Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система для персоналізації досвіду гравців у настільні ігри. Серверна частина
(тема)

Виконав:
здобувач _4_ року навчання
групи ПЗПІ-21-6

Владислав ЦВИК
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник проф. Зоя ДУДАР
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Цвику Владиславу Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система для персоналізації досвіду гравців у настільні ігри. Серверна частина
Затверджена наказом по університету від 19.05.2025р. № 397 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 12.06.2025
3. Вихідні дані до роботи Розробити серверну частину програмної системи для персоналізації досвіду гравців у настільні ігри з використанням технологій та мов програмування C#, .NET, ASP.NET Core та СУБД MS SQL Server
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	22.05.2025	*виконано*
2	Створення специфікації ПЗ	25.05.2025	*виконано*
3	Проєктування ПЗ	27.05.2025	*виконано*
4	Розробка ПЗ	01.06.2025	*виконано*
5	Тестування ПЗ	02.06.2025	*виконано*
6	Оформлення пояснювальної записки	04.06.2025	*виконано*
7	Підготовка презентації та доповіді	05.06.2025	*виконано*
8	Попередній захист	06.06.2025	*виконано*
9	Нормоконтроль, рецензування	07.06.2025	*виконано*
10	Здача роботи у електронний архів	08.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	09.06.2025	*виконано*

Дата видачі завдання « 19 » « травня » 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ ____проф. Зоя ДУДАР____
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 104 стор., 20 рис., 4 табл., 16 джерел, 5 додатків.

НАСТІЛЬНІ ІГРИ, РЕКОМЕНДАЦІЙНА СИСТЕМА, ASP.NET CORE, C#, ENTITY FRAMEWORK CORE, MS SQL SERVER, .NET, OAUTH2, REACT.JS, REST API,

Об’єктом розробки є програмна система для персоналізації досвіду гравців у настільні ігри.

Метою розробки є створення повнофункціональної веб-платформи, що дозволяє користувачам брати участь у подіях, присвячених настільним іграм, отримувати персоналізовані рекомендації, керувати власним профілем гравця та відстежувати історію участі у подіях.

Метод рішення – середовище розробки Visual Studio, мова програмування C#, платформа .NET.

У результаті розробки спроєктовано загальну архітектуру системи, описано архітектурні рішення для серверної частини, а також реалізовано серверну частину програмної системи для персоналізації досвіду гравців у настільні ігри.

ABSTRACT

BOARD GAMES, RECOMMENDATION SYSTEM, ASP.NET CORE, C#, ENTITY FRAMEWORK CORE, MS SQL SERVER, .NET, OAUTH2, REACT.JS, REST API,

The object of the development is a software system for personalizing the experience of board game players.

The goal of the development is to create a fully functional web platform that allows users to participate in board game-related events, receive personalized recommendations, manage their player profiles, and track their participation history in events.

Solution methods – integrated development environment Visual Studio, programming language C#, .NET platform.

As a result of the development, the overall system architecture was designed, architectural solutions for the server side were described, and the server part of the software system for personalizing the experience of board game players was implemented.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

PWA – Progressive Web Application

SPA – Single Page Application

BGG – Board Game Geek

API – Application Programming Interface

DTO – Data Transfer Object

ВСТУП

У сучасному суспільстві інтерес до настільних ігор не лише не зменшується, а навпаки — стрімко зростає, набуваючи нових форм і форматів. Настільні ігри об’єднують людей різного віку та інтересів, сприяють розвитку критичного мислення, логіки та соціальної взаємодії. В останній час особливо стрімко збільшується кількість молодих людей, які полюбляють грати у різноманітні настільні ігри.

Актуальність обраної теми зумовлена зростанням популярності настільних ігор серед та запитом на інструменти, які можуть зробити ігровий досвід більш індивідуалізованим і зручним. Хоча настільні ігри традиційно не потребують складного обладнання, у сучасних умовах зростає потреба у допоміжних інструментах, які могли покращити досвід гравців, зробити його зручнішим та простішим. Все більше молоді захоплюється настільними іграми і саме ця категорія людей потребує особливої уваги. Цифрові рішення можуть допомогти гравцям швидше знаходити цікаві ігри, отримувати рекомендації на основі власних уподобань та досвіду, а також знайомитися з іншими учасниками спільноти. Це сприяє не лише покращенню ігрового досвіду, а й соціалізації, що є важливим для багатьох користувачів.

Тема даної роботи — розробка програмної системи для персоналізації ігрового досвіду гравців у настільні ігри. Така система має на меті не лише автоматизувати облік ігрових подій, а й забезпечити аналітику, рекомендації щодо ігор, формування профілю гравця та полегшення організації ігор у компаніях з різною кількістю учасників і рівнем досвіду.

Метою цієї роботи є створення зручної програмної системи, яка дозволить гравцям у настільні ігри зберігати результати, отримувати аналітику, рекомендації, переглядати особисту статистику та підбирати ігри відповідно до індивідуальних переваг. Передбачається, що система буде доступна у вигляді веб-застосунку з адаптивним інтерфейсом для різних пристроїв.

Для досягнення поставленої мети необхідно проаналізувати предметну область, вивчити потреби потенційних користувачів системи, визначити основні функціональні вимоги до програмного продукту, розробити архітектуру системи, реалізувати її компоненти та протестувати роботу програмного забезпечення в умовах, наближених до реального використання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Світ настільних ігор має багатовікову історію, яка розпочалась ще в Давньому Єгипті з простих ігор, таких як сенет, і досягла сучасних складних стратегічних ігор, що вимагають від гравців глибокого планування та мислення. Однак у останні десятиліття спостерігається значний розвиток індустрії настільних ігор [1], особливо в контексті популяризації цієї форми дозвілля серед різних вікових груп.

Сучасні настільні ігри варіюються від класичних (шахи, нарди) до новітніх варіантів, таких як "Катан", "Діксіт", "Каркассон", які зазвичай вимагають взаємодії між учасниками та стратегічного підходу. Важливим аспектом є швидке зростання популярності настільних ігор серед молоді, для яких важливо не лише грати, а й мати можливість обмінюватися враженнями, брати участь у змаганнях та соціалізуватися через спільний інтерес. Висновок про активний розвиток галузі можна зробити з графіків. На рисунку 1.1 наведено графік кількості добровільних вкладень на платформі «Kickstarter» у створення нових настільних ігор.

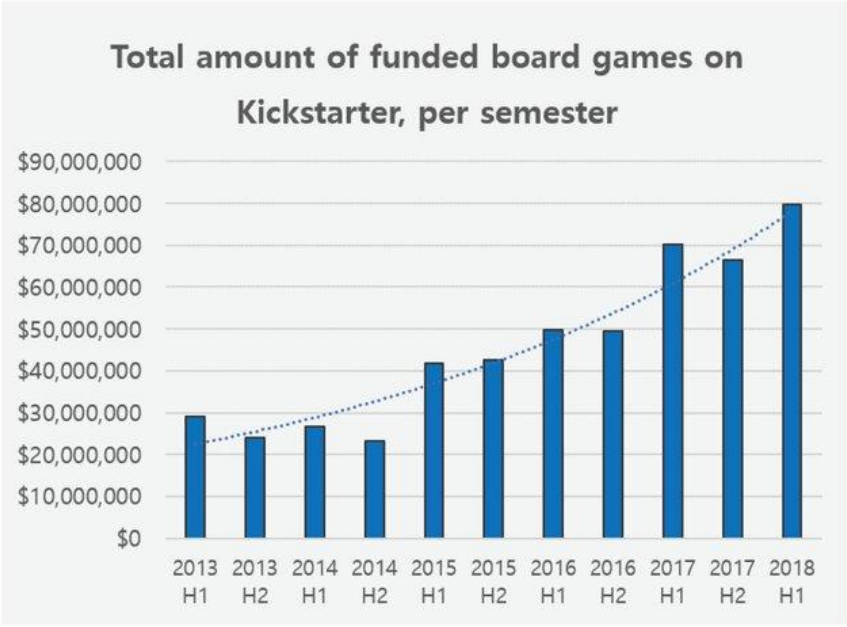


Рисунок 1.1 – Динаміка кількості вкладень у настільні ігри через краудфандингову платформу «Kickstarer» [1]

Зі збільшенням кількості гравців ігор з’являються нові форми подій, змагань, відкриваються нові ігрові клуби. Все це сприяє розвитку цієї галузі. Крім того, спостерігається інтеграція технологій у галузь настільних ігор, що дозволяє покращити взаємодію між учасниками, автоматизувати певні організаційні процеси, а також покращити ігровий досвід.

1.2 Виявлення та вирішення проблем

У той час як у цифрових відеоіграх інтеграція цифрових технологій, рекомендаційних систем, а також інших інструментів давно є стандартом, сфера настільних ігор лише починає впроваджувати подібні технології. Відсутність таких рішень гальмує розвиток спільнот, зменшує залученість нових учасників та ускладнює вибір гри під конкретну компанію чи ситуацію.

Однією з основних проблем є відсутність повноцінних рекомендаційних інструментів для персоналізації ігрового досвіду. У той час як у цифрових відеоіграх такі рішення існують повсюдно, а налаштування персонажів та адаптацію контенту вже давно стала стандартом, настільні ігри ще не забезпечують таких можливостей. Відсутність персоналізованих рекомендацій значно ускладнює вибір ігор, знижує зацікавленість гравців і позбавляє їх можливості отримувати максимально приємний та релевантний досвід. Гравцям доводиться обмінюватися досвідом та іграми вручну без застосування програмних систем.

Також значним обмеженням є відсутність єдиної онлайн-платформи для соціальної взаємодії гравців в Україні. Без такої платформи гравці не мають змоги легко знаходити однодумців, обмінюватися досвідом, планувати ігрові зустрічі або брати участь у подіях. Це знижує залученість нових учасників до ігрової спільноти, особливо серед молоді, яка віддає перевагу цифровим рішенням для організації своєї діяльності та соціалізації. Існуючі платформи часто обмежуються лише обміном інформацією або продажом ігор, а не створенням можливостей для активної взаємодії та організації ігрових заходів. Усі новини щодо ігрових подій учасники спільноти дізнаються переважно зі сторінок у соціальних мережах різноманітних ігрових клубів чи інших груп.

1.3 Аналіз існуючих аналогів

У сфері настільних ігор існує декілька ключових платформ, що пропонують різноманітні функції: від каталогів ігор і відгуків до організації спільнот чи планування подій. Однак більшість з них зосереджені на окремих аспектах взаємодії, не охоплюючи повного циклу ігрового досвіду — від персональних рекомендацій до соціалізації та участі у заходах. Для аналізу було обрано найпопулярніші рішення для вирішення проблеми створення подібної програмної системи для гравців у настільні ігри, які є популярними як у міжнародній спільноті, так і в Україні.

