процес.

Тривалість анімації 400 мс не тягне на себе увагу надто істотно.

На екрані списку ігор розмістимо рекламний банер. Це окремий UI-компонент, який не порушує основний use-case - перегляд та вибіру гри. Функціональна роль баннера полягає в монетизації застосунку через показ реклами та ненав'язливе доповнення інтерфейсу без блокування основного контенту. Користувач може ігнорувати банер, тмоу що він розміщується внизу списку і не перекриває елементи інтерфейсу.

Лістинг 3.9 – Код GameAdapter для списків

```
public class GameAdapter extends
RecyclerView.Adapter<GameAdapter.GameViewHolder> {
    private final ArrayList<Game> gameList;
    private final OnItemClickListener listener;
    public interface OnItemClickListener {
        void onItemClick(Game game);}
    public GameViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
        View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.item_game,
parent, false);
        return new GameViewHolder(view);}
    public void onBindViewHolder(GameViewHolder holder, int
position) {
        Game game = gameList.get(position);
        holder.itemTitle.setText(game.getTitle());
        holder.itemYear.setText("Рік випуску: " + game.getYear());
        Glide.with(holder.itemView.getContext())
            .load(game.getImageURL())
            .placeholder(android.R.drawable.ic_menu_gallery)
            .into(holder.itemImage);
        holder.itemView.setOnClickListener(v ->
listener.onItemClick(game));}
    public int getItemCount() {
        return gameList != null ? gameList.size() : 0;}
    public static class GameViewHolder extends
RecyclerView.ViewHolder {
        ImageView itemImage;
        TextView itemTitle, itemYear;
        public GameViewHolder(View itemView) {
            super(itemView);
            itemImage = itemView.findViewById(R.id.itemImage);
            itemTitle = itemView.findViewById(R.id.itemTitle);
            itemYear = itemView.findViewById(R.id.itemYear);} } }
```

Останній екран застосунку згідно прототипу - екран детальної інформації про гру (лістинг 3.10). Він виконує виключно функцію представлення вже отриманої інформації. В корені layout є ScrollView, що дозволяє відобразити потенційно великий обсяг інформації (опис гри) без обмеження висотою екрана. Тут допускається вертикальна прокрутка, оскільки інформація читається послідовно.

ImageView у верхній частині концентрує увагу на постері з грою. Встановлена висота 300dp і scaleType=fitCenter забезпечують коректне відображення зображень різних пропорцій без обрізання. Завантаження, як і в списку, відбувається через Glide. TextView detailTitle із збільшеним розміром 28sp показує назву гри. Вся подальша інформація структурована у вигляді пар мітка - значення, реалізованих через горизонтальні LinearLayout.

Жовтий колір міток використовується як акцент, який допомагає швидко орієнтуватися в структурі даних. Значення відображаються білим кольором. На цьому екрані вже не використовуються ідентифікатори. Замість цього з бази витягуються текстові дані через DBController. Таким чином, UI отримує вже підготовлені до відображення дані і не містить логіки доступу до бази. Блок «Отримати гру» реалізує можливість переходу на сайт видавця гри. На відміну від попередніх екранів тут використовується динамічна видимість кнопок android:visibility=gone.

Кнопки платформ (PC, XBOX, PS, Mobile) відображаються лише у випадку, якщо для конкретної гри існує відповідне посилання або підтримка платформи. Після отримання об'єкта Game перевіряються поля з посиланнями на платформу гри. В результаті відповідні кнопки переводяться у стан VISIBLE.

Лістинг 3.10 – Фрагмент коду fragment_game_detail

```

<ScrollView ...
    android:background="@drawable/gradient_computer">
    <LinearLayout ...>
    <ImageView
        android:id="@+id/detailImage"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:scaleType="fitCenter" />
    <TextView
        android:id="@+id/detailTitle"...
        android:textSize="28sp"
        android:textColor="#FFF"/>
    <LinearLayout ...>
    <TextView ... android:text="Жанр: "
android:textColor="#FFEB3B" />
    <TextView android:id="@+id/detailGenre" .../>
    </LinearLayout>
    <LinearLayout ...>
    <TextView ... android:text="Пік: "/>
    <TextView android:id="@+id/detailYear" .../>
    </LinearLayout>
    ...
    <Button android:id="@+id/btnPC" ... android:text="PC"
android:visibility="gone"/>
    <Button android:id="@+id/btnXbox" .../>
    <Button android:id="@+id/btnPS" ... android:text="PS Store"/>
    <Button android:id="@+id/btnMobile" .../>
    ...</ScrollView>

```

3.3 Реалізація бізнес-логіки системи

Бізнес-логіка системи реалізована на рівні Fragment-компонентів та частково делегується до DBController. Частина фільтрації виконується на рівні SQL-запитів в базу даних, інша - на рівні логіки обробки колекцій, витягнутих з бази. Логіка фрагментів ComputerGameFragment, MobileGameFragment та BoardGameFragment схожа, тому детальніше опишемо саме ComputerGameFragment. В ньому є два основні методи loadSpinnersData (лістинг 3.11), який завантажує дані з бази та демонструє їх в елементах інтерфейсу та performSearch (лістинг 3.12), який шукає список ігор по заданих критеріях пошуку фрагмент коду якого наведено в лістингу 3.11.

Реалізація бізнес-логіки починається з формування параметрів пошуку на основі стану елементів керування. Кожен параметр фіотру (жанр, видавець, віковий рейтинг) бере участь у запиті лише за умови, що відповідний CheckBox активний. У протилежному випадку використовується службове значення -1, яке інтерпретується на рівні запиту як відсутність обмеження. Це дозволяє уникнути динамічного конструювання SQL-виразу з різною кількістю.

Перед виконанням пошуку відбувається отримання довідкових даних з бази через DBController методами getAllGenres, getAllPublishers та getAllAgeRatings. Ці виклики фактично є SELECT-запитами без фільтрації, які повертають повні таблиці відповідних сутностей. В пам'яті зберігаються не лише текстові значення, а і самі об'єкти. Це дозволяє коректно отримати їхні ідентифікатори для формування запиту.

Основний запит виконується через метод getFilteredGames, якому передається категорія гри та набір параметрів фільтрації. На цьому етапі реалізується SELECT даних на рівні SQLite. Логіка запиту будується таким чином, що кожен параметр або враховується у WHERE-умові, або ігнорується, якщо його значення дорівнює -1. Це дозволяє обмежити вибірку і зменшити обсяг оброблюваних даних.

Лістинг 3.11 – Метод loadSpinnersData фрагменту ComputerGameFragment

```
private void loadSpinnersData() {
    DBController db = ((MainActivity)
getActivity()).getDbController();
    genreList = db.getAllGenres();
    ArrayList<String> genreNames = new ArrayList<>();
    for (Genre g : genreList) genreNames.add(g.getName());
    spGenre.setAdapter(new ArrayAdapter<>(getContext(),
android.R.layout.simple_spinner_dropdown_item, genreNames));
    publisherList = db.getAllPublishers();
    ArrayList<String> pubNames = new ArrayList<>();
    for (Publisher p : publisherList) pubNames.add(p.getName());
    spPublisher.setAdapter(new ArrayAdapter<>(getContext(),
android.R.layout.simple_spinner_dropdown_item, pubNames));
    ratingList = db.getAllAgeRatings();
    ArrayList<String> ratingNames = new ArrayList<>();
    for (AgeRating r : ratingList)
ratingNames.add(r.getRating());
    spRating.setAdapter(new ArrayAdapter<>(getContext(),
android.R.layout.simple_spinner_dropdown_item, ratingNames));
}
```

Після отримання списку об'єктів Game виконується додаткова фільтрація вже в оперативній пам'яті. Вона реалізована через Iterator, що дозволяє безпечно видаляти елементи колекції під час обходу. Тут перевіряється мінімальний рік випуску гри, якщо активовано відповідний фільтр, а також наявність посилань на конкретні платформи.

Перевірка платформи зводиться до аналізу рядкових полів URL: якщо поле порожнє або null, вважається, що гра не підтримує відповідну платформу. Після завершення фільтрації відбувається передача результату в MainActivity, яка виконує роль сховища стану. Список знайдених ігор зберігається у пам'яті, після чого ініціюється перехід до GameListFragment через метод replaceFragment.

Лістинг 3.12 – Метод performSearch фрагменту ComputerGameFragment

```
private void performSearch() {
    MainActivity activity = (MainActivity) getActivity();
    if (activity == null) return;
    long genId = chGenre.isChecked() ?
genreList.get(spGenre.getSelectedItemPosition()).getId() : -1;
    long pubId = chPub.isChecked() ?
publisherList.get(spPublisher.getSelectedItemPosition()).getId() : -1;
    long ratId = chRating.isChecked() ?
ratingList.get(spRating.getSelectedItemPosition()).getId() : -1;
    int offline = cbSingle.isChecked() ? 1 : -1;
    int multi = cbMulti.isChecked() ? 1 : -1;
    ArrayList<Game> games =
activity.getDbController().getFilteredGames(1, genId, pubId, ratId,
offline, multi);
    Iterator<Game> iterator = games.iterator();
    while (iterator.hasNext()) { Game game = iterator.next();
        boolean remove = false;
        if (chYear.isChecked() && !etYear.getText().toString().isEmpty()) {
            int minYear = Integer.parseInt(etYear.getText().toString());
            if (game.getYear() < minYear) remove = true;}
            if (cbPC.isChecked() && (game.getUrlPC() == null ||
game.getUrlPC().isEmpty())) remove = true;
            if (cbXbox.isChecked() && (game.getUrlXBOX() == null ||
game.getUrlXBOX().isEmpty())) remove = true;
            if (cbPS.isChecked() && (game.getUrlPS() == null ||
game.getUrlPS().isEmpty())) remove = true;
            if (remove) { iterator.remove();}
        }
        if (games.isEmpty()) {
            Toast.makeText(getContext(), "Жодна актуальна гра не
знайдена", Toast.LENGTH_LONG).show();}
        else{ activity.setFoundGames(games);
            GameListFragment listFragment = new GameListFragment();
            Bundle args = new Bundle();
            args.putInt("bg_gradient", R.drawable.gradient_computer);
            listFragment.setArguments(args);
            activity.replaceFragment(listFragment, true);} }
```

У `GameListFragment` бізнес-логіка зводиться до отримання вже підготовленого списку ігор із `MainActivity` без повторного звернення до бази даних. Це рішення мінімізує кількість. Вибір конкретної гри користувачем обробляється через `callback`, який ініціює створення нового `GameDetailsFragment` і передає в нього об'єкт гри. Таким чином формується наступний перехід по фрагментах згідно прототипу застосунку. На цьому ж фрагменті реалізована логіка роботи з банером (лістинг 3.13). Зображення банеру не зберігається локально і не входить до ресурсів додатку, а підтягується з мережі під час виконання. `Glide` виконує весь процес: виконує HTTP-запит, декодує зображення, кешує його і встановлює у `ImageView`. Клік по банеру відкриває сайт магазину з іграми.