Board Game Geek (BGG) є одним із найбільших і найвідоміших онлайн-ресурсів, присвячених настільним іграм [2]. Він надає обширну базу даних ігор, включаючи рецензії, оцінки, форуми для обговорення та інструменти для управління колекцією ігор. Користувачі можуть відстежувати свої ігри, планувати ігрові сесії та знаходити нові ігри, які можуть їм сподобатися. Інтерфейс сервісу відображено на рисунку 1.2.

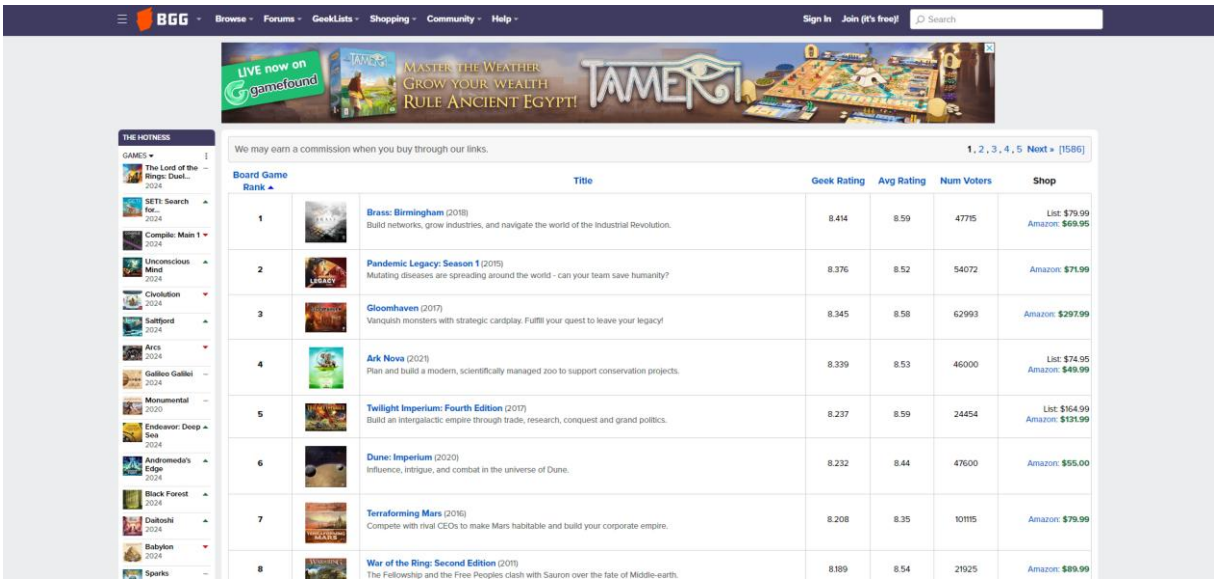


Рисунок 1.2 – Інтерфейс сервісу Board Game Geek [2]

Однією з сильних сторін BGG є його спільнота. Велика та активна спільнота користувачів означає, що на платформі завжди можна знайти детальні огляди та обговорення будь-якої ігри. Крім того, сайт має велику кількість інструментів пошуку ігор та їх купівлі.

Проте, є й недоліки. Інтерфейс користувача може здатися перевантаженим і складним для новачків, оскільки на сайті присутня велика кількість інформації та функціональних можливостей. Це може зробити перші враження від використання BGG трохи пригнічуючими. Крім того, хоча сайт і пропонує рекомендації щодо ігор, вони часто базуються на загальних оцінках і відгуках, а не на персоналізованому підході, який би враховував унікальні ігрові переваги та досвід конкретного користувача. Також у сервісі відсутня зручна система пошуку гравців для ігор та клубів настільних ігор.

Таким чином, хоча Board Game Geek є важливим ресурсом для спільноти любителів настільних ігор, існують можливості для покращення, особливо у сферах удосконалення користувацького інтерфейсу та розробки більш персоналізованих рекомендаційних систем.

Сервіс Try These Games є онлайн-платформою, яка зосереджена на пошуку і рекомендаціях настільних ігор [3]. Вона дозволяє користувачам швидко знаходити ігри на основі різних критеріїв, таких як кількість гравців, вік гравців, тривалість гри та інші параметри. Це робить сервіс корисним інструментом для тих, хто шукає нові ігри для певних ситуацій або настроїв. Також, можна просто ввести гру, для того, щоб отримати аналогічні ігри. Усі настільні ігри оцінюються за різними числовими критеріями у вигляді зірочок, що є наглядним відображенням для потенційних гравців. Інтерфейс сервісу відображено на рисунку 1.3.

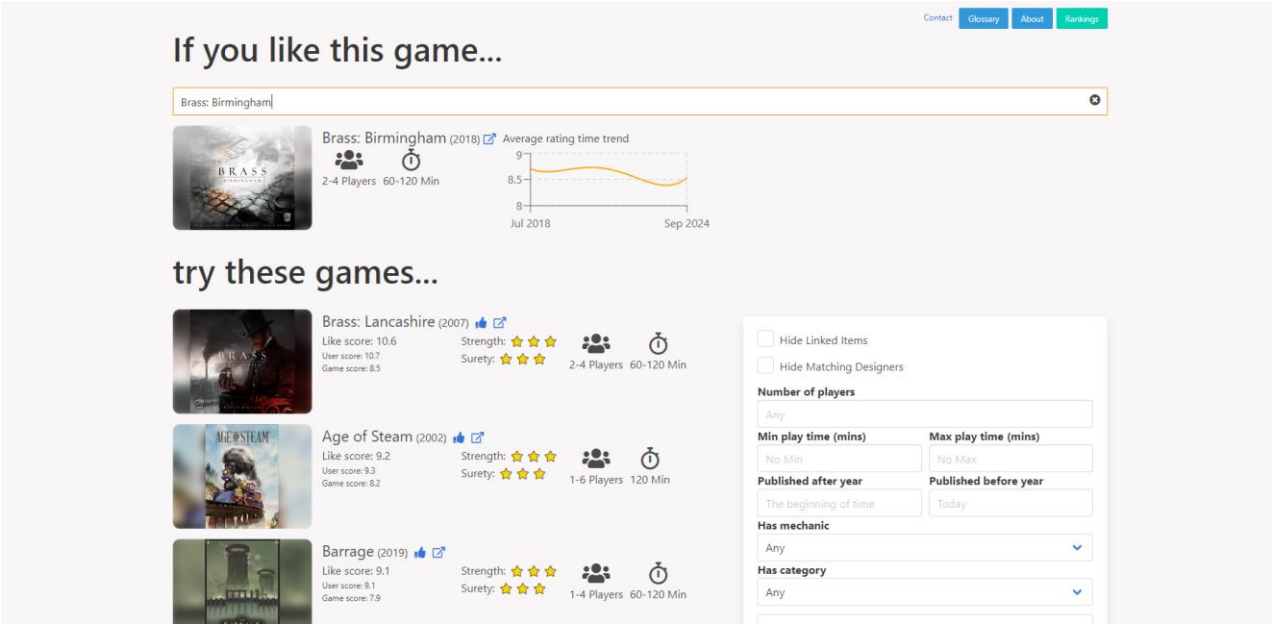


Рисунок 1.3 – Інтерфейс сервісу Try These Games [3]

Однією з сильних сторін Try These Games є його простота та інтуїтивно зрозумілий інтерфейс, що робить пошук ігор швидким і легким. Це може бути особливо привабливим для новачків у світі настільних ігор, які можуть відчувати себе перевантаженими більш складними платформами, як Board Game Geek.

Проте, сервіс має і свої обмеження. На відміну від комплексних платформ, таких як Board Game Geek, Try These Games не пропонує додаткових функцій, таких як обговорення, рецензії, управління колекцією ігор або організація ігрових заходів. Також, хоча сервіс і дозволяє знаходити ігри, він не надає глибокої персоналізації рекомендацій, яка б враховувала попередній досвід користувача або детальні ігрові переваги.

Враховуючи це, Try These Games може бути корисним як допоміжний інструмент для швидкого пошуку настільних ігор, але для більш досвідчених гравців або тих, хто шукає більш глибокі та персоналізовані ігрові рекомендації, може знадобитися більш комплексна платформа.

Платформа Geeker є відносно новою українською розробкою, яка сьогодні активно просувається серед поціновувачів настільних ігор в Україні та поза її межами [4]. Ця платформа орієнтована на тих, хто бажає швидко знайти гравців

для проведення подій, ігрових сесій та турнірів. Організатор події може створити гру, а усі бажаючі – доєднатися до неї. Інтерфейс відображено на рисунку 1.4.

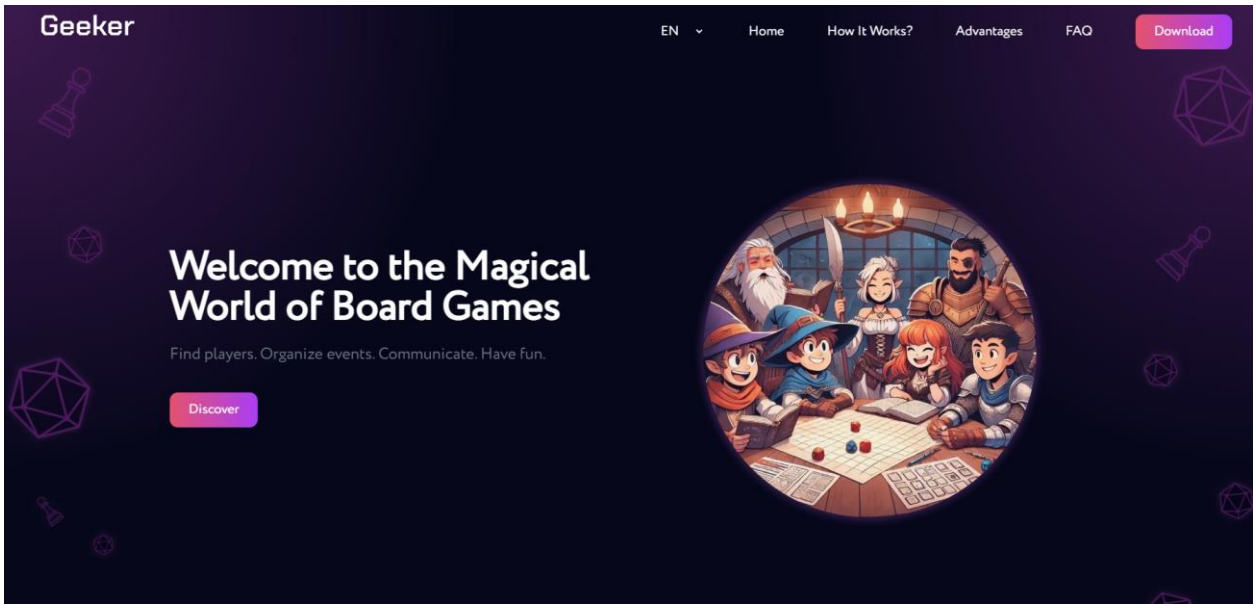


Рисунок 1.4 – Інтерфейс сервісу Geeker [4]

Цей сервіс має свої переваги та недоліки. По-перше, значною перевагою є наявність зручного мобільного застосунку. Доступ до сервісу можна отримати як з браузеру, так і через мобільний застосунок для Android та IOS. Це надає можливість використовувати платформу будь-де. По-другое, сервіс надає зручний пошук бажаних ігор і подій для того, щоб не витрачати багато часу на цей процес.