Лістинг 3.13 – Логіка роботи з банером

```
ImageView banner = view.findViewById(R.id.bannerRozetka);
Glide.with(this).load("https://xl-static.rozetka.com.ua/assets/img/main/rozetka.png").into(banner);
banner.setOnClickListener(v -> {
    Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse("https://rozetka.com.ua/ua/games/c80066/"));
    startActivity(intent);
});
```

У `GameDetailsFragment` відбувається перетворення ідентифікаторів у читабельні значення. Для цього виконуються точкові запити через `DBController`. Кожен із таких викликів є `SELECT`-запитом з умовою по первинному ключу, який повертає одне значення. Перетворення виконується безпосередньо перед відображенням. Додатково реалізована логіка перевірки наявності зовнішніх ресурсів у вигляді URL. Для кожної платформи перевіряється, чи існує відповідне посилання, і лише в цьому випадку користувачу надається можливість переходу. Натискання на кнопку створює `Intent` із дією `ACTION_VIEW`, що передає обробку зовнішньому браузеру. Саму гру, яку обрав користувач і яку потрібно показати на екрані фрагмент дістає з бандлу `Game game = (Game) args.getSerializable("game")`.

Для роботи застосунку в файлі маніфесту потрібно додати дозвіл на роботу з Інтернет `<uses-permission android:name="android.permission.INTERNET"/>`. В налаштуваннях системи зборки проєкту Gradle потрібно додати залежність на бібліотеку Glide: `implementation 'com.github.bumptech.glide:glide:4.15.1'`. Загальна структура проєкту системи наведена на рис. 3.1. Структура xml-документів верстки екранів наведена на рис. 3.2.

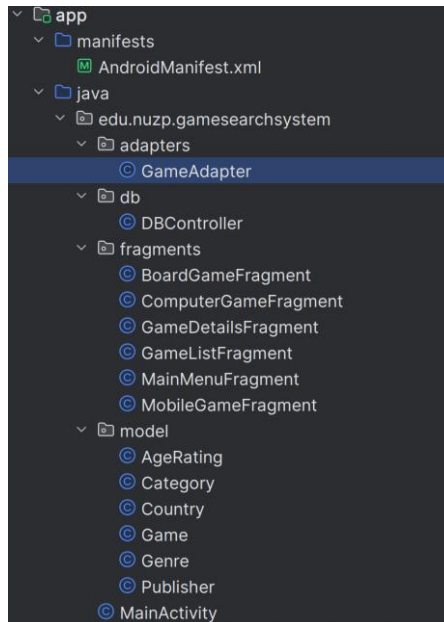


Рисунок 3.1 – Загальна структура проєкту системи

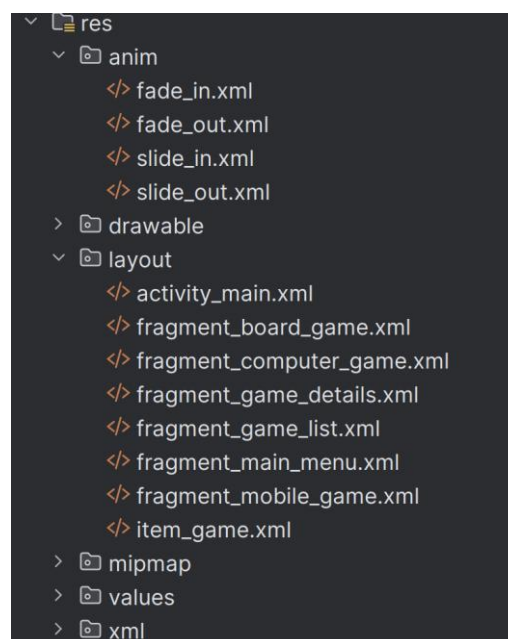


Рисунок 3.2 – Структура xml-документів верстки екранів

3.4 Висновки по розділу

У цьому розділі реалізовано повноцінну комп'ютерну систему пошуку ігор із чітким розділенням рівнів даних, логіки та представлення. Основа системи - це локальна база даних SQLite, яка забезпечує зберігання структурованої інформації та підтримку цілісності через механізм зовнішніх ключів. Використання SQLiteOpenHelper дозволило організувати створення схеми, ініціалізацію та доступ до даних, що усуває дублювання SQL-коду в інших частинах застосунку.

Реалізована структура бази даних є нормалізованою: сутності винесені в окремі таблиці, а основна таблиця Games містить лише посилання на них. Ключовим елементом є механізм динамічного формування SQL-запитів у методі `getFilteredGames`. Він дозволяє будувати WHERE-умову залежно від вибору користувача без створення множини жорстко заданих запитів. Це рішення є масштабованим і легко розширюється при додаванні нових параметрів фільтрації.

Бізнес-логіка системи реалізована як комбінація сервероподібної обробки на рівні бази даних та постобробки в оперативній пам'яті. Частина фільтрації виконується на рівні SQL, що зменшує обсяг вибірки, а частина - на рівні колекцій, що дозволяє реалізувати складніші умови без ускладнення запиту.

Інтерфейс системи побудований на основі Fragment-підходу, де `MainActivity` виконує роль координатора навігації та сховища стану. Передача даних між фрагментами здійснюється через `Bundle` із використанням `Serializable`. Структура графічних елементів фільтрації відповідає структурі SQL-запитів, а екран списку та деталей працюють із уже підготовленими даними. Використання `RecyclerView` і патерну `ViewHolder` забезпечує ефективну роботу зі списками. Бібліотека `Glide` вирішує задачі асинхронного завантаження та кешування зображень. Додатково реалізовано інтеграцію із зовнішніми ресурсами через банер та переходи на сторінки ігор.

4 ВИПРОБУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ ПОШУКУ ІГОР

4.1 Випробування бази даних

Випробування бази даних є важливим етапом, що дозволяє підтвердити цілісність структури, правильність встановлених зв'язків між таблицями та актуальність наповнення системи. Вся бізнес-логіка тримається а взаємодії з базою, тому якщо на цьому етапі будуть знайдені помилки, то подальше випробування системи без їх виправлення не матиме сенсу. Для випробування бази цього ми запустили застосунок, він створив таблиці та наповнив їх контентом згідно методу onCreate класу DBController. Далі в емуляторі Android запускаємо наш застосунок і він створює базу GameCatalog.db в сховищі даних цього застосунку. Копіюємо цю базу через файловий менеджер емулятора Android та відкриваємо її.

Спочатку було перевірено загальний стан сховища, що відображено на рисунку 4.1, де зафіксовано наявність шести основних таблиць та поточну кількість записів у них. Чітко видно, що основна таблиця ігор на момент випробувань містить двадцять об'єктів. Інші таблиці також містять рівно стільки записів скільки ми додавали на етапі створення бази.

Tables		name	seq
▶ AgeRatings		Filter = 0 !	Filter = 0 !
▶ android_metadata			
▶ Categories	1	Countries	8
▶ Countries	2	Categories	3
▶ Games	3	Genres	9
▶ Genres	4	AgeRatings	4
▶ Publishers	5	Publishers	8
▶ sqlite_sequence	6	Games	20
	7		

Рисунок 4.1 – Випробування бази даних

Для перевірки коректності класифікації ігор було проаналізовано таблиці, які забезпечують роботу фільтрів застосунку. На рисунку 4.2 показано три базові категорії: комп'ютерні, мобільні та настільні ігри, кожна з яких має унікальний ідентифікатор та детальний опис призначення. Під Computer game ми розуміємо всі десктопні платформи, тобто PC, XBOX, Playstation.

_id	name	description
1	Computer Game	Ігри для персональних комп'ютерів та ігрових консоле...
2	Mobile Game	Сучасні мобільні проекти для платформи Android
3	Board Game	Популярні настільні ігри з високим рейтингом BGG

Рисунок 4.2 – Вміст таблиці Categories

Відповідність ігор віковим обмеженням перевірялася за допомогою даних, представлених в таблиці AgeRatings на рисунку 4.3. Тут зафіксовано чотири рівні рейтингу PEGI з відповідними текстовими поясненнями щодо змісту контенту.

_id	rating	description
1	PEGI 3	Для будь-якого віку
2	PEGI 12	Від 12 років: помірне насильство
3	PEGI 16	Від 16 років: реалістичне насильство
4	PEGI 18	Тільки для дорослих: жорсткий контент

Рисунок 4.3 – Вміст таблиці AgeRatings

Жанрова різноманітність ігор, представлених в нашій системі, підтверджується даними з таблиці Genres. Тут показано дев'ять різних ігрових напрямків, від Action-RPG до Dungeon Crawler. Це дозволяє користувачу здійснювати точне налаштування жанру в пошуковому запиті. Вміст таблиці показано на рис. 4.4.

	_id	name	description
1	1	Action-RPG	Рольові ігри з фокусом на бойову систему
2	2	Survival Horror	Жанр, де акцент зроблено на виживанні та атмосфері
3	3	Strategy	Ігри, що вимагають тактичного та стратегічного мисле...
4	4	MOBA	Командні битви на арені в реальному часі
5	5	Puzzle	Логічні ігри та просторові головоломки
6	6	Battle Royale	Змагання на виживання у відкритому світі
7	7	Eurogame	Настільні ігри з глибокою економічною складовою
8	8	Deck-building	Механіка формування власної колоди під час партії
9	9	Dungeon Crawler	Пригоди героїв у небезпечних підземеллях
10			

Рисунок 4.4 – Вміст таблиці Genres

Таблиця Counties демонструє країни, в яких працюють видавці ігор, а таблиця Publishers показує конкретні компанії, наприклад, CD Projekt Red або GSC Game World, що прив'язані до своїх країн через зовнішні ключі. Publishers також містить посилання на офіційні вебсайти видавців. На рисунку 4.5 продемонстрований вміст цих таблиць.