Серед недоліків можна виділити відсутність аналізу місцезнаходження користувача та пошук найближчих до нього подій, а також відсутність просунутого функціоналу профілю гравця. На відміну від BGG, Geeker не має такої обширної бази даних ігор та детальних рецензій на ігри, бо не спеціалізується на цьому.

Враховуючи проведений аналіз конкурентів можна встановити основні критерії порівняння існуючих платформ та визначити оптимальні показники для нашої системи, що проєктується. Результат дослідження відображено у таблиці 1.1.

Таблиця 1.1 – Результат аналізу конкурентів (таблиця виконана самостійно)

Аналізований показник	Наша платформа	Board Game Geek	Try These Games	Geeker
Інтерактивність	Висока	Середня	Низька	Середня
Кількість ігор	Велика	Дуже велика	Середня	Середня
Персоналізація рекомендацій	Висока	Низька	Низька	Низька
Спільнота	Активна	Дуже активна	Не активна	Не активна
Огляди та аналізи ігор	Так	Так	Ні	Ні
Доступність	Висока	Висока	Висока	Висока
Підтримка багатомовності	Так	Так	Ні	Частково
Можливість організації подій	Так	Частково	Ні	Так

1.4 Постановка задачі

Для вирішення проблем потенційних користувачів необхідно розробити програмну систему, яка б створила єдину платформу для учасників спільноти настільних ігор та дозволила б їм взаємодіяти між собою, шукати релевантні ігрові події, отримувати персональні пропозиції щодо подій і настільних ігор на основі їх вподобань, які формуються у профілі гравця на основі його попередніх ігрових сесій, доданих улюблених ігор, місцезнаходження, а також інших факторів.

Основною метою розробки є створення зручної та інтуїтивної веб-платформи, яка дозволятиме користувачам переглядати календар подій, створювати власні ігрові зустрічі, приєднуватися до подій інших гравців, а також мати доступ до історії попередніх ігрових сесій. У системі повинен бути реалізований модуль управління списком настільних ігор, що включає опис гри, жанр, кількість гравців, середню тривалість партії, рівень складності тощо.

Важливою функцією платформи стане механізм персональних рекомендацій настільних ігор та ігрових подій. Він має працювати на основі вподобань користувача. Спочатку має бути реалізовано алгоритм підбору настільних ігор на основі обраних улюблених ігор. В подальшому він може враховувати такі фактори,

як участь в попередніх подіях, ігрових оцінках, геолокації, з можливим застосуванням поведінкових моделей. Це дозволить пропонувати релевантні ігри та події саме тим користувачам, яким вони ймовірно будуть цікаві.

Кожен користувач повинен мати власний профіль, де зберігається інформація про його ігрову активність, улюблені ігри, друзів зі спільноти, рейтинг, а також заплановані події.

У перспективі така платформа має стати місцем, що об’єднує однодумців, дозволяє знаходити нові знайомства, а також гарно проводити час. Це буде сприяти зростанню та розвитку спільнот настільних ігор, яка на даний момент активно популяризується.

1.5 Цільова аудиторія

Цільовою аудиторією розроблюваного застосунку є широке коло користувачів, які цікавляться настільними іграми, незалежно від рівня досвіду. Серед них можна виділити як досвідчених гравців, що регулярно беруть участь у ігрових подіях та мають сформовані вподобання, так і новачків, які тільки бажають долучитися до спільноти.

Особливу увагу приділено молоді (вікова категорія приблизно 16–35 років), яка активно використовує цифрові технології у повсякденному житті, шукає нові способи соціалізації, нові хобі та охоче приймає участь у тематичних зустрічах. Ця категорія користувачів зацікавлена у зручному пошуку ігрових подій за інтересами та геолокацією, формуванні нових знайомств та отриманні персональних рекомендацій.

Також серед потенційних користувачів можна виділити організаторів ігрових заходів (геймmастерів, представників клубів або кафе з настільними іграми), для яких платформа стане інструментом для популяризації подій, пошуку учасників та комунікації з ними.

1.6 Монетизація

Для розвитку платформи передбачено реалізацію декількох моделей монетизації, які орієнтовані як на кінцевих користувачів (B2C), так і на бізнес-клієнтів (B2B).

Перший напрям — це платні підписки для користувачів (B2C). Звичайні гравці можуть оформити доступну підписку, яка відкриватиме персоналізовані рекомендації настільних ігор, візуальне виділення профілю (значок, рамка тощо), а також ранній доступ до нових функцій. Для організаторів подій пропонується окрема підписка, що дозволяє просувати свої події у списку, переглядати аналітику за проведеними іграми та користуватися інструментами для управління серіями подій.

Другий напрям — взаємодія з ігровими клубами (B2B). Клуби зможуть отримати спеціальний тип облікового запису, що включає брендовану сторінку клубу у спеціальній вкладці, можливість створення подій від імені клубу з пріоритетним відображенням у результатах пошуку, а також доступ до розширеної аналітики аудиторії.

У перспективі планується запровадження комісії за платні події. Організатори зможуть приймати оплату за участь безпосередньо через платформу, а сервіс утримуватиме невелику комісію (3–5%). Ця функція потребує додаткових технічних засобів, тому реалізовуватиметься на наступних етапах розвитку.

Окремим напрямом монетизації є партнерство з видавцями настільних ігор, які буть зацікавлені в цьому після отримання платформою певної аудиторії. На сторінках зі списком ігор та подій передбачено місце для рекламних блоків, які можуть містити новинки, тематичні добірки або банери з посиланнями на онлайн-магазини чи краудфандингові кампанії.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Проєктована система має бути представлена веб-застосунком, доступним через сучасні браузери (Google Chrome, Microsoft Edge, Opera) на різних пристроях (комп’ютерах, планшетах, смартфонах). Система призначена для організації та участі в подіях, пов’язаних із настільними іграми, з можливістю взаємодії між користувачами, власниками ігрових клубів та адміністраторами. Основною метою платформи є спрощення пошуку настільних ігор, подій за інтересами та локацією, а також формування персоналізованих рекомендацій для користувачів. Браузерний застосунок буде являти собою SPA, який динамічно підвантажуватиме дані з серверу без необхідності оновлення сторінок. Також в подальшому передбачається створення PWA для можливості доступу до системи з мобільних пристроїв.

Функціональність реалізується відповідно до ролей користувачів, які мають різні рівні доступу до можливостей системи. У системі передбачено декілька ролей користувачів, кожна з яких має доступ до певного набору функцій відповідно до своїх прав та призначення:

- незареєстрований користувач – може переглядати загальнодоступну інформацію, ознайомлюватися з настільними іграми та запланованими подіями, але не має доступу до персоналізованих функцій і не може взаємодіяти з іншими користувачами;
- зареєстрований користувач (аутентифікований) – отримує можливість створення та редагування профілю, додавання улюблених ігор, перегляду історії участі в подіях, реєстрації на події, а також отримання персоналізованих рекомендацій; за бажанням у зареєстрованих користувачів є можливість створювати публічні або локальні події та керувати реєстраціями учасників;
- власник ігрового клубу – спеціальна роль, яка дозволяє створювати та керувати інформацією про свій ігровий клуб, а також проводити події від імені клубу;

– адміністратор (бізнес-логіки) – користувач з розширеними правами, який має змогу модерувати контент, керувати каталогом ігор та категорій, взаємодіяти з зовнішніми сервісами для оновлення даних, а також керування аккаунтами користувачів.

Усі вимоги до системи розділимо на функціональні вимоги та нефункціональні вимоги. Також додатково виділимо обов'язково реалізований функціонал для початкового релізу, а також обсяг наступних випусків.

2.1 Функціональні вимоги

Визначені функціональні вимоги до системи:

а) користувацький доступ та аутентифікація: система повинна забезпечувати безпечну автентифікацію та авторизацію для користувачів з різними ролями:
   1) реєстрація нового користувача за допомогою email та пароля;
   2) вхід до системи з використанням логіна/пароля або через обліковий запис Google.
   3) підтвердження електронної пошти після реєстрації;
   4) можливість скидання пароля через електронну пошту;
   5) вихід із системи;

б) доступ до основної інформації: система повинна забезпечувати доступ до інформаційних сторінок з основною інформацією про продукт для інформування та зацікавлення користувачів:
   1) відображення інформації про платформу та її можливості;
   2) показ обраних настільних ігор та майбутніх подій для ознайомлення;

в) профіль користувача: кожен зареєстрований користувач повинен мати персональну сторінку профіля;
   1) можливість оновлення особистих даних (нікнейм, аватар, біографія тощо);
   2) можливість встановлення місцезнаходження користувача;
   3) перегляд історії участі у подій;

4) перегляд створених подій;
5) редагування (додавання та видалення) списку улюблених ігор;
6) перегляд профілів інших користувачів;

г) каталог настільних ігор: система повинна містити каталог настільних ігор з можливістю навігації та фільтрації;
1) перегляд списку усіх настільних ігор доданих до системи;
2) фільтрація ігор за категорією, складністю, кількістю гравців, жанрами тощо;
3) перегляд детальної інформації про ігри;

д) функціонал ігрових клубів: для власників настільних клубів передбачається додатковий функціонал керування список ігрових клубів:
1) створення, редагування та видалення ігрового клубу з описом та місцезнаходженням;
2) можливість створення подій з прив’язкою до ігрового клубу;
3) перегляд списку усіх доданих ігрових клубів усіма користувачами;

е) події: система повинна надавати функціонал перегляду та керування подіями:
1) перегляд майбутніх подій усіма користувачами;
2) фільтрація подій за датою, місцем знаходження, грою, назвою тощо;
3) створення та редагування подій організаторами;
4) управління та перегляд зареєстрованих учасників на створені події;

ж) звіти та статистики: повинна надавати можливість генерації звітів пов’язаних з іграми та подіями:
1) звіт про власну ігрову активність (відвідані події та улюблені ігри);
2) звіт зі списком учасників події для організаторів;
3) статистика про найпопулярніші ігри та події адміністраторами та власниками клубів;

з) адміністрування системи: розширені функції управління платформою для адміністратора:

1) редагування списку настільних ігор, а також характеристик ігор (жанрів, механік, тем тощо);
2) можливість додавання та оновлення інформації про настільні ігри через зовнішній API BGG;
3) керування користувачами (зміна ролей, блокування акаунтів);

и) персональні рекомендації: надання рекомендацій зареєстрованим користувачам щодо подій та настільних ігор:
1) формування списку рекомендованих ігор на основі вподобань користувача;
2) сортування подій по релевантності виходячи з улюблених ігор користувача, популярності подій, місцезнаходження користувача та інших факторів.