	_id	name
1	1	USA
2	2	Japan
3	3	Poland
4	4	France
5	5	Ukraine
6	6	Belgium
7	7	China
8	8	Germany
9		

	_id	name	idCo...	website
1	1	CD Projekt Red	3	https://www.cdprojektred.com
2	2	FromSoftware	2	https://www.fromsoftware.jp
3	3	Ubisoft	4	https://www.ubisoft.com
4	4	GSC Game World	5	https://www.gsc-game.com
5	5	HoYoverse	7	https://www.hoyoverse.com
6	6	Dire Wolf	1	https://www.direwolfdigital.com
7	7	Stonemaier Games	1	https://stonemaiergames.com
8	8	Larian Studios	6	https://larian.com
9				

Рисунок 4.5 – Вміст таблиць Counties та Publishers

Важливим етапом випробувань бази даних є аудит головної таблиці ігор. На рисунку 4.6 показано заповнення основних атрибутів: назва гри, рік випуску, короткий опис, посилання на постер в Інтернеті та ідентифікатори зв'язків, що підтверджує успішне об'єднання всіх допоміжних таблиць у єдину сутність. Також тут зберігаються прапорці для офлайн- та багатокористувацьких режимів та прямі посилання на цифрові магазини PC, Xbox, PlayStation та Google Play. Відсутність гри на певній платформі відображається в таблиці як NULL у відповідному полі. База даних не лише зберігає інформацію, а і забезпечує можливість переходу користувача до сторінки обраної гри.

Rows: 20											
	id	title	idCat...	idGenre	description	imageURL	year	idPubl...	idRati...	isOffline	isMultipl...
1	1	Elden Ring	1	1	Масштабна екшн-RPG у відкрито...	https://cdn.g.euro...	2022	2	3	1	1
2	2	Cyberpunk 2077	1	1	Пригодницька рольова гра, дія як...	https://cdn.artstatio...	2020	1	4	1	0
3	3	Baldur's Gate 3	1	1	Нове покоління RPG від творців ...	https://images.gog-s...	2023	8	4	1	1
4	4	S.T.A.L.K.E.R. 2: Heart of Ch...	1	2	Легендарне повернення в Чорно...	https://images.gog-s...	2024	4	4	1	0
5	5	Hogwarts Legacy	1	1	Закоплива рольова гра у відкрит...	https://cdn1.epicga...	2023	3	2	1	0
6	6	Alan Wake 2	1	2	Психологічний хоррор про Cary ...	https://upload.wikim...	2023	3	4	1	0
7	7	Resident Evil Village	1	2	Викликання у тасмичному засніже...	https://images.squar...	2021	2	4	1	0
8	8	Genshin Impact	2	4	Грандіозна RPG у відкритому світі...	https://store-images...	2020	5	2	0	1
9	9	Honkai: Star Rail	2	4	Космічна стратегічна рольова гра...	https://i1.sndcdn.co...	2023	5	2	0	1
10	10	Marvel Snap	2	8	Інноваційна карткова гра з швид...	https://m.media-am...	2022	6	2	0	1
11	11	Diablo Immortal	2	1	Повноцінний досвід Diablo на мо...	https://gaming-cdn...	2022	8	4	0	1
12	12	Wild Rift	2	4	Мобільна адаптація League of Leg...	https://cdnb.artstati...	2020	5	2	0	1
13	13	Monument Valley 2	2	5	Подорож крізь неможливу архіте...	https://m.media-am...	2020	3	1	1	0
14	14	Call of Duty: Mobile	2	6	Легендарний шутер з багатьма р...	https://i.pinimg.com...	2020	3	4	0	1
15	15	Dune: Imperium	3	8	Боротьба за Арракіс: побудова ко...	https://cf.geekdo-im...	2020	6	2	1	1
16	16	Ark Nova	3	7	Створення та розвиток сучасного...	https://cf.geekdo-im...	2021	7	2	1	1
17	17	Cascadia	3	7	Екологічна стратегія про розбуд...	https://geekdad.com...	2021	7	1	1	1
18	18	Wingspan	3	7	Змагання орнітологів: збирайте п...	https://blog.nature.o...	2019	7	1	1	1
19	19	7 Wonders Architects	3	7	Динамічне будівництво чудес сві...	https://cdn.svc.asmo...	2021	3	1	1	1
20	20	Gloomhaven: Jaws of the L...	3	9	Кооперативні пригоди у похмуро...	https://cdn.shopify.c...	2020	7	2	1	1
Rows: 20											
	isMultipl...	urlPC	urlXBOX	urlPS	urlMobile						
1	1	https://store.steampowered.com/app/1245620/ELDEN_RL...	https://www.xbox.com/games/elden-ring	https://www.playstation.com/games/elden-ring/	NULL						
2	0	https://www.cyberpunk.net/buy	https://www.xbox.com/games/cyberpunk-2077	https://www.playstation.com/games/cyberpunk-2077/	NULL						
3	1	https://baldursgate3.game/	https://www.xbox.com/games/baldurs-gate-3	https://www.playstation.com/games/baldurs-gate-3/	NULL						
4	0	https://www.stalker2.com/uk	https://www.xbox.com/games/stalker-2	NULL	NULL						
5	0	https://www.hogwartslegacy.com/	https://www.xbox.com/games/hogwarts-legacy	https://www.playstation.com/games/hogwarts-legacy/	NULL						
6	0	https://www.alanwake.com/	https://www.xbox.com/games/alan-wake-2	https://www.playstation.com/games/alan-wake-2/	NULL						
7	0	https://www.residentevil.com/village/	https://www.xbox.com/games/resident-evil-village	https://www.playstation.com/games/resident-evil-village/	NULL						
8	1	https://genshin.hoyoverse.com/	NULL	https://www.playstation.com/games/genshin-impact/	https://play.google.com/store/apps/details?id=com.miH...						
9	1	https://hkr.hoyoverse.com/	NULL	https://www.playstation.com/games/honkai-star-rail/	https://play.google.com/store/apps/details?id=com.HoY...						
10	1	https://www.marvelsnap.com/	NULL	NULL	https://play.google.com/store/apps/details?id=com.nvsg...						
11	1	https://diabloimmortal.blizzard.com/	NULL	NULL	https://play.google.com/store/apps/details?id=com.blizz...						
12	1	NULL	NULL	NULL	https://play.google.com/store/apps/details?id=com.riotg...						
13	0	NULL	NULL	NULL	https://play.google.com/store/apps/details?id=com.usw...						
14	1	NULL	NULL	NULL	https://play.google.com/store/apps/details?id=com.activ...						
15	1	https://www.direwolfdigital.com/dune-imperium/	NULL	NULL	NULL						
16	1	https://boardgamegeek.com/boardgame/342942/ark-no...	NULL	NULL	NULL						
17	1	https://www.alderac.com/cascadia/	NULL	NULL	NULL						
18	1	https://stonemaiergames.com/games/wingspan/	NULL	NULL	NULL						
19	1	https://www.rprod.com/en/games/7-wonders-architects	NULL	NULL	NULL						
20	1	https://boardgamegeek.com/boardgame/291457/gloom...	NULL	NULL	NULL						

Рисунок 4.6 – Дані з таблиці Games

Усі виявлені під час випробувань дані відповідають спроектованій схемі.

Зв'язки між первинними та зовнішніми ключами працюють без помилок.

4.2 Випробування сценарію пошуку десктопної гри

Випробування сценарію пошуку десктопних ігор розпочинається з головного екрана застосунку, зображеного на рисунку 4.7, де користувач має можливість обрати одну з трьох основних категорій ігор.



Рисунок 4.7 – Головний екран застосунку

Після натискання на кнопку «Десктопні ігри» система перенаправляє користувача до спеціалізованого фрагмента пошуку, який представлений на рисунку 4.8. У цьому вікні акцент зроблено на технічних параметрах ігрових платформ та році релізу гри. Під час випробування було задано фільтрацію за роком випуску від 2021 року, обрано режими «Single» та «Multi», а також

обмежено пошук платформами PC та XBOX. Чекбокси вибору конкретного жанру та видавця було залишено неактивними, що дозволяє перевірити здатність системи до широкої вибірки за технічними характеристиками.

Після ініціації пошуку кнопкою «ЗНАЙТИ ІГРИ» застосунок відображає список результатів, продемонстрований на рисунку 4.9. Тут ми бачимо картки ігор Elden Ring та Baldur's Gate 3, які повністю відповідають заданим критеріям за роком випуску та доступністю. Також на цьому екрані підтверджено коректність відображення рекламного банера в нижній частині інтерфейсу, що є частиною функціональних вимог.

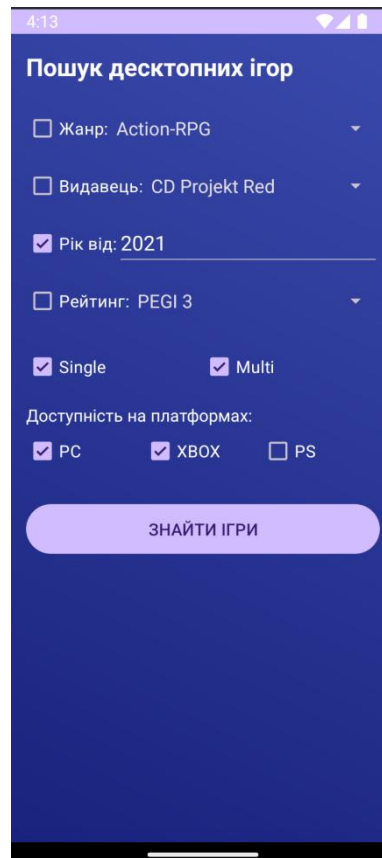


Рисунок 4.8 – Фрагмент пошуку десктопних ігор

Для детальної перевірки інформаційного наповнення було здійснено перехід до картки конкретної гри (рис. 4.10). На прикладі Baldur's Gate 3 система демонструє повний набір метаданих: жанр, рік релізу 2023, видавця та віковий рейтинг PEGI 18. У нижній частині екрана успішно відображаються кнопки

швидкого переходу до відповідних магазинів для PC, XBOX та PS Store.