У початковому релізі програмної системи буде реалізовано основний функціонал, необхідний для взаємодії користувачів з платформою, додавання та перегляду ігор, базові рекомендації та пошук. Проте частина розширених можливостей буде перенесена на майбутні етапи розробки. До функціоналу, який планується у наступних випусках можна віднести:

- розширене адміністрування контенту через панель адміністратора;
- модерація користувацького контенту та профілів;
- поглиблені рекомендації настільних ігор на основі взаємодії користувачів;
- система оповіщень та оновлень;
- механізм зворотного зв’язку та репортів.

2.2 Нефункціональні вимоги

Визначені нефункціональні вимоги до системи:

– система повинна забезпечувати швидке завантаження застосунку (перше завантаження SPA до 5 секунд при стандартному інтернет-з’єднанні та підвантаження інших даних до 2 секунд при запиті з серверу);

- архітектура система повинна підтримувати горизонтальне масштабування та передбачати обробку великої кількості запитів від користувачів одночасно;
- система повинна бути доступна 24/7 з мінімальним часом простою (<1% на місяць);
- уся веб взаємодія повинна відбуватися по захищеному протоколу HTTPS;
- збереження паролів повинно здійснюватися з використанням хешування;
- платформа повинна мати базовий механізм запобігання атакам типу XSS, CSRF, SQL injection;
- функціонал системи (включно з запитами на сервер) має бути доступний згідно з ролями користувачів;
- інтерфейс повинен бути інтуїтивно зрозумілим та адаптивним;
- система повинна підтримувати англійську та українську мову інтерфейсу (переклад основної текстової інформації без урахування перекладу контенту)
- повинна бути передбачена система логування основних подій та помилок.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML-проєктування ПЗ

На цьому етапі проєктування програмної системи виконується моделювання її основних структурних та поведінкових аспектів з використанням мови UML (Unified Modeling Language). Метою UML-проєктування є візуалізація ключових компонентів системи, їхніх взаємозв'язків, а також опис основних сценаріїв взаємодії користувачів із системою. У межах UML-проєктування були розроблені діаграма варіантів використання (use case diagram) та діаграма розгортання (deployment diagram).

Діаграма прецедентів відображає основні варіанти використання система та взаємодії користувачів. Створено окремі діаграми прецедентів для основних ролей у системі (незареєстрований користувач, аутентифікований користувач, власник клубу, адміністратор системи). Діаграми відображені на рисунках 3.1 – 3.4.

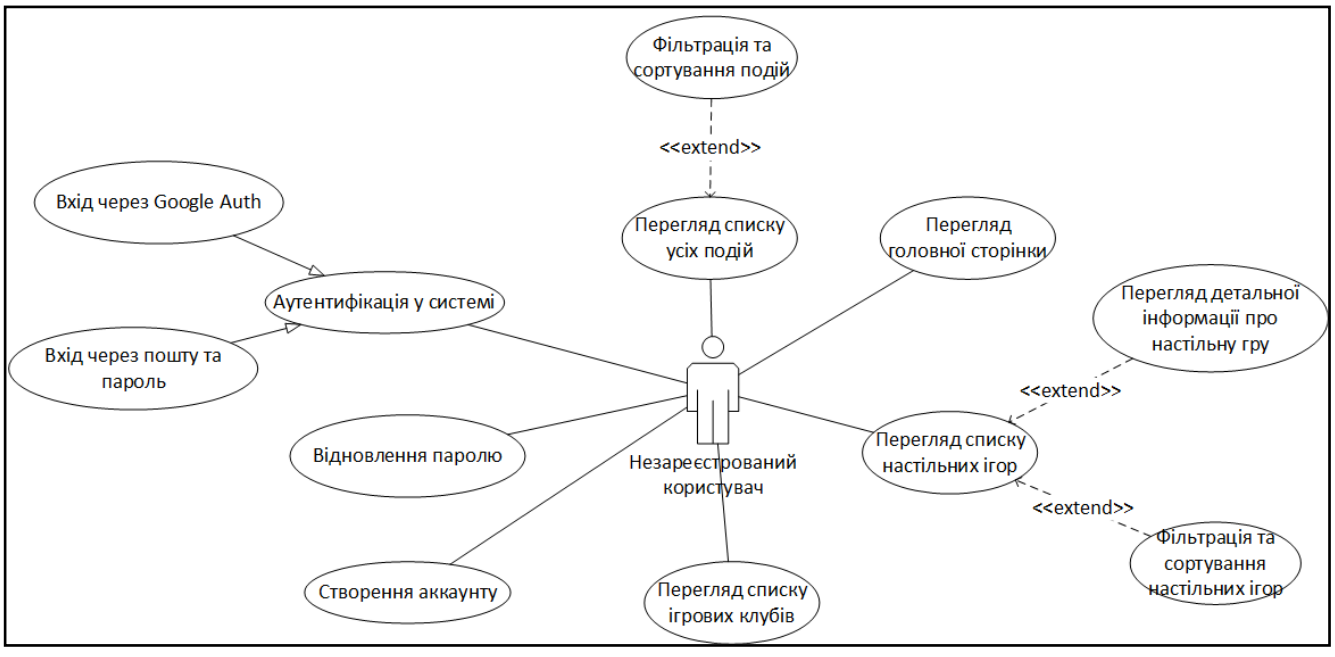


Рисунок 3.1 – Діаграма варіантів використання для незареєстрованого користувача (рисунок виконано самостійно)

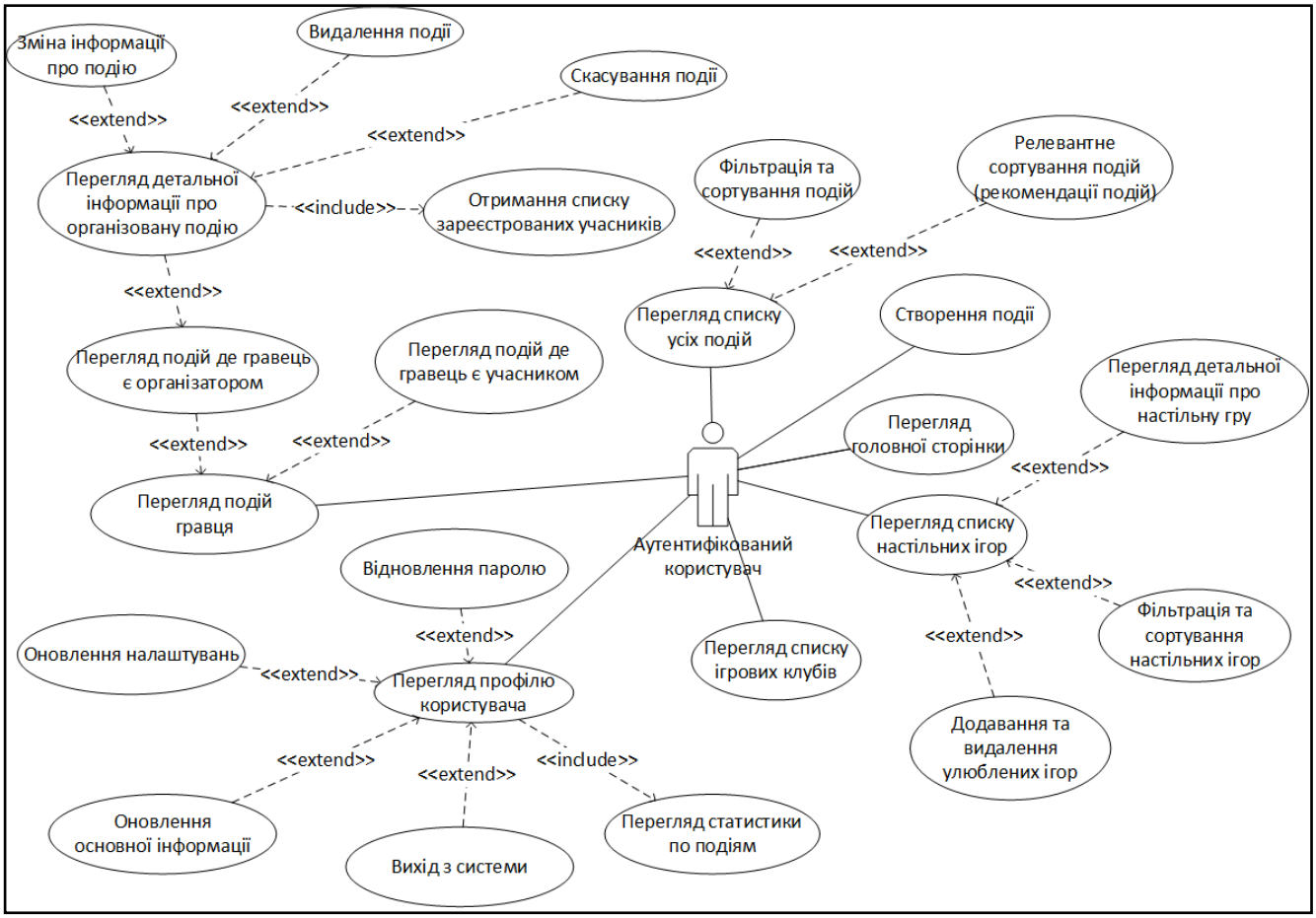


Рисунок 3.2 – Діаграма варіантів використання для аутентифікованого користувача (рисунок виконано самостійно)

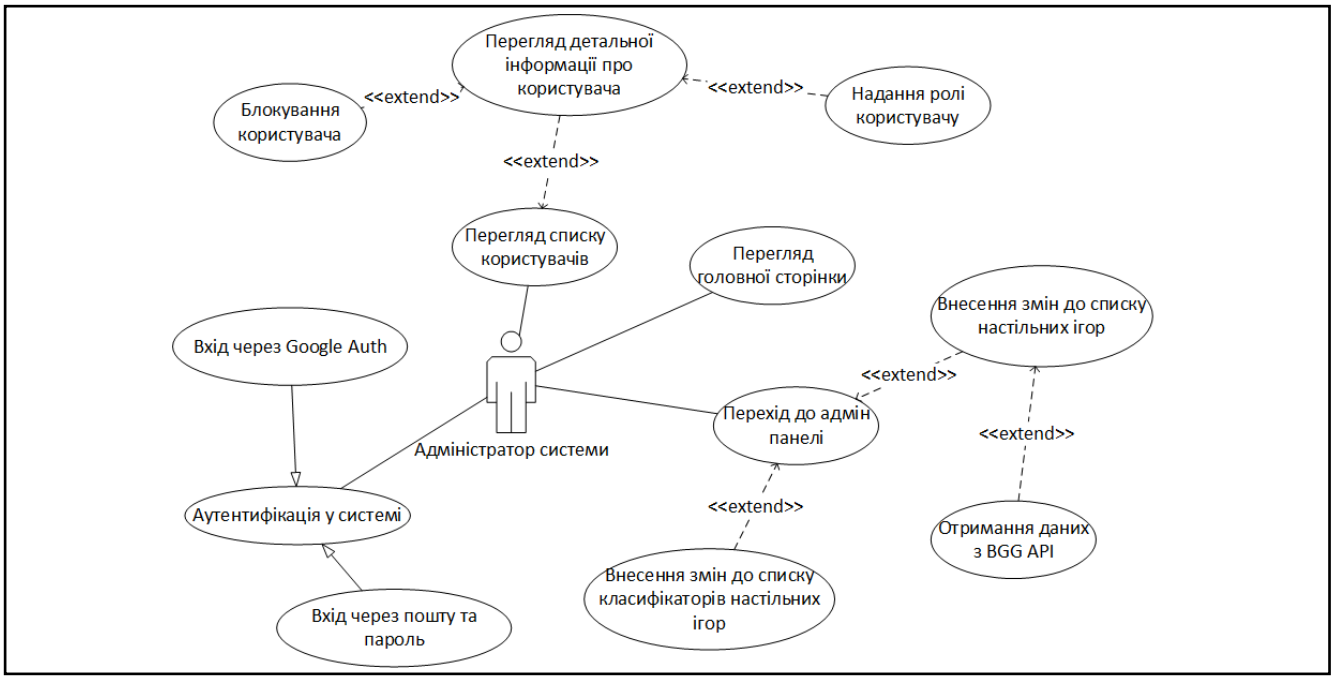


Рисунок 3.3 – Діаграма варіантів використання для адміністратора бізнес-логіки (рисунок виконано самостійно)