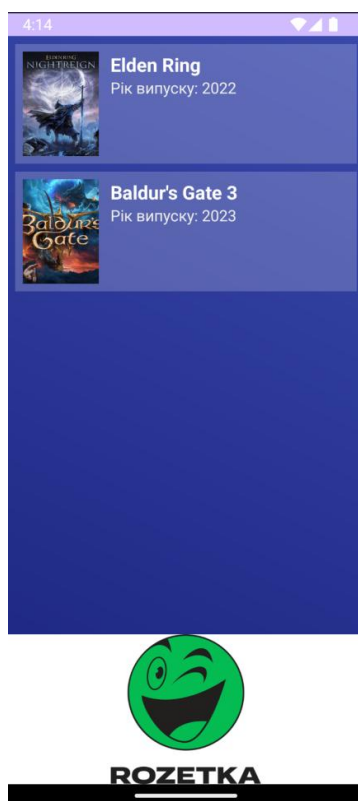


Рисунок 4.9 – Фрагмент списку результатів пошуку десктопних ігор

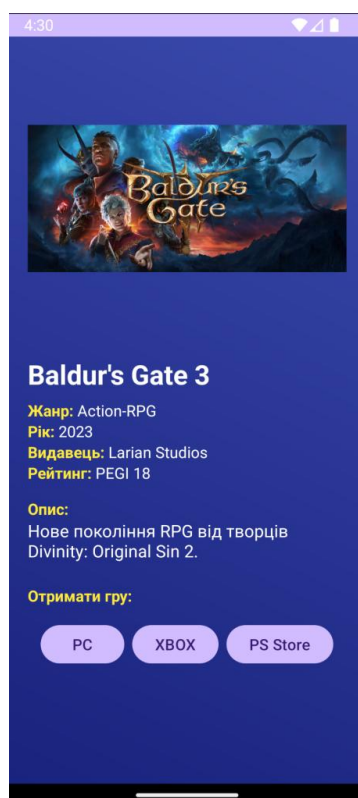


Рисунок 4.10 – Фрагмент деталізації обраної гри

Клік на кнопці PC відкриває браузер та переходить на офіційний сайт видавця гри з можливістю її придбати та завантажити (рис. 4.11). Бачимо на сайті гри, що платформи, під які гра доступна, чітко співпадають з кнопками в нашому застосунку, тобто можна завантажити для PC (в даному випадку зі Steam), PS5, XBOX series. В даному випадку всі три платформи ведуть на один сайт видавця, але це не завжди так і відповідні кнопки можуть вести на різні сайти.

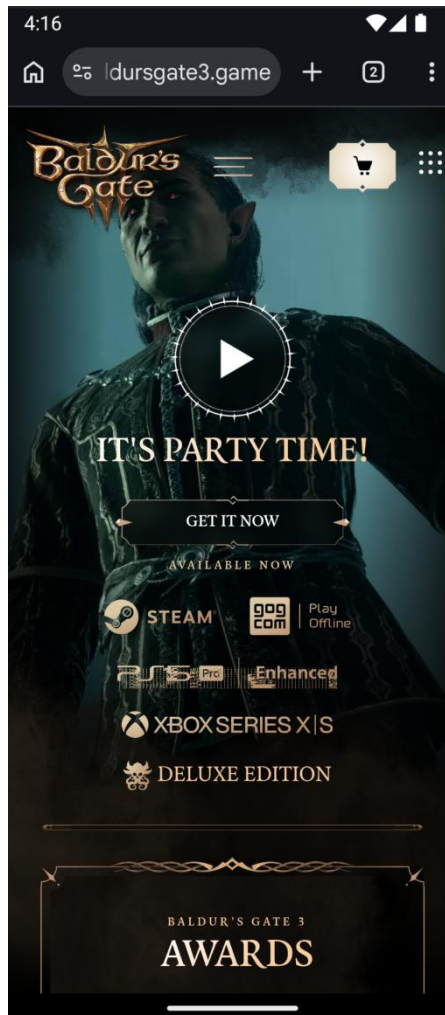


Рисунок 4.11 – Перехід на сайт гри

Для перевірки повноти бази даних на рисунку 4.12 представлено загальний перелік усіх десктопних ігор без застосування фільтрів. Сюди входять такі проекти як Cyberpunk 2077, S.T.A.L.K.E.R. 2, Hogwarts Legacy та інші. Якщо користувач натискає на рекламний банер, то відкривається браузер та здійснюється перехід на сайт продавця ігор (рис. 4.13). Конкретний продавець

ігор та банер може бути легко змінений під час оновлення застосунку. Для цього змінюємо посилання на постер магазину і посилання на сайт магазину. Користувачі побачать оновлення застосунку в Google Play, завантажуть його і разом з новою версією бази даних, де буде більше ігор, зможуть побачити інший магазин ігор.

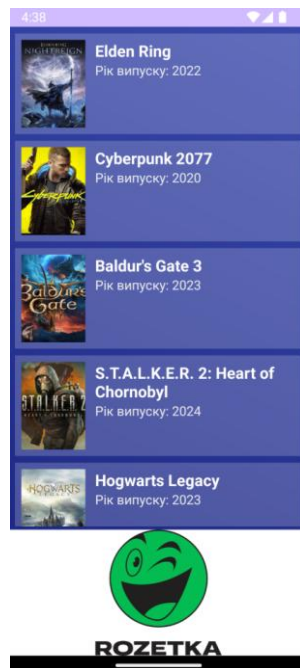


Рисунок 4.12 – Повний перелік десктопних ігор в застосунку

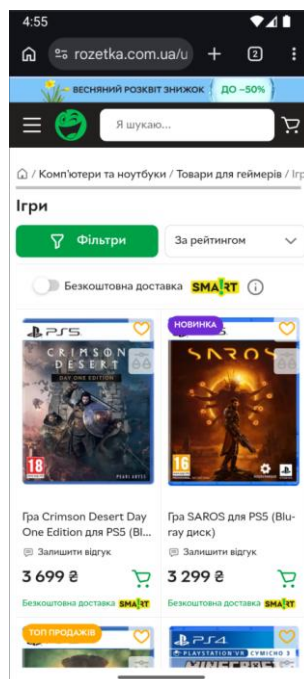


Рисунок 4.13 – Перехід на сайт продавця ігор

Для випробування реакції системи на відсутність результатів було змодельовано ситуацію пошуку гри у жанрі Action-RPG від видавця CD Projekt Red з роком випуску від 2026 року для платформи PlayStation. Оскільки на поточний момент таких записів у базі не існує, система коректно відпрацювала сценарій, вивівши в нижній частині екрана спливаюче повідомлення (Toast) з текстом «Жодна актуальна гра не знайдена» (рис. 4.14).

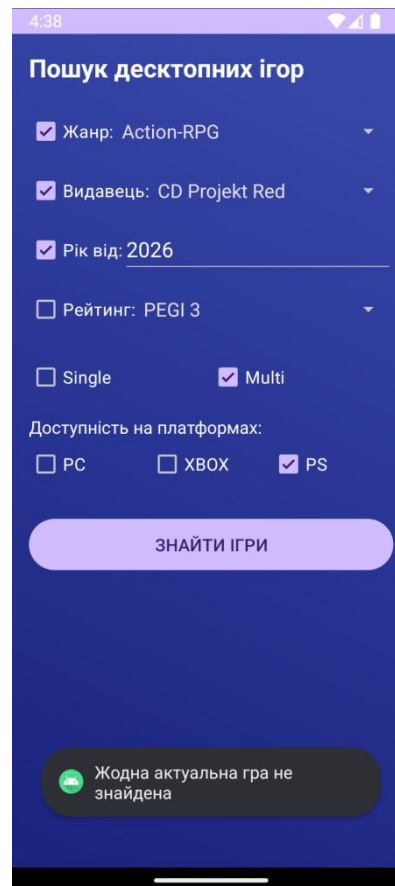


Рисунок 4.14 – Відсутність результатів пошуку

4.3 Випробування сценарію пошуку мобільної гри

Переходимо до перевірки того, як система справляється з мобільним сегментом ігор. Для цієї категорії ми обрали зелений дизайн інтерфейсу, який допомагає користувачу не заплутатися в категоріях. Екран пошуку представлено

на рисунку 4.15. Під час випробування ми вирішили зосередитися на сучасних проєктах, тому виставили фільтр «Рік від: 2022» та активували параметр «Multiplayer», щоб відсіяти прості офлайн-ігри. З рисунку видно, що тут фільтрів менше ніж в десктопних іграх.

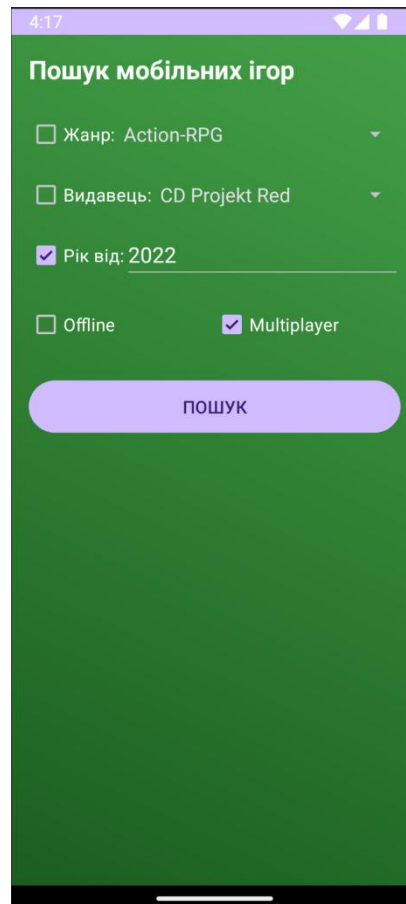


Рисунок 4.15 – Екран пошуку мобільних ігор

Результати пошуку, які ми бачимо на рисунку 4.16, підтвердили, що база даних реагує на запити без помилок. У списку з'явилися саме ті ігри, що відповідають критеріям: Honkai: Star Rail, Marvel Snap та Diablo Immortal. Всі обкладинки завантажилися коректно, рекламний блок показується у нижній частині екрана.

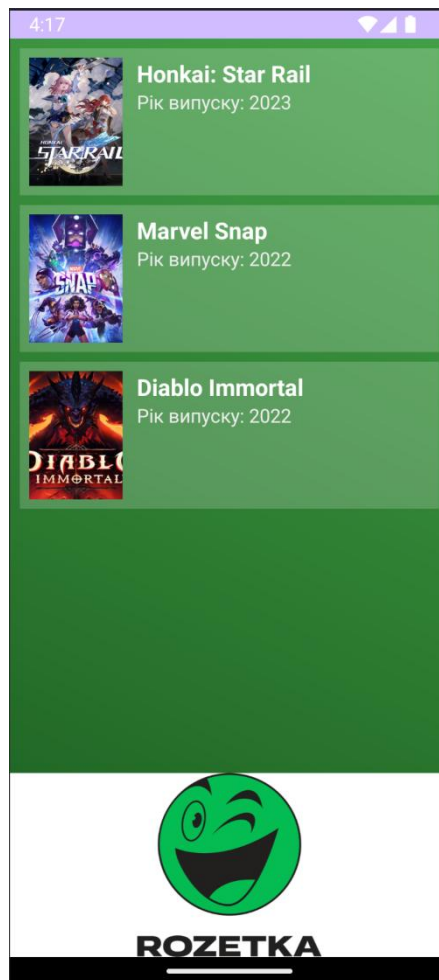


Рисунок 4.16 – Список мобільних ігор після застосування фільтра

Для детальної перевірки ми відкрили картку гри Marvel Snap, зображену на рисунку 4.17. Тут ми переконалися, що всі метадані від жанру «Deck-building» до вікового рейтингу PEGI 12 підтягуються з бази саме так, як ми їх туди занесли. Також ми перевірили найважливіший функціонал для мобільного геймера - можливість швидко перейти до завантаження. При натисканні на кнопку «Google Play» застосунок миттєво відкрив сторінку гри у зовнішньому браузері, що показано на рисунку 4.18. Цей успішний перехід доводить, що зв'язок внутрішньої бази даних із зовнішніми ресурсами працює коректно.

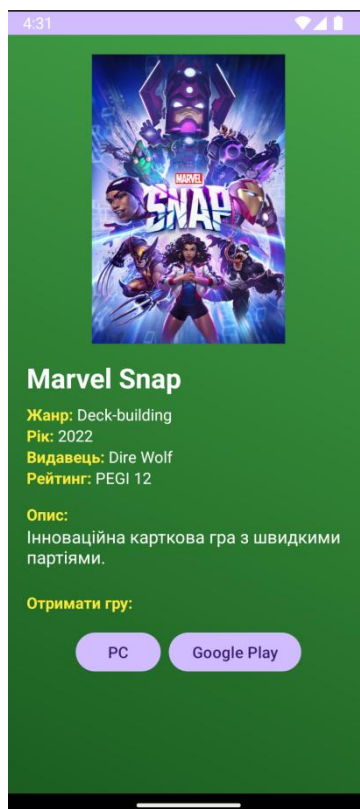


Рисунок 4.17 – Картка мобільної гри

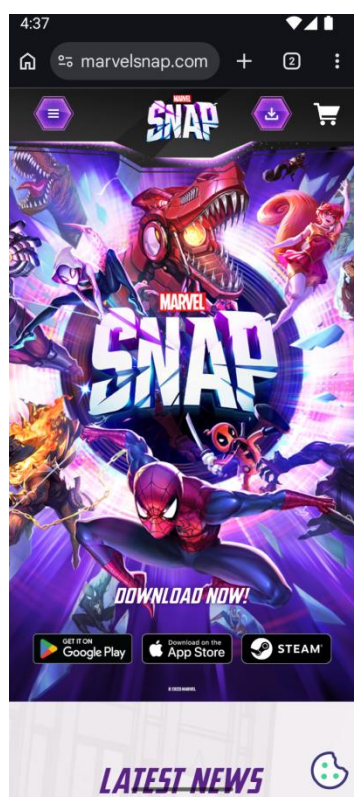


Рисунок 4.18 – Перехід на сторінку мобільної гри в браузері

4.4 Випробування сценарію пошуку настільної гри

Завершальним етапом випробувань стала перевірка розділу настільних ігор. Тут ми врахували специфіку настільних ігор: на відміну від відеоігор де важливі платформи, для настільних ігор головним є жанр. На рисунку 4.19 представлено вікно пошуку, виконане у коричневих тонах. Для випробування ми обрали фільтрацію за жанром «Dungeon Crawler», щоб знайти ігри про дослідження підземель, не обмежуючи систему за роком чи віком.

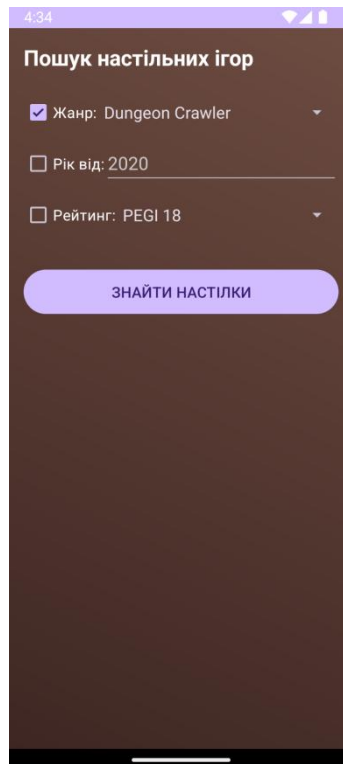


Рисунок 4.19 – Екран фільтру настільних ігор

Результат пошуку ігор на рисунку 4.20 видав нам один із найвідоміших проєктів у цьому сегменті: «Gloomhaven: Jaws of the Lion». Система відфільтрувала базу, залишивши лише ту гру, що відповідає критеріям пошуку. Список виглядає коректно.

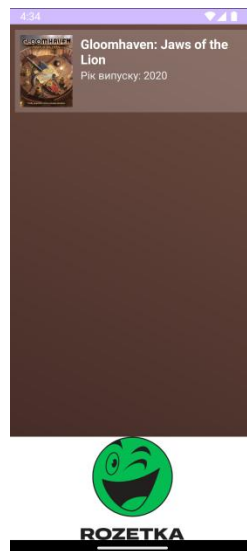


Рисунок 4.20 – Список настільних ігор після фільтрації

Коли ми відкриваємо картку гри, то на рисунку 4.21 бачимо повний опис гри: від року випуску до короткого опису. Для настільних ігор ми зберегли можливість переходу до зовнішніх ресурсів. Натиснувши на кнопку переходу, ми потрапляємо на сайт BoardGameGeek, на якому зібрано багато настільних ігор. На рисунку 4.22 видно, що посилання спрацювало правильно, відкривши сторінку саме цієї гри. Це підтверджує, що наш застосунок успішно пов'язує локальну базу даних із зовнішніми ресурсами.



Рисунок 4.21 – Картка настільної гри

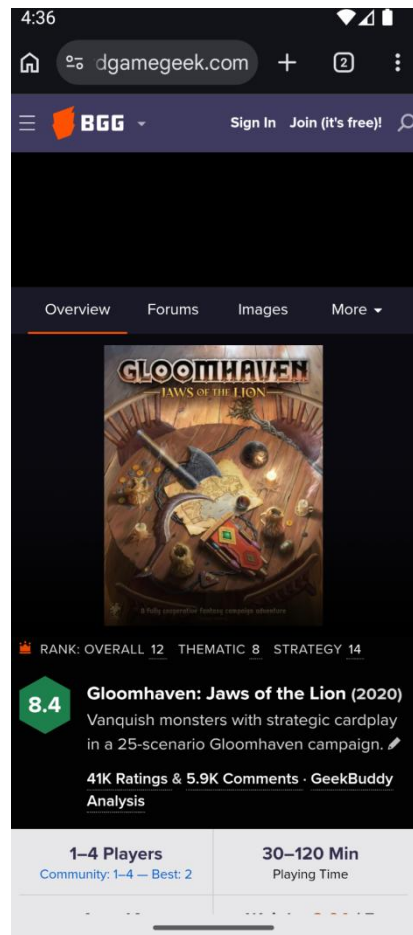


Рисунок 4.22 – Перехід на сторінку настільної гри в браузері

Таким чином, випробування всіх основних сценаріїв застосування продемонструвало його відповідність поставленим завданням.

4.5 Висновки по розділу

Після проведення всіх випробувань можна зробити кілька висновків:

- база даних працює надійно. Усі таблиці правильно зв'язані між собою. Кожна гра підтягує свою категорію, жанр та рейтинг без жодних збоїв;
- пошук працює точно. Фільтри в усіх трьох розділах (десктоп, мобільні, настільні) видають саме ті результати, які запитує користувач. Система не помиляється з роками випуску чи жанрами;

- інтерфейс зрозумілий. Колірне розділення (синій для ПК, зелений для мобільних, коричневий для настільних ігор) допомагає одразу зорієнтуватися в додатку;
- зовнішні посилання активні. Кнопки переходу в магазини працюють миттєво і відкривають правильні сторінки в зовнішньому браузері;
- помилки обробляються коректно. Якщо в базі немає гри за обраними фільтрами, застосунок не вилітає, а просто попереджає про це користувача коротким повідомленням.

Загалом, система повністю готова до роботи. Вона швидко шукає ігри та надає користувачу всю необхідну інформацію.

ВИСНОВКИ

У ході виконання дипломної роботи було успішно вирішено задачу розробки комп'ютерної системи пошуку ігор, реалізованої у вигляді автономного мобільного застосунку для ОС Android. У результаті розробки було досягнуто наступних результатів:

- проведено аналіз предметної області, який показав, що сучасні цифрові магазини (Steam, Google Play тощо) перевантажені комерційними рекомендаціями та не надають універсального інструменту для чистого пошуку ігор різних категорій (включно з настільними). Це підтвердило актуальність створення незалежної пошукової системи;
- спроектовано архітектуру та базу даних системи. Обґрунтовано вибір локальної СКБД SQLite для забезпечення високої швидкодії та роботи в офлайн-режимі. Розроблено реляційну схему даних, яка включає центральну таблицю ігор та таблиці жанрів, категорій, видавців і вікових рейтингів;
- розроблено застосунок мовою Java. Створено інтуїтивно зрозумілий користувацький інтерфейс на основі Activity та фрагментної навігації. Реалізовано динамічний алгоритм формування SQL-запитів, що дозволяє користувачу комбінувати різні фільтри;
- забезпечено баланс між автономністю та мережевими можливостями. Вся текстова інформація та логіка фільтрації працюють локально. Завантаження постерів ігор та переходи до офіційних магазинів виконуються асинхронно за наявності інтернет-з'єднання;
- проведено випробування системи. Воно підтвердило стабільність роботи застосунку, коректність вибірки даних за складними критеріями та відповідність системи сформованим на етапі проектування функціональним та нефункціональним вимогам.