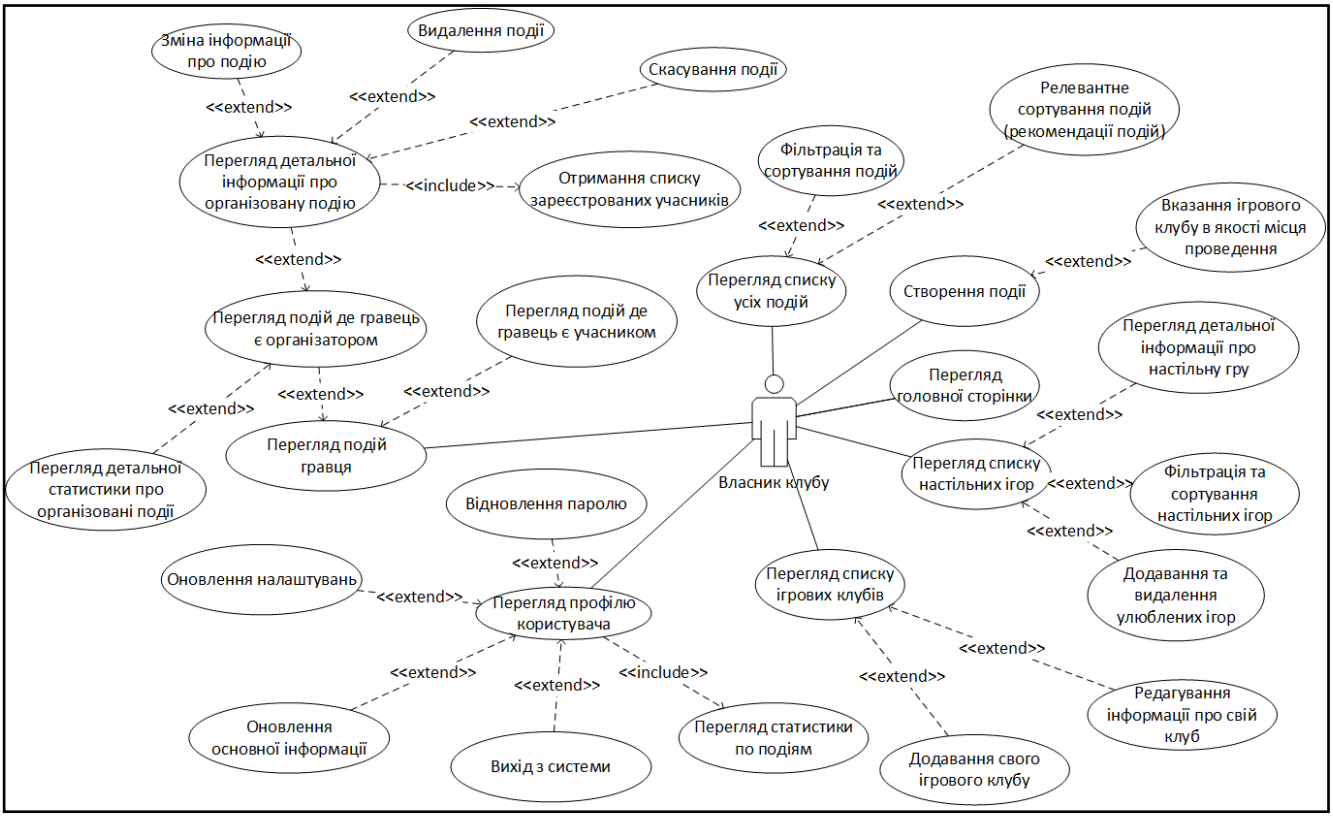


Рисунок 3.4 – Діаграма варіантів використання для власника ігрового клубу (рисунок виконано самостійно)

На діаграмі розгортання зображено архітектуру веб-додатку для платформи настільних ігор, яка функціонує в хмарному середовищі Azure. Основна логіка додатку реалізована як ASP.NET Web API (API.exe), розгорнута на веб-сервері, що знаходиться в Azure Cloud [5]. Користувачі взаємодіють із системою через браузер, у якому завантажується клієнтська частина — React-застосунок (index.js), що надсилає HTTP-запити до API за допомогою REST API.

У якості джерела даних для зберігання інформації використовується Azure SQL Server, до якого API-сервер звертається через захищене з'єднання (SQL over TLS). Додаток також інтегрується з кількома зовнішніми сервісами через REST API: BoardGameGeek (для отримання даних про ігри), Google Auth Service (для автентифікації користувачів через Google), GeoNames API Service (для геолокаційних функцій) та Nominatim API Service (для зворотного геокодування адрес). Google Auth Service потрібен як для серверної частини (для обробки

токенів), так і для клієнтської (отримання та надсилання токенів на сервер). Діаграма розгортання наведена на рисунку 3.5.

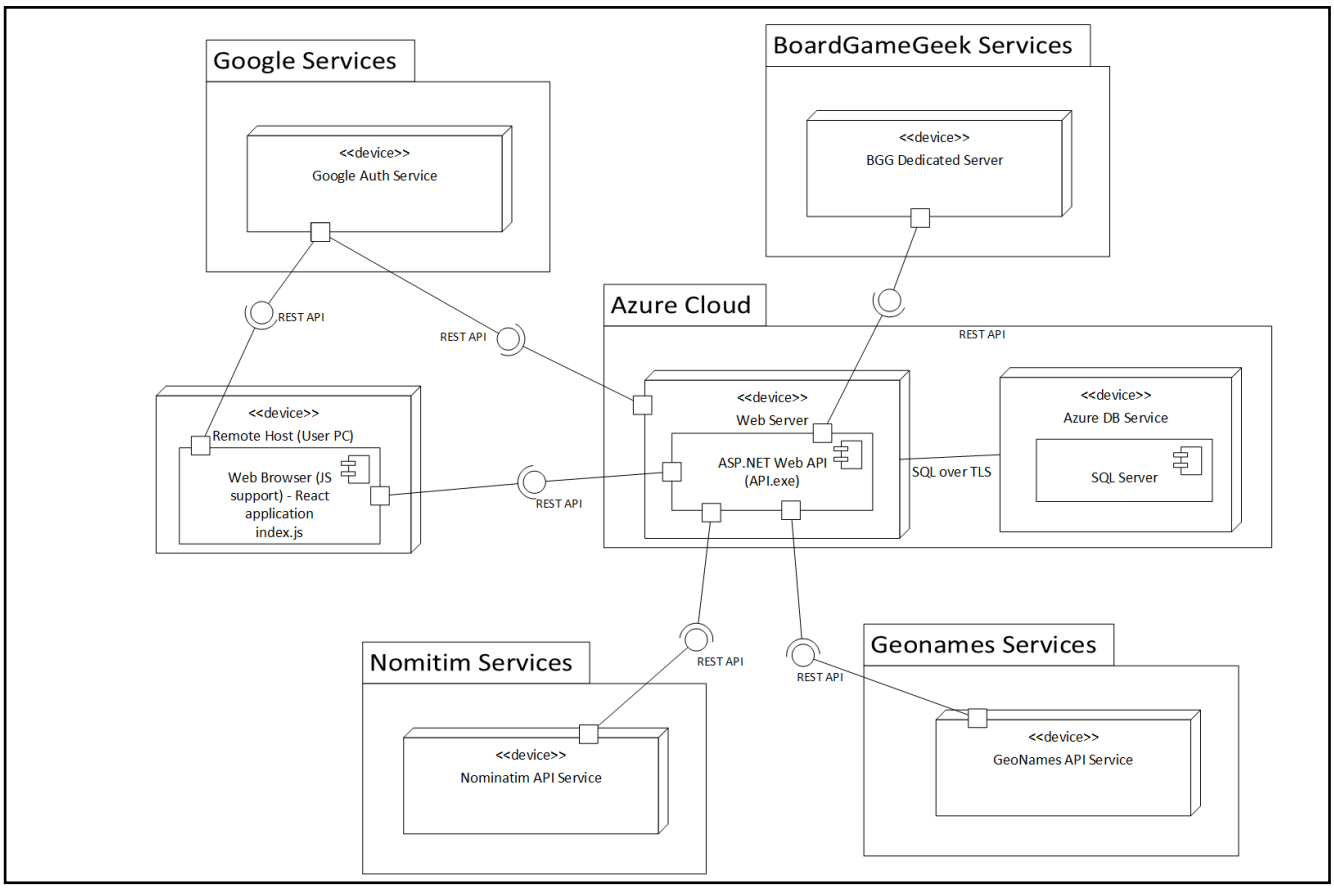


Рисунок 3.5 – Діаграма розгортання програмної системи TabletopConnect (рисунок виконано самостійно)

3.2 Проєктування архітектури програмної системи

Програмна система має типову багаторівневу архітектуру, де розділяється рівні представлення (клієнтська браузерна частина), рівень обробки даних (серверна частина) та рівень зберігання даних (база даних).