Розроблена комп'ютерна система повністю готова до використання як інструмент для пошуку ігрового контенту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Adams E. Fundamentals of Game Design. 3rd ed. Berkeley : New Riders, 2014. 560 p.
2. Elden Ring. URL: <https://www.bandainamcoent.com/games/elden-ring> (дата звернення: 24.04.2026).
3. Get the Ultimate Cyberpunk 2077 Experience. URL: <https://www.cyberpunk.net> (дата звернення: 24.04.2026).
4. Baldur's Gate 3. URL: <https://baldursgate3.game/> (дата звернення: 24.04.2026).
5. S.T.A.L.K.E.R. 2. URL: <https://www.stalker2.com/uk> (дата звернення: 24.04.2026).
6. Weichbroth P. Usability testing of mobile applications: a methodological framework. Applied Sciences. 2024. Vol. 14, No. 5. P. 1-27.
7. Genshin Impact. URL: <https://genshin.hoyoverse.com> (дата звернення: 25.04.2026).
8. Marvel Snap. URL: <https://www.marvelsnap.com> (дата звернення: 25.04.2026).
9. Dune: Imperium. URL: <https://www.direwolfdigital.com/dune-imperium/> (дата звернення: 25.04.2026).
10. Wingspan. URL: <https://stonemaiergames.com/games/wingspan/> (дата звернення: 26.04.2026).
11. Ark Nova. URL: <https://boardgamegeek.com/boardgame/342942/ark-nova> (дата звернення: 25.04.2026).
12. PEGI Age Ratings. URL: <https://pegi.info/page/pegi-age-ratings> (дата звернення: 26.04.2026).
13. Офіційний веб-сайт Microsoft Xbox. URL: <https://www.xbox.com/uk-UA/> (дата звернення: 26.04.2026).
14. PlayStation 5. URL: <https://www.playstation.com/uk-ua/ps5/> (дата

звернення: 26.04.2026).

15. Steam Store. URL: <https://store.steampowered.com/> (дата звернення: 26.04.2026).

16. Epic Games Store. URL: <https://store.epicgames.com/en-US> (дата звернення: 27.04.2026).

17. GOG Galaxy 2.0. URL: <https://www.gog.com/galaxy> (дата звернення: 27.04.2026).

18. Ubisoft Connect. URL: <https://www.ubisoft.com/en-gb/ubisoft-connect> (дата звернення: 27.04.2026).

19. Electronic Arts App. URL: <https://www.ea.com/ea-app> (дата звернення: 27.04.2026).

20. Програма Xbox для ПК. URL: <https://www.xbox.com/uk-UA/apps/xbox-app-on-pc> (дата звернення: 27.04.2026).

21. Google Play Store. URL: <https://play.google.com/store/apps?hl=uk> (дата звернення: 27.04.2026).

22. MiniReview - Game Reviews. URL: <https://play.google.com/store/apps/details?id=minireview.best.android.games.reviews> (дата звернення: 27.04.2026).

23. IGDB. URL: <https://www.igdb.com/> (дата звернення: 27.04.2026).

24. GameTrack . URL: <https://gametrack.app/> (дата звернення: 27.04.2026).

25. RAWG Video Game Database. URL: <https://rawg.io/> (дата звернення: 27.04.2026).

26. GamerPower. URL: <https://www.gamerpower.com/> (дата звернення: 27.04.2026).

27. Horstmann C. S. Core Java. Volume I: Fundamentals. 12th ed. Hoboken : Pearson Education, 2022. 912 p.

28. Android Developers. Guide to app architecture. URL: <https://developer.android.com/topic/architecture> (дата звернення: 28.04.2026).

29. Glide. URL: <https://github.com/bumptech/glide> (дата звернення: 28.04.2026).

30. Haldar S. SQLite Database System Design and Implementation. Sunnyvale : Sibsankar Haldar, 2011. 286 p.
31. Cobley P., Geneste G. Understanding Realm Database Basics. MSAB, 2026. 210 p.
32. Murphy M. L. Elements of Android Room. CommonsWare, 2021. 264 p.

ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ

17 слайдів

РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ПОШУКУ ІГОР

дипломна робота бакалавра

Виконала: Анастасія СКОВОРОДНІКОВА
Студентка КНТ-512

Керівник: Степан СКРУПСЬКИЙ
Доцент, к.т.н.

Рисунок А.1 – Слайд 1

2

Предмет – комп'ютерна система пошуку ігор, реалізована у вигляді Android-застосунку з локальною реляційною базою даних та гнучкою системою фільтрації.

Мета – підвищення зручності та ефективності пошуку ігрового контенту (десктопних, мобільних та настільних ігор) шляхом створення спеціалізованого застосунку, незалежного від алгоритмів комерційних рекомендацій та постійного підключення до мережі Інтернет.

Новизна – поєднання автономного зберігання структурованої інформації про ігри різних платформ у локальній базі даних мобільного пристрою з мінімалістичним інтерфейсом. Це забезпечує чистий пошук без нав'язливих маркетингових компонентів.

Практична цінність - можливість використання розробленої системи як швидкого та надійного офлайн-довідника для індивідуального підбору ігор за заданими критеріями.

Рисунок А.2 – Слайд 2

3

Апаратні вимоги системи

Параметр	Мінімально	Рекомендовано
SOC	ARMv8-A з 4 ядрами Cortex-A53 від 1.6 ГГц	Cortex-A55 + Cortex-A76 з частотою 2.0+ ГГц
Оперативна пам'ять	≥ 4 ГБ RAM	≥ 6 ГБ RAM
Вбудована пам'ять	≥ 500 МБ вільного простору	≥ 2048 МБ вільного простору
GPU	Adreno 506 / Mali-G52	Adreno 618+ / Mali-G57+
Роздільна здатність екрану	≥ 1280×720, діагональ ≥ 4.7"	≥ 1920×1080, діагональ ≥ 6", щільність ≥ 300 ppi
Мережеве з'єднання	3G з пропускнуою здатністю ≥ 1 Мбіт/с	4G LTE або Wi-Fi 802.11n/ac/ax

Рисунок А.3 – Слайд 3

4

ER-модель бази даних системи

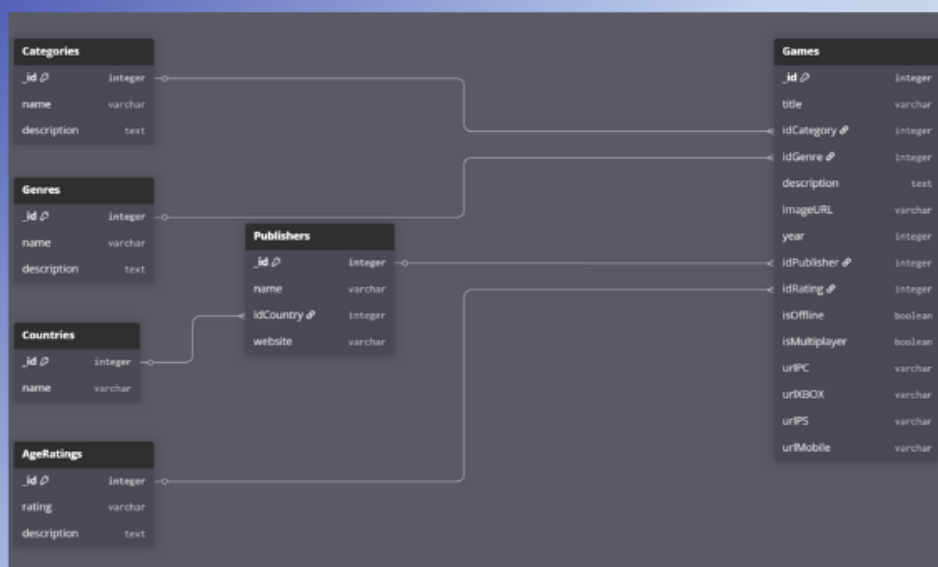


Рисунок А.4 – Слайд 4

5

Діаграма варіантів використання

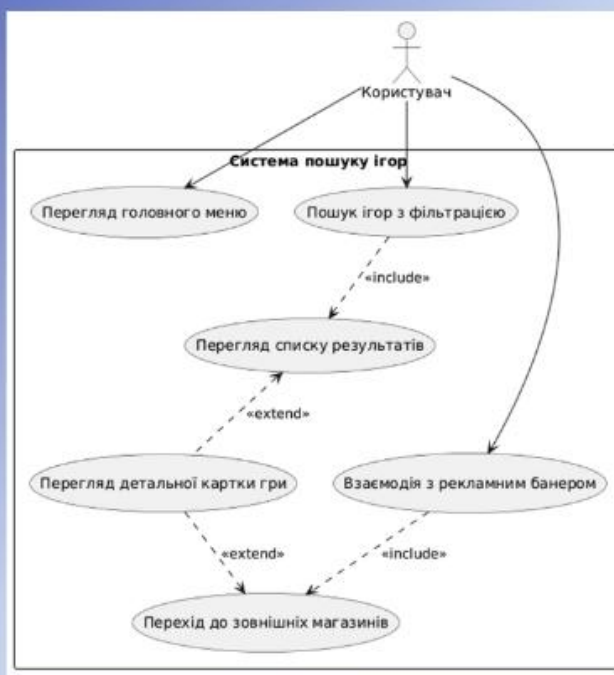


Рисунок А.5 – Слайд 5

6

Прототип (low-fidelity wireframe) застосунку системи

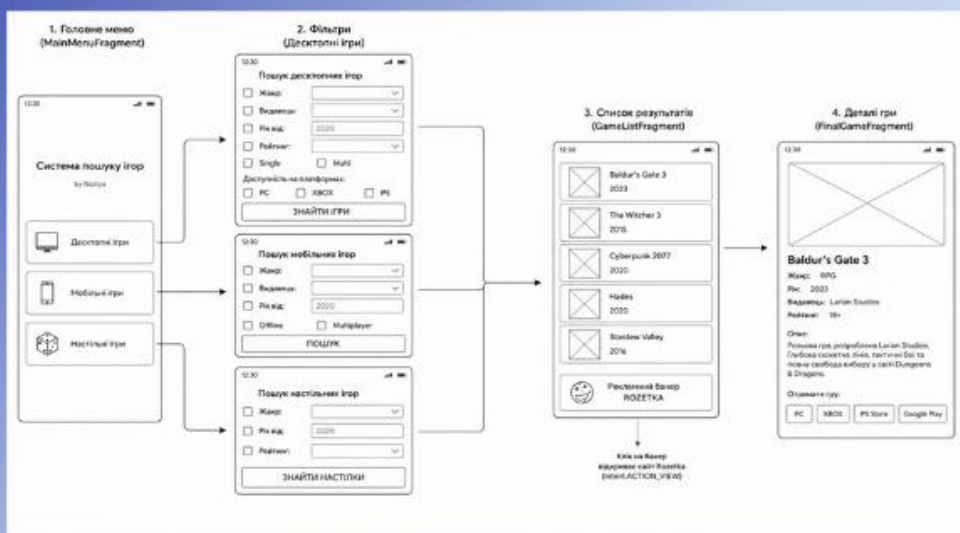


Рисунок А.6 – Слайд 6

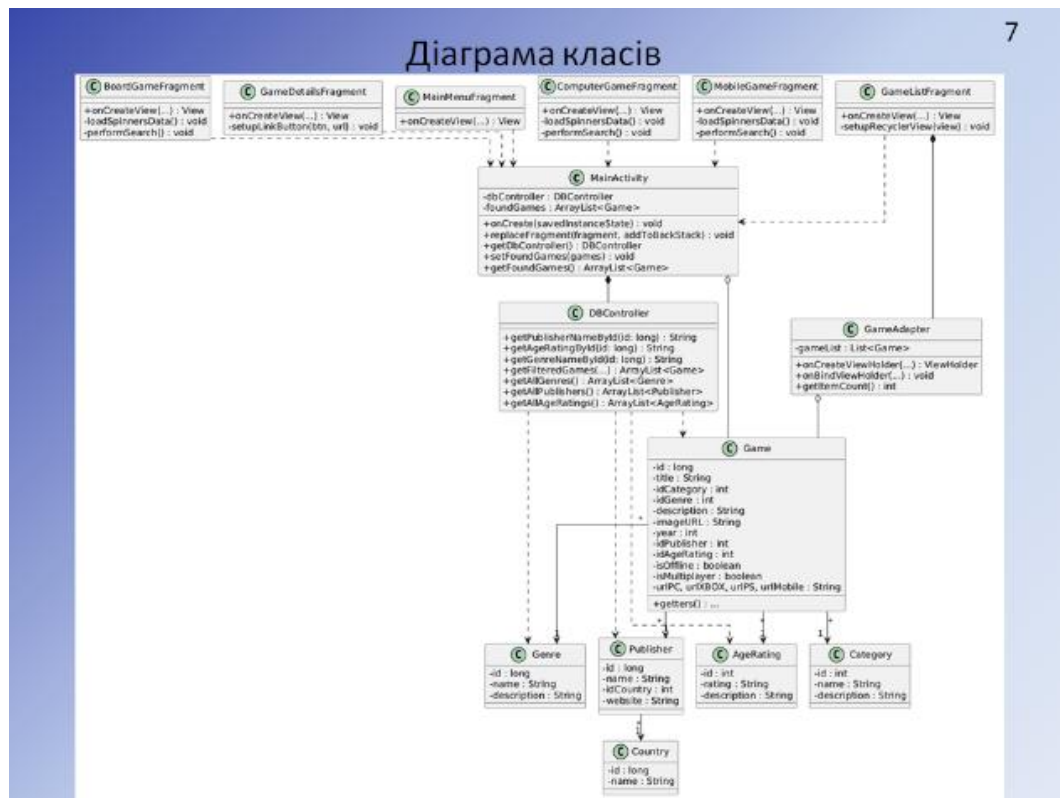


Рисунок А.7 – Слайд 7

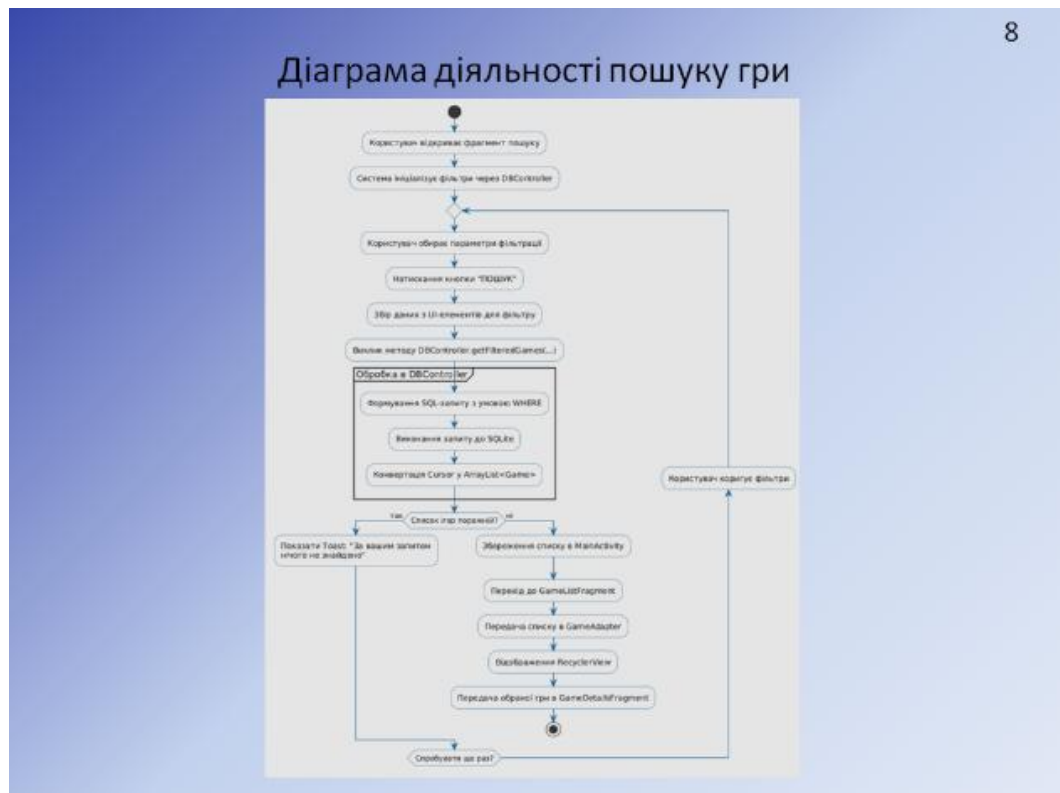


Рисунок А.8 – Слайд 8

9

Діаграма послідовностей зміни екранів

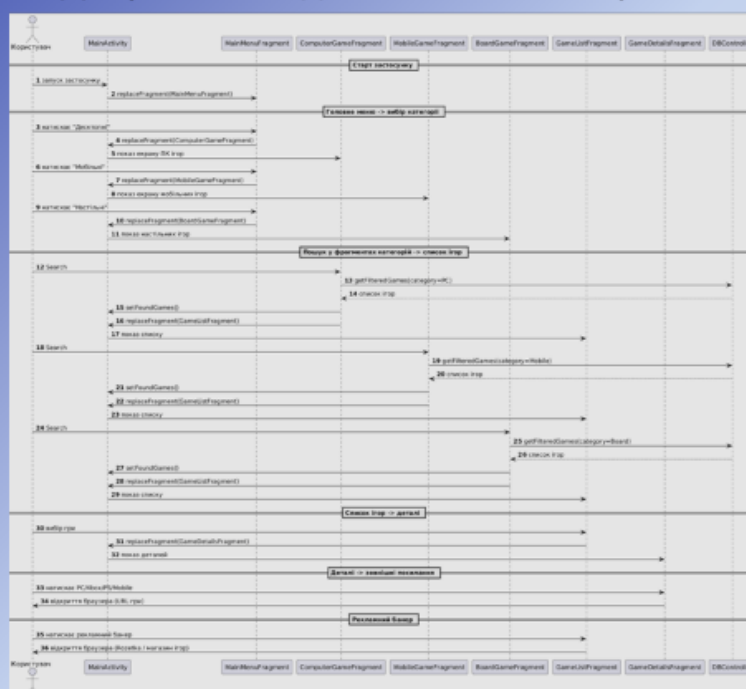


Рисунок А.9 – Слайд 9

10

Випробування системи. База даних

Tables	name	seq
AgeRatings	1	8
android_metadata	2	3
Categories	3	9
Countries	4	4
Games	5	8
Genres	6	20
Publishers	7	
sqlite_sequence		

Tables	_id	name	description
AgeRatings	1	Computer Game	Ігри для персональних комп'ютерів та ігрових консолей...
android_metadata	2	Mobile Game	Сучасні мобільні проекти для платформи Android
Categories	3	Board Game	Популярні настільні ігри з високим рейтингом BGG
Countries	4		
Games			
Genres			
Publishers			
sqlite_sequence			

Tables	_id	rating	description
AgeRatings	1	PEGI 3	Для будь-якого віку
android_metadata	2	PEGI 12	Від 12 років: помірне насильство
Categories	3	PEGI 16	Від 16 років: реалістичне насильство
Countries	4	PEGI 18	Тільки для дорослих: жорсткий контент
Games	5		
Genres			
Publishers			
sqlite_sequence			