Клієнтський застосунок реалізований як веб-застосунок, який працює у браузері. Він відповідає за відображення інтерфейсу користувача, відображення даних, обробку введення та взаємодію з сервером через HTTP-запити. Користувачі можуть взаємодіяти з системою через зручний та інтуїтивний інтерфейс, що оновлюється без перезавантаження сторінки, завдяки підходу SPA (Single Page Application). Це дозволяє забезпечити швидку навігацію між сторінками, покращити досвід користування та знизити навантаження на сервер. Клієнтська

частина побудована з використанням технологій HTML, CSS, JavaScript/TypeScript з використанням сучасного фронтенд-фреймворку React, що забезпечує компонентну архітектуру, повторне використання коду та ефективну роботу з віртуальним DOM. Для зборки та розгортання застосунку використовується інструмент Vite, який забезпечує високу швидкість розробки завдяки швидкому старту, гарячому оновленню модулів (HMR) та оптимізації продуктивної збірки. Взаємодія з сервером здійснюється через REST API з обробкою асинхронних запитів, що дозволяє ефективно синхронізувати інтерфейс з отриманими даними. У подальших версіях планується реалізація адаптивного дизайну для підтримки мобільних пристроїв.

Серверна частина відповідає за обробку запитів від клієнта, реалізацію бізнес-логіки та доступу до бази даних та інтеграцію з зовнішніми сервісами. Вона представлена у вигляді веб-служби, реалізованої за допомогою технології ASP.NET Core на платформі .NET на мові C#. Завдяки своїй модульності ASP.NET Core забезпечує високу продуктивність, кросплатформеність та можливість масштабування проєкту. Доступ до даних реалізовано за допомогою технології Entity Framework, яка являє собою ORM-фреймворк для зручного доступу до даних і є абстракцією над низькорівневою технологією доступу до даних ADO.NET. Він дозволяє працювати з даними у вигляді об’єктів і використовувати запити LINQ замість SQL-інструкцій. Entity Framework також відповідає за автоматичне створення структури бази даних на основі C# моделей (підхід Code First) та підтримку міграцій для поступового оновлення схеми БД. Серверна частина надає інтерфейс у вигляді RESTful API з використанням усіх HTTP методів (GET, POST, PUT, DELETE), через який клієнтська частина взаємодіє з бізнес-логікою, отримує та надсилає дані для обробки.

База даних використовується для збереження інформації про усі сутності, які представлені у системі. В якості реляційної системи керування базою даних використовується SQL Server, оскільки вона добре інтегрується з технологіями .NET та підтримується у хмарному середовищі Azure. Усі основні сутності зберігаються у вигляді таблиць з відповідними зв’язками, що забезпечує цілісність

даних. Після розгортання системи у храмі передбачено зберіганню даних у хмарній версії SQL Server (Azure SQL Database), що забезпечує високу доступність, масштабованість, резервне копіювання та захищене з’єднання між застосунком і базою. Підключення до бази даних реалізовано через захищене з'єднання із застосуванням строк підключення, що зберігаються у конфігураційному файлі застосунку. Це дозволяє відокремити логіку застосунку від деталей зберігання та надає можливість легко змінювати середовище бази даних без модифікації бізнес-логіки.

Архітектура серверної частини побудована на основі підходу Onion Architecture з використанням принципів Domain-Driven Design (DDD) [6]. Основною ідеєю цієї архітектури є чітке розділення відповідальностей та залежностей між шарами, що дозволяє підвищити гнучкість, тестованість і підтримуваність системи. В архітектурі серверної частини представлено такі шари: Domain, Application, Infrastructure, Persistence, API та Common. Зовнішні шари мають залежності на внутрішні, але не навпаки. Найважливішим є доменний рівень, що знаходиться в центрі архітектури, і саме він визначає бізнес-логіку. Усі інші частини системи побудовані навколо нього.

Шар Domain є ядром системи і втілює основні концепції предметної області відповідно до підходу Domain-Driven Design (DDD). Його головною метою є моделювання бізнес-логіки максимально наближено до реального світу, в якому працює система. У цьому шарі зосереджена уся важлива логіка, що описує поведінку сутностей системи, а не просто їх структуру.

За принципами DDD доменна модель побудована навколо агрегатів (Aggregates) — цілісних груп об’єктів, що об’єднані бізнес-логікою і мають чітко визначений корінь (Aggregate Root). Саме через цей корінь здійснюється вся взаємодія з агрегатом ззовні. Це дозволяє централізовано управляти інваріантами, валідувати стан моделі та гарантувати консистентність даних всередині агрегату. У системі представлено такі агрегати як Board Games Aggregate, Events Aggregate, Game Club Aggregate та інші.

Крім агрегатів, у доменній моделі використовуються value objects (об’єкти-значення) — це об’єкти, які не мають ідентичності, а лише набір властивостей. Вони завжди порівнюються за значенням і часто є незмінними. Вони дозволяють інкапсулювати валідацію та логіку роботи з певними типами даних, що покращує читаність і повторне використання коду.

Шар Application відповідає за реалізацію бізнес-операцій на рівні всієї системи. Він координує взаємодію між доменною логікою (Domain), інфраструктурними компонентами (Infrastructure, Persistence) та зовнішніми запитами (через API). Основна задача цього шару — організація use cases (випадків використання), які описують, як саме має працювати система у відповідь на дії користувача або інші події.

У цьому шарі розміщені Application Services — класи, які інкапсулюють логіку сценаріїв використання. Наприклад, окремий сервіс може відповідати за додавання нової гри, оновлення профілю користувача або обробку запиту на рекомендацію. Ці сервіси зазвичай мають просту структуру: вони приймають DTO як вхідні параметри, виконують необхідну координацію між доменними об’єктами та викликають відповідні інфраструктурні сервіси чи репозиторії.

Шар також відповідає за обробку транзакцій, валідацію введених даних (на рівні сценарію), логування і виклик сервісу повідомлень.

Шар Infrastructure відповідає за реалізацію технічних аспектів роботи системи, які не пов’язані безпосередньо з бізнес-логікою, але є критично важливими для функціонування застосунку. Цей шар забезпечує інтеграцію з зовнішніми сервісами, реалізує інтерфейси, визначені у шарі Application, та включає реалізації крос-секторальних інфраструктурних сервісів.

У поточній системі шар Infrastructure включає наступні ключові компоненти:

- налаштування аутентифікація та авторизації за допомогою JWT та зовнішнього API Google OAuth2;
- сервіси надсилання email листів для підтвердження реєстрації, відновлення паролю та інші, що базується на SMTP;

- інтеграція з зовнішніми API, такими як BGG API (для отримання інформації про настільні ігри), GeoNames та Nominatim (для роботи з геоданими);
- провайдери системного часу та поточного користувача;
- налаштування безпеки та конфігурації;
- налаштування Swagger, що базується на специфікації OpenAPI для опису формальної структури API;
- реалізацію інтерфейсу доступу до сховища статичних файлів (таких як картинки профіля користувача);
- налаштування імпорту даних з CSV файлу;
- кешування основних сутностей, доступ до яких займає багато часу (список усіх настільних ігор з їх пов'язаними параметрами).

Шар Persistence відповідає за реалізацію доступу до бази даних та зберігає реалізації інтерфейсів, що були визначені у шарі Application. У цьому шарі реалізуються репозиторії, патерн Unit of Work (для підтримки транзакції), а також конфігурація доступу до бази даних з використанням Entity Framework Core. Його основне завдання — інкапсуляція логіки роботи з джерелом зберігання даних, щоб інші частини системи не мали справу з конкретною реалізацією БД. Також цей шар містить механізм міграцій, який дозволяє генерувати схему даних без ручного внесення змін до БД за допомогою SQL запитів. Схема даних змінюється по мірі еволюції шару Domain та забезпечує узгодженість схеми даних у всіх середовищах. Persistence-шар повністю ізольований від бізнес-логіки й взаємодіє лише через інтерфейси, що забезпечує гнучкість, можливість тестування і відповідність принципам Clean Architecture.

Шар API є верхнім рівнем серверної частини застосунку, який відповідає за прийом HTTP-запитів від клієнтської частини, виклик відповідної бізнес-логіки та формування HTTP-відповідей. Основними елементами цього шару є контролери (Controllers), які реалізують REST API. Кожен контролер обслуговує певну групу функціональності (наприклад, ігри, користувачі, автентифікація) і викликає відповідні сервіси з Application-шару. Для передачі даних між клієнтом і сервером

використовуються DTO (Data Transfer Objects), що забезпечують ізоляцію внутрішньої структури домену від зовнішнього API. Контролери повинні залишатися «тонкими» — без реалізації логіки, лише з викликом потрібних сервісів і поверненням результатів.

Крім контролерів, у шарі API налаштовуються middleware-компоненти, які обробляють запити до або після основного контролера. Серед них — автентифікація, авторизація, логування, CORS, обробка винятків тощо. Завдяки цьому архітектурному розподілу API-шар є лише зовнішнім інтерфейсом до системи та не залежить від внутрішніх реалізацій з Persistence або Infrastructure, що забезпечує гнучкість і масштабованість розробки.

Шар Common містить загальні елементи, які можуть використовуватися у різних частинах системи та не належать до конкретної предметної області. Сюди входять константи, які забезпечують централізоване зберігання незмінних значень (наприклад, ролі користувачів, назви політик авторизації, налаштування конфігурацій), перерахування (enums) для уніфікованого представлення статусів, типів або категорій, а також розширення (extension methods), які додають зручні методи до стандартних типів .NET або типів, що використовуються у системі. Цей шар не має залежностей від інших шарів архітектури і може використовуватись у будь-якому з них, що робить його зручним інструментом для підтримки повторного використання коду та уникнення дублювання.

Залежності між архітектурними шарами серверної частини наведено на UML діаграмі компонентів (рисунок 3.6)