Рисунок А.10 – Слайд 10

Рисунок А.11 – Слайд 11

12

Рисунок А.12 – Слайд 12

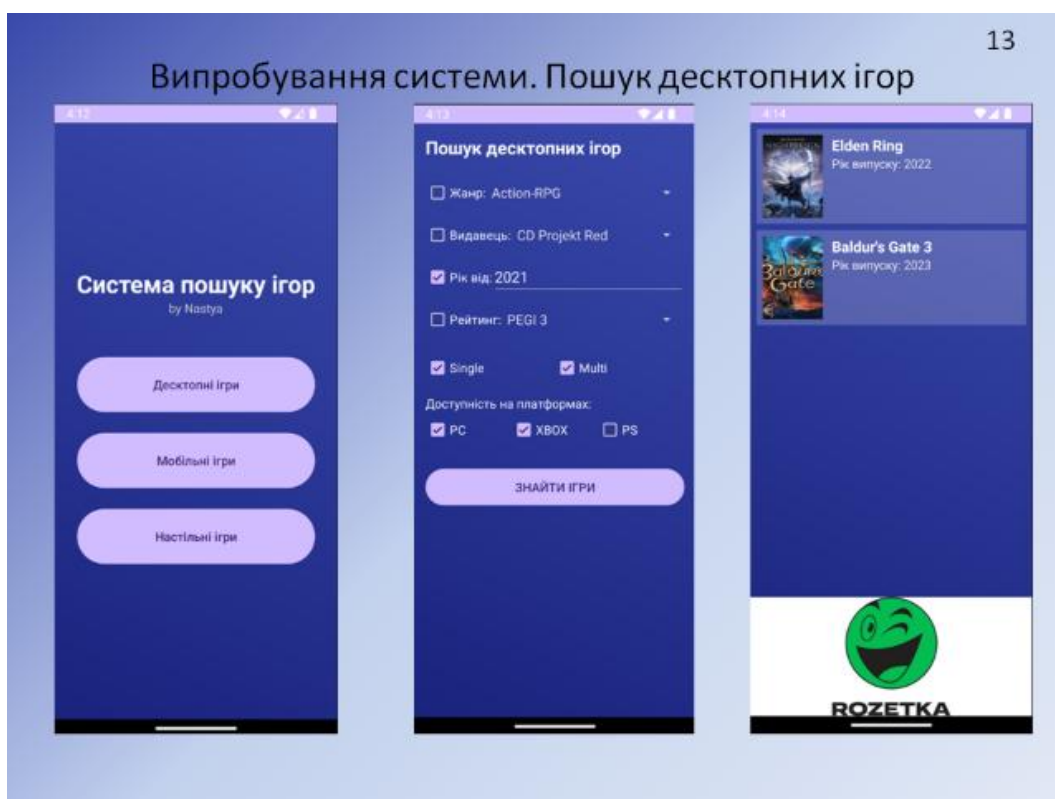


Рисунок А.13 – Слайд 13

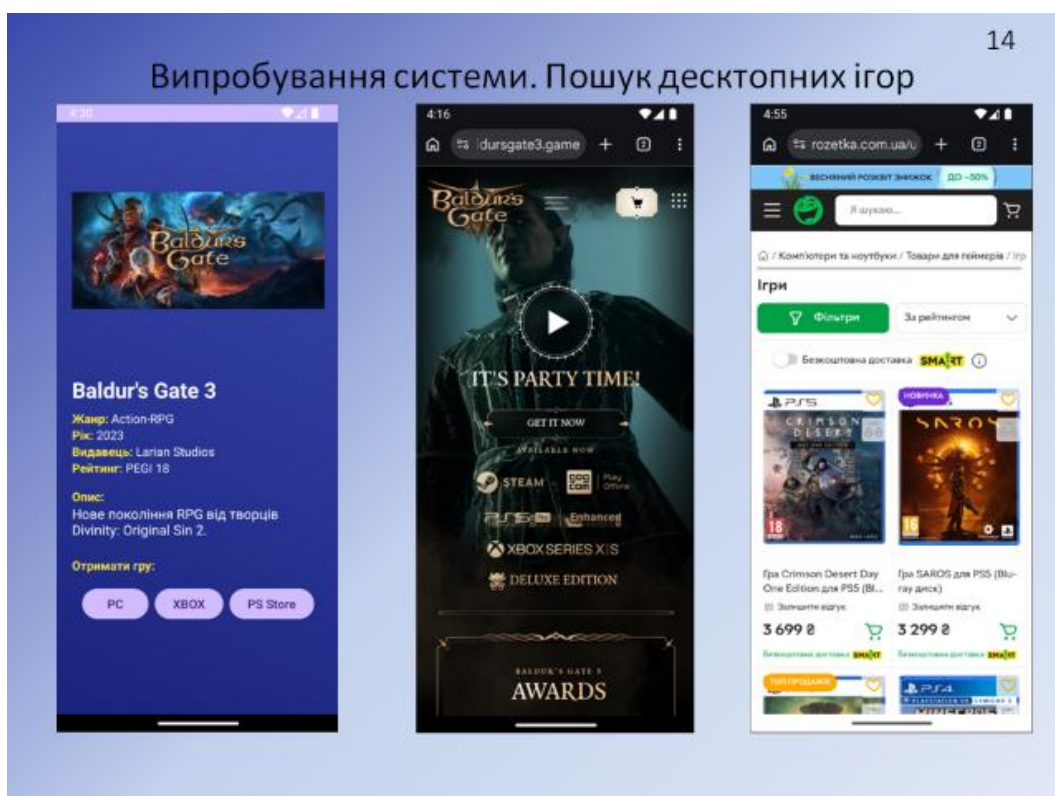


Рисунок А.14 – Слайд 14

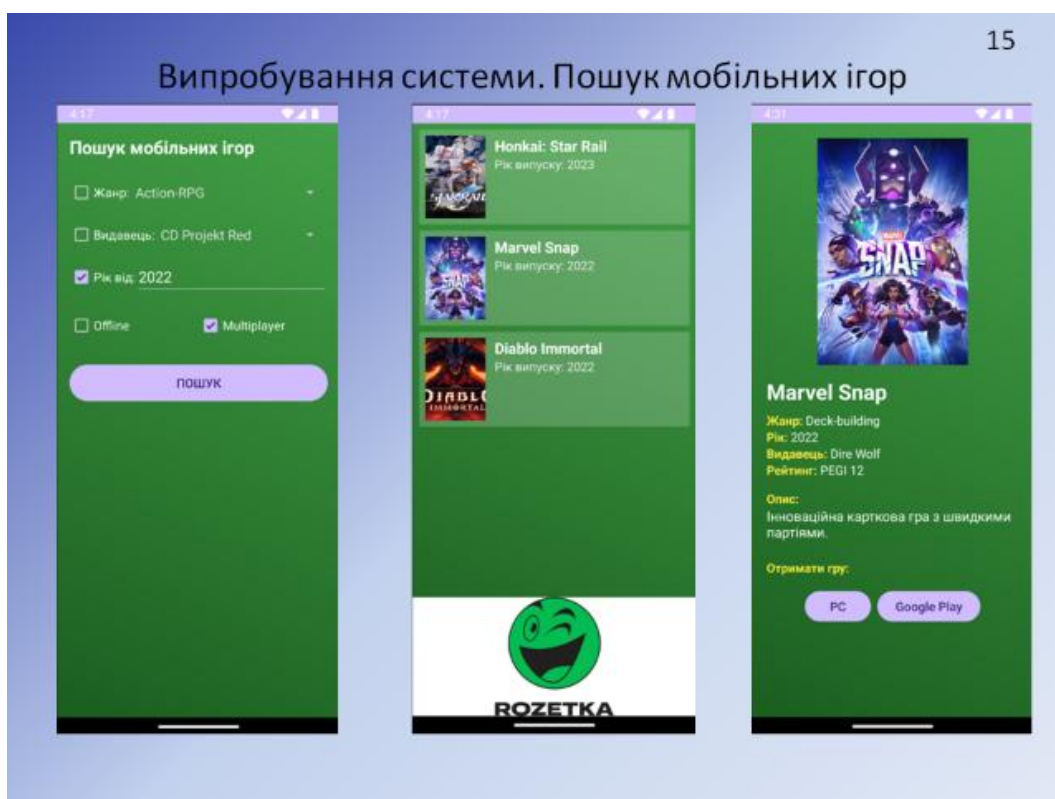


Рисунок А.15 – Слайд 15

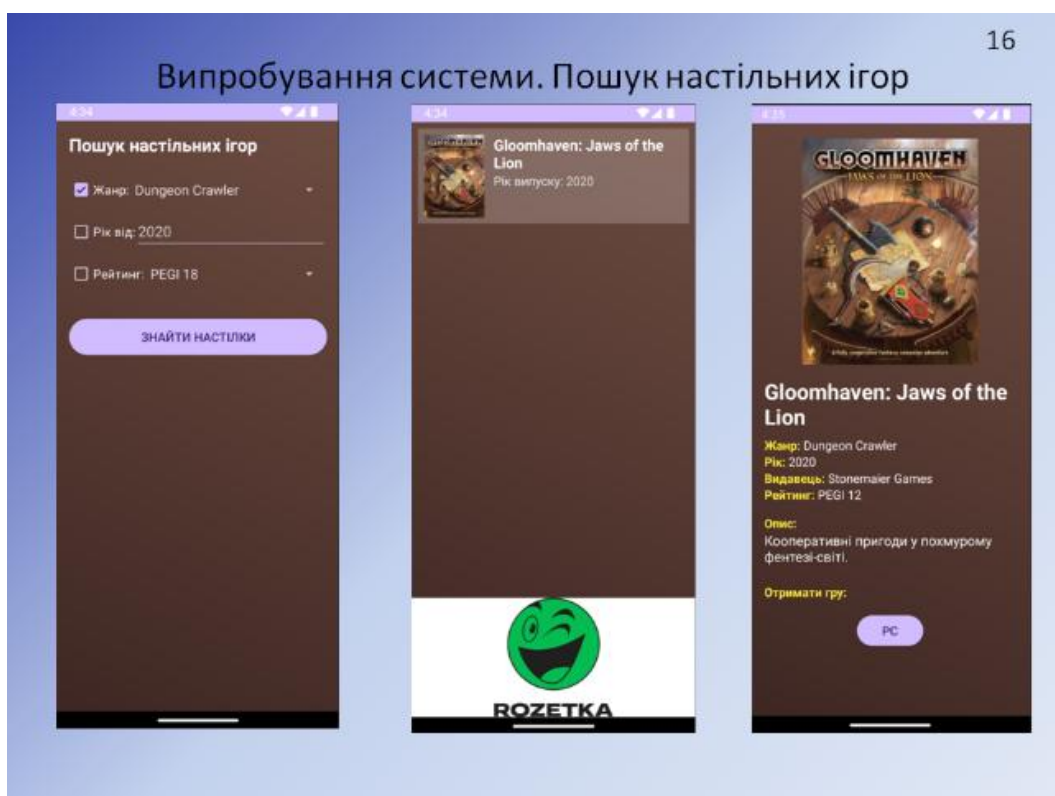


Рисунок А.16 – Слайд 16

Висновки

17

1. Сучасні цифрові магазини перевантажені комерційними рекомендаціями та не надають універсального інструменту для чистого пошуку ігор різних категорій;
2. Спроектовано архітектуру та базу даних системи;
3. Виконана розробка основних артефактів системи.
4. Розроблено мобільний застосунок для пошуку ігор;
5. Забезпечено баланс між автономністю та мережевими можливостями;
6. Проведено випробування системи, яке підтвердило відповідність поставленим задачам.

Рисунок А.17 – Слайд 17