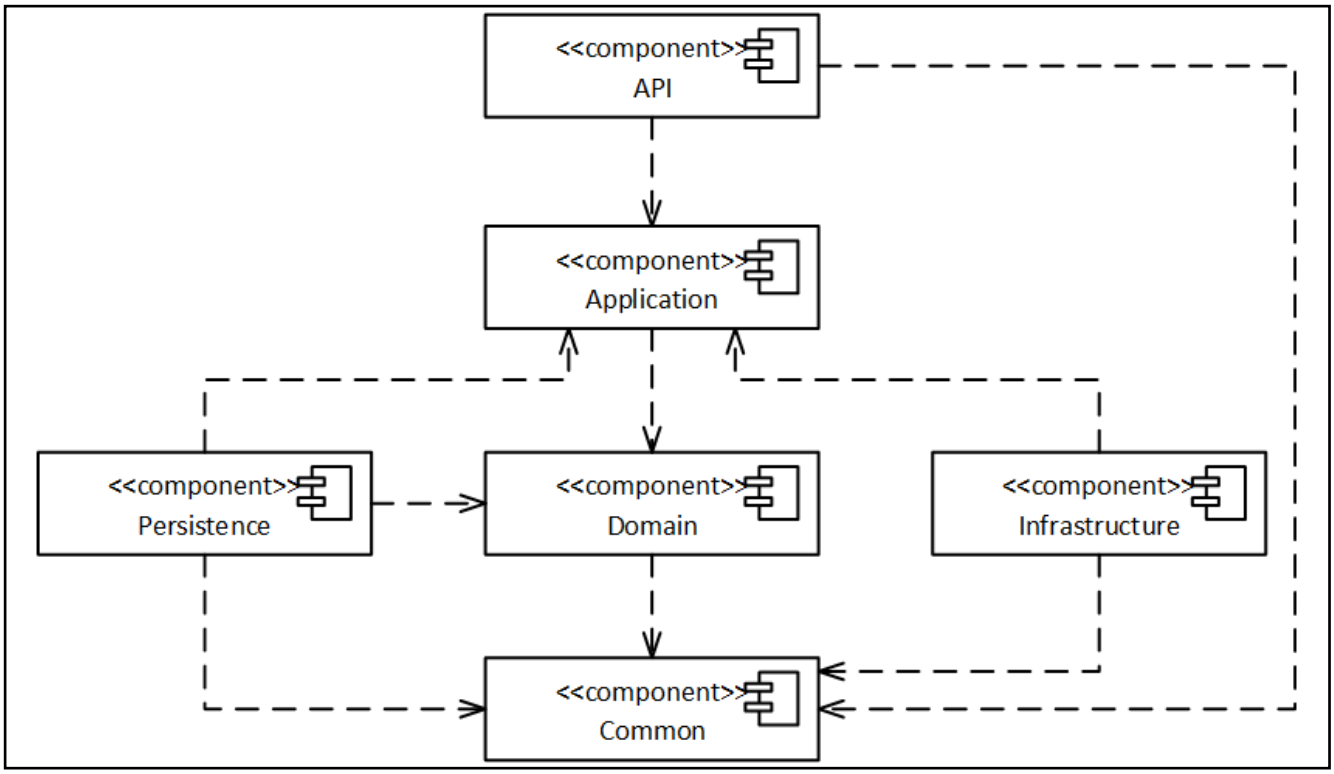


Рисунок 3.6 – UML діаграма компонентів шарів серверної частини (рисунок виконано самостійно)

3.3 Проєктування структури зберігання даних

Для зберігання даних обрано реляційну модель, оскільки вона найкраще підходить для структурованої інформації з чіткими зв’язками між сутностями, що характерно для даної системи. Реляційна модель забезпечує надійність, цілісність даних та потужні можливості для побудови запитів за допомогою мови SQL. Вона дозволяє ефективно реалізувати складні зв’язки типу "один до багатьох" та "багато до багатьох", які часто зустрічаються в інформаційних системах.

У рамках системи дані зберігаються у вигляді таблиць, що відповідають основним сутностям предметної області — користувачам, іграм, подіям, профілям тощо. Кожна таблиця має первинний ключ, який унікально ідентифікує кожен запис, та зовнішні ключі, що встановлюють зв’язки між таблицями. Для підвищення ефективності запитів та забезпечення узгодженості даних використовуються індекси, обмеження цілісності (наприклад, обов’язковість полів або унікальність значень) та каскадні правила для оновлення і видалення пов’язаних записів.

Структура реляційної бази даних наведена на ER-діаграмі, побудованій у нотації Crow's Foot (додаток А).

На діаграмі відображено наступні сутності та їх атрибути:

– BoardGames — центральна сутність, яка зберігає основну інформацію про настільні ігри (назва, опис, складність, рекомендована кількість гравців, вікові обмеження, дані з BoardGameGeek тощо);
– Users — користувачі платформи з даними для аутентифікації;
– PlayerProfile — профіль користувача з додатковими персональними даними (ім’я, нікнейм, біографія тощо);
– GameClub — сутність, що представляє клуб настільних ігор з контактними даними та геолокацією;
– Events — події, які організовуються клубами та користувачами, включають дані про місце, час, кількість гравців, ціну, тип події;
– EventParticipation — таблиця участі гравців у подіях;
– FavouriteGames — улюблені ігри гравців;
– Families — сімейства ігор;
– ConfirmationTokens — токени підтвердження для користувачів;
– Publisher, Designer, Mechanic, Category, Theme, Subcategory) — окремі довідники, що характеризують ігри, пов’язані з ними через таблиці-зв’язки типу many-to-many (BoardGamePublishers, BoardGameMechanics, BoardGameDesigners, BoardGameThemes, BoardGameCategories, BoardGameSubcategories);

На діаграмі представлені наступні зв’язки між сутностями:

– зв’язок «BoardGames – Publishers»: одна настільна гра може мати одного або більше видавців, а один видавець може бути пов’язаний із багатьма іграми. Це зв’язок «багато до багатьох», реалізований через проміжну таблицю BoardGamePublishers;
– зв’язок «BoardGames – Designers»: одна гра може мати кількох дизайнерів, а один дизайнер — бути автором багатьох ігор. Це також зв’язок «багато до багатьох», реалізований через таблицю BoardGameDesigners;

- зв’язок «BoardGames – Subcategories»: одна гра може належати до кількох підкатегорій, а одна підкатегорія може включати багато ігор. Це зв’язок «багато до багатьох», через BoardGameSubcategories;
- зв’язок «BoardGames – Mechanics»: одна гра може використовувати кілька механік, одна механіка може бути в багатьох іграх. Це зв’язок «багато до багатьох», реалізований через BoardGameMechanics;
- зв’язок «BoardGames – Themes»: одна гра може мати кілька тем, одна тема — бути у багатьох іграх. Це зв’язок «багато до багатьох», реалізований через BoardGameThemes;
- зв’язок «BoardGames – Categories»: одна гра може належати до кількох категорій, одна категорія — включати багато ігор. Це зв’язок «багато до багатьох», через BoardGameCategories;
- зв’язок «BoardGames – Families»: одна гра може належати до 0 або 1 сімейства, одне сімейство може включати багато ігор. Це зв’язок «один до багатьох»;
- зв’язок «Users – PlayerProfile»: кожен користувач має один профіль, а кожен профіль належить одному користувачу. Це зв’язок «один до одного»;
- зв’язок «PlayerProfile – FavouriteGames»: один гравець може мати багато улюблених ігор, але кожен запис у таблиці FavouriteGames належить до одного профілю. Це зв’язок «один до багатьох»;
- зв’язок «BoardGames – FavouriteGames»: одна гра може бути улюбленою для багатьох гравців. Це зв’язок «один до багатьох»;
- зв’язок «PlayerProfile – EventParticipation»: один гравець може брати участь у багатьох подіях, а одна участь належить лише одному гравцю. Це зв’язок «один до багатьох»;
- зв’язок «Events – EventParticipation»: одна подія може мати багато учасників, а участь стосується однієї події. Це також зв’язок «один до багатьох»;

- зв’язок «Events – PlayerProfile (OrganizerPlayerId)»: одна подія має одного організатора (гравця), гравець може організовувати багато подій. Це зв’язок «один до багатьох»;
- зв’язок «Events – BoardGames»: подія може бути пов’язана з однією грою, але одна гра може бути частиною багатьох подій. Це зв’язок «один до багатьох»;
- зв’язок «Events – GameClub»: одна подія може проводитися в одному клубі, а клуб може приймати багато подій. Це зв’язок «один до багатьох»;
- зв’язок «GameClub – Users (OwnerId)»: один клуб має одного власника (користувача), а користувач може володіти багатьма клубами. Це зв’язок «один до багатьох»;
- зв’язок «Users – ConfirmationTokens»: один користувач може мати кілька токенів підтвердження, токен належить одному користувачу. Це зв’язок «один до багатьох».

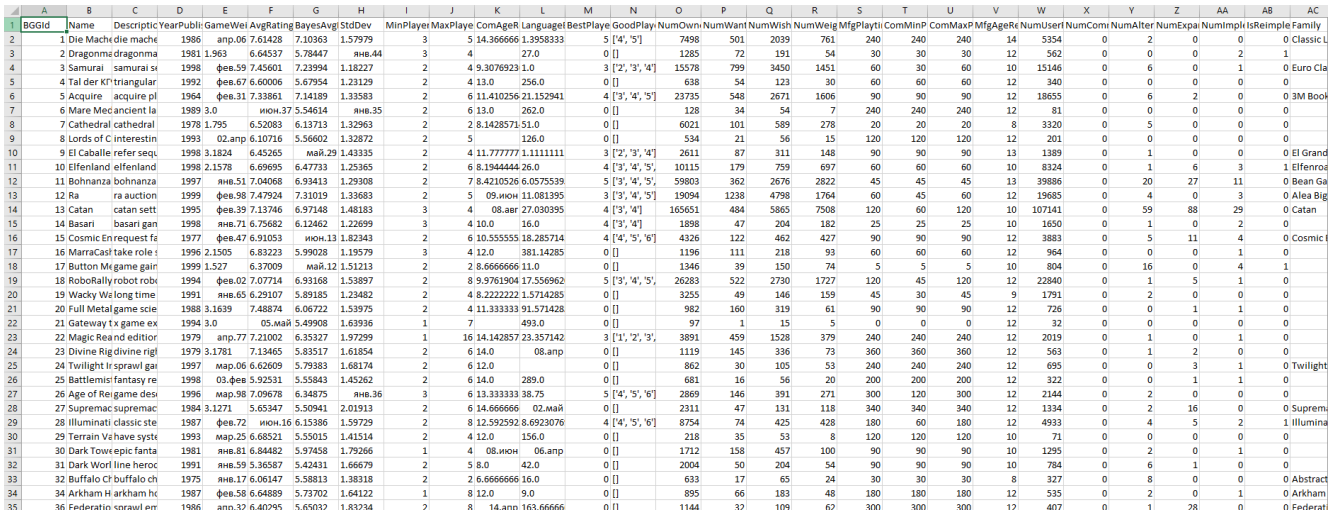
Дана схема задовольняє третю нормальну форму (3NF), так як вона задовольняє другу нормальну форму (2NF) (усі неключові атрибути залежать від повного первинного ключа), задовольняє першу нормальну форму (1NF) (усі атрибути атомарні, відсутні множинні або повторювані групи), і не має транзитивних залежностей (кожен неключовий атрибут залежить тільки від первинного ключа і не залежить від інших неключових атрибутів).

3.4 Приклади найцікавіших алгоритмів та методів

3.4.1 Імпорт та даних про настільні ігри та їх класифікатори

Для створення персоналізованих рекомендацій та організації ігрових подій система потребує детальної інформації про настільні ігри. Такі дані включають рейтинг, кількість гравців, складність, вікові обмеження, категорії тощо. Щоб отримати якісну та об’єктивну вибірку, використовується датасет із сервісу Kaggle [7], сформований на основі інформації з популярного ігрового порталу Board Game Geek (BGG). Цей набір даних містить інформацію про 20000 ігор з їх класифікаторами, такими як категорії, підкатегорії, теми, жанри тощо. Також

міститься інформація про час ігрової сесії, середні оцінки гравців, рекомендований вік та інше. Фрагмент набору даних наведено на рисунку 3.7.



BGGId	Name	Description	YearPublished	GameWeight	AvgRating	BayesAvgRating	StdDev	MinPlayers	MaxPlayers	ComAgeRec	LanguageDependence	BestPlayers	GoodPlayers	NumOwned	NumWant	NumWish	NumWeights	MfgPlaytime	ComMinPlaytime	ComMaxPlaytime	MfgAgeRec	NumUserRatings	NumComments	NumAlterns	NumExpansions	NumImplementations	IsReimplementation	Family
1	Die Mache	die mache	1986	апр.06	7.61428	7.10363	1.57979	3	5	14.366666	1.3958333	5	['4', '5']	7498	501	2039	761	240	240	240	14	5354	0	2	0	0	0	Classic L
2	Dragonma	dragonma	1981	1.963	6.64537	5.78447	янв.44	3	4		27.0	0	[]	1285	72	191	54	30	30	30	12	562	0	0	0	2	1	
3	Samurai	samurai s	1998	фев.59	7.45601	7.23994	1.18227	2	4	9.3076923	1.0	3	['2', '3', '4']	15578	799	3450	1451	60	30	60	10	15146	0	6	0	1	0	Euro Cla
4	Tal der Kö	triangular	1992	фев.67	6.60006	5.67954	1.23129	2	4	13.0	256.0	0	[]	638	54	123	30	60	60	60	12	340	0	0	0	0	0	
5	Acquire	acquire pl	1964	фев.31	7.33861	7.14189	1.33583	2	6	11.410256	21.152941	4	['3', '4', '5']	23735	548	2671	1606	90	90	90	12	18655	0	6	2	0	0	3M Book
6	Mare Med	ancient la	1989	3.0	июн.37	5.54614	янв.35	2	6	13.0	262.0	0	[]	128	34	54	7	240	240	240	12	81	0	0	0	0	0	
7	Cathedral	cathedral	1978	1.795	6.52083	6.13713	1.32963	2	2	8.1428571	51.0	0	[]	6021	101	589	278	20	20	20	8	3320	0	5	0	0	0	
8	Lords of C	interestin	1993	02.апр	6.10716	5.56602	1.32872	2	5		126.0	0	[]	534	21	56	15	120	120	120	12	201	0	0	0	0	0	
9	El Caballe	refer sequ	1998	3.1824	6.45265	май.29	1.43335	2	4	11.777777	1.1111111	3	['2', '3', '4']	2611	87	311	148	90	90	90	13	1389	0	1	0	0	0	El Grand
10	Elfenland	elfenland	1998	2.1578	6.69695	6.47733	1.25365	2	6	8.1944444	26.0	4	['3', '4', '5',	10115	179	759	697	60	60	60	10	8324	0	1	6	3	1	Elfenroa
11	Bohnanza	bohnanza	1997	янв.51	7.04068	6.93413	1.29308	2	7	8.4210526	6.0575539	5	['3', '4', '5',	59803	362	2676	2822	45	45	45	13	39886	0	20	27	11	0	Bean Ga
12	Ra	ra auction	1999	фев.98	7.47924	7.31019	1.33683	2	5	09.июн	11.081395	3	['3', '4', '5']	19094	1238	4798	1764	60	45	60	12	19685	0	4	0	3	0	Alea Big
13	Catan	catan sett	1995	фев.39	7.13746	6.97148	1.48183	3	4	08.авг	27.030395	4	['3', '4']	165651	484	5865	7508	120	60	120	10	107141	0	59	88	29	0	Catan
14	Basari	basari gan	1998	янв.71	6.75682	6.12462	1.22699	3	4	10.0	16.0	4	['3', '4']	1898	47	204	182	25	25	25	10	1650	0	1	0	2	0	
15	Cosmic En	request fa	1977	фев.47	6.91053	июн.13	1.82343	2	6	10.555555	18.285714	4	['4', '5', '6']	4326	122	462	427	90	90	90	12	3883	0	5	11	4	0	Cosmic I
16	MarraCast	take role s	1996	2.1505	6.83223	5.99028	1.19579	3	4	12.0	381.14285	0	[]	1196	111	218	93	60	60	60	12	964	0	0	0	1	0	
17	Button Me	game gain	1999	1.527	6.37009	май.12	1.51213	2	2	8.6666666	11.0	0	[]	1346	39	150	74	5	5	5	10	804	0	16	0	4	1	
18	RoboRally	robot robo	1994	фев.02	7.07714	6.93168	1.53897	2	8	9.9761904	17.556962	5	['3', '4', '5',	26283	522	2730	1727	120	45	120	12	22840	0	1	5	1	0	
19	Wacky Wo	long time	1991	янв.65	6.29107	5.89185	1.23482	2	4	8.2222222	1.5714285	0	[]	3255	49	146	159	45	30	45	9	1791	0	2	0	0	0	
20	Full Metal	game scie	1988	3.1639	7.48874	6.06722	1.53975	2	4	11.333333	91.571428	0	[]	982	160	319	61	90	90	90	12	726	0	0	1	1	0	
21	Gateway t	x game ex	1994	3.0	05.май	5.49908	1.63936	1	7		493.0	0	[]	97	1	15	5	0	0	0	12	32	0	0	0	0	0	
22	Magic Rea	nd editior	1979	апр.77	7.21002	6.35327	1.97299	1	16	14.142857	23.357142	3	['1', '2', '3',	3891	459	1528	379	240	240	240	12	2019	0	1	0	1	0	
23	Divine Rig	divine rigl	1979	3.1781	7.13465	5.83517	1.61854	2	6	14.0	08.апр	0	[]	1119	145	336	73	360	360	360	12	563	0	1	2	0	0	
24	Twilight Ir	sprawl gar	1997	мар.06	6.62609	5.79383	1.68174	2	6	12.0		0	[]	862	30	105	53	240	240	240	12	695	0	0	3	1	0	Twilight
25	Battlemis	fantasy re	1998	03.фев	5.92531	5.55843	1.45262	2	6	14.0	289.0	0	[]	681	16	56	20	200	200	200	12	322	0	0	1	1	0	
26	Age of Re	game des	1996	мар.98	7.09678	6.34875	янв.36	3	6	13.333333	38.75	5	['4', '5', '6']	2869	146	391	271	300	120	300	12	2144	0	2	0	0	0	
27	Supremac	supremac	1984	3.1271	5.65347	5.50941	2.01913	2	6	14.666666	02.май	0	[]	2311	47	131	118	340	340	340	12	1334	0	2	16	0	0	Suprema
28	Illuminati	classic ste	1987	фев.72	июн.16	6.15386	1.59729	2	8	12.592592	8.6923076	4	['4', '5', '6']	8754	74	425	428	180	60	180	12	4933	0	4	5	2	1	Illumina
29	Terrain Vs	have syste	1993	мар.25	6.68521	5.55015	1.41514	2	4	12.0	156.0	0	[]	218	35	53	8	120	120	120	10	71	0	0	0	0	0	
30	Dark Towe	epic fanta	1981	янв.81	6.84482	5.97458	1.79266	1	4	08.июн	06.апр	0	[]	1712	158	457	100	90	90	90	10	1295	0	2	0	1	0	
31	Dark Worl	line heroc	1991	янв.59	5.36587	5.42431	1.66679	2	5	8.0	42.0	0	[]	2004	50	204	54	90	90	90	10	784	0	6	1	0	0	
32	Buffalo Cl	buffalo ch	1975	янв.17	6.06147	5.58813	1.38318	2	2	6.6666666	16.0	0	[]	633	17	65	24	30	30	30	8	327	0	8	0	0	0	Abstract
34	Arkham H	arkham hc	1987	фев.58	6.64889	5.73702	1.64122	1	8	12.0	9.0	0	[]	895	66	183	48	180	180	180	12	535	0	2	0	1	0	Arkham
36	Federatio	sprawl ent	1986	апр.32	6.40295	5.65032	1.83234	2	8	14.апр	163.66666	0	[]	1144	32	109	62	300	300	300	12	407	0	1	28	0	0	Federati

Рисунок 3.7 – Фрагмент набору даних про настільні ігри
(рисунок виконано самостійно)

Основний файл для імпорту – це CSV-файл із розширеним набором характеристик настільних ігор. Для його зчитування використовується бібліотека CsvHelper, яка дозволяє зручно парсити структуровані табличні дані у C#-об’єкти на основі відповідного DTO-класу.

Перед використанням даних у системі проводиться низка перевірок та очищення, що є частиною етапу підготовки даних. Основні дії:

– значення GameWeight, якщо менше або дорівнює 1, замінюється на 1 (так як допустимий діапазон значень від 1 до 5, інші дані вважаємо помилковими);
– LanguageEase, що характеризує залежність гри від мови, приводиться до перерахування LanguageDependence, але тільки в межах допустимих значень (1–5);
– значення вікових рекомендацій (ComAgeRec) округлюється до цілого, якщо не є null;
– якщо для гри задана сімейна категорія (Family), вона звіряється з уже наявними у системі категоріями — якщо категорія знайдена, вона

асоціюється з грою; так само виконуємо співставленні гри з категоріями та іншими параметрами, які пов’язані з грою зв’язком багато-до-багатьох;

– після усіх трансформацій список ігор сортується за BggId, що полегшує наступну обробку та відображення.

Код сервісу для імпорту даних наведено у додатку Б.1.

3.4.2 Рекомендації настільних ігор

Однією з функцій система є рекомендації настільних ігор за вподобаннями користувача. Система рекомендацій у реалізованому варіанті працює за принципом контентно-орієнтованої фільтрації (content-based filtering) [8]. Вона не аналізує поведінку інших користувачів, а натомість формує рекомендації, порівнюючи властивості (ознаки) ігор, які подобаються користувачу, з усіма іншими іграми.

Кожна настільна гра представляється у вигляді числового вектора ознак (features), що містить One-hot encoding категорій, механік, тем і підкатегорій та нормалізовані числові параметри, такі як рік випуску, складність, рейтинг, мінімальна та максимальна кількість гравців. Додатково враховуються оцінки гри, її позиції у рейтингу, а також рік випуску (дані з зовнішнього сервісу BGG) [9]. Далі усі обрані користувачем ігри перетворюються у вектори, і обчислюється середній вектор вподобань. Всі інші ігри також перетворюються у вектори, після чого обчислюється косинусна подібність (cosine similarity) між цим середнім вектором і вектором кожної гри. Чим ближче значення до 1 — тим більше схожі вектори, тобто ігри.

Косинусна подібність визначає кут між двома векторами (формула 3.1)

$$similarity = \frac{V_1 \cdot V_2}{|V_1| \cdot |V_1|} \tag{3.1}$$

Після роботи алгоритму відбувається аналіз отриманих рекомендованих ігор і на основі їх характеристик встановлюються схожі класифікатори (механіки, жанри, піджанри, тематики), які повертаються у результаті роботи алгоритму як пояснення до рекомендацій.

Для математичних розрахунків використана бібліотека MathNet.Numerics.LinearAlgebra.Double (для векторних операцій: DenseVector, DotProduct, L2Norm тощо).

Код методів для рекомендації настільних ігор наведено у додатку Б.2.

Опис роботи алгоритму рекомендацій наведено на діаграмі активності (рисунок 3.8).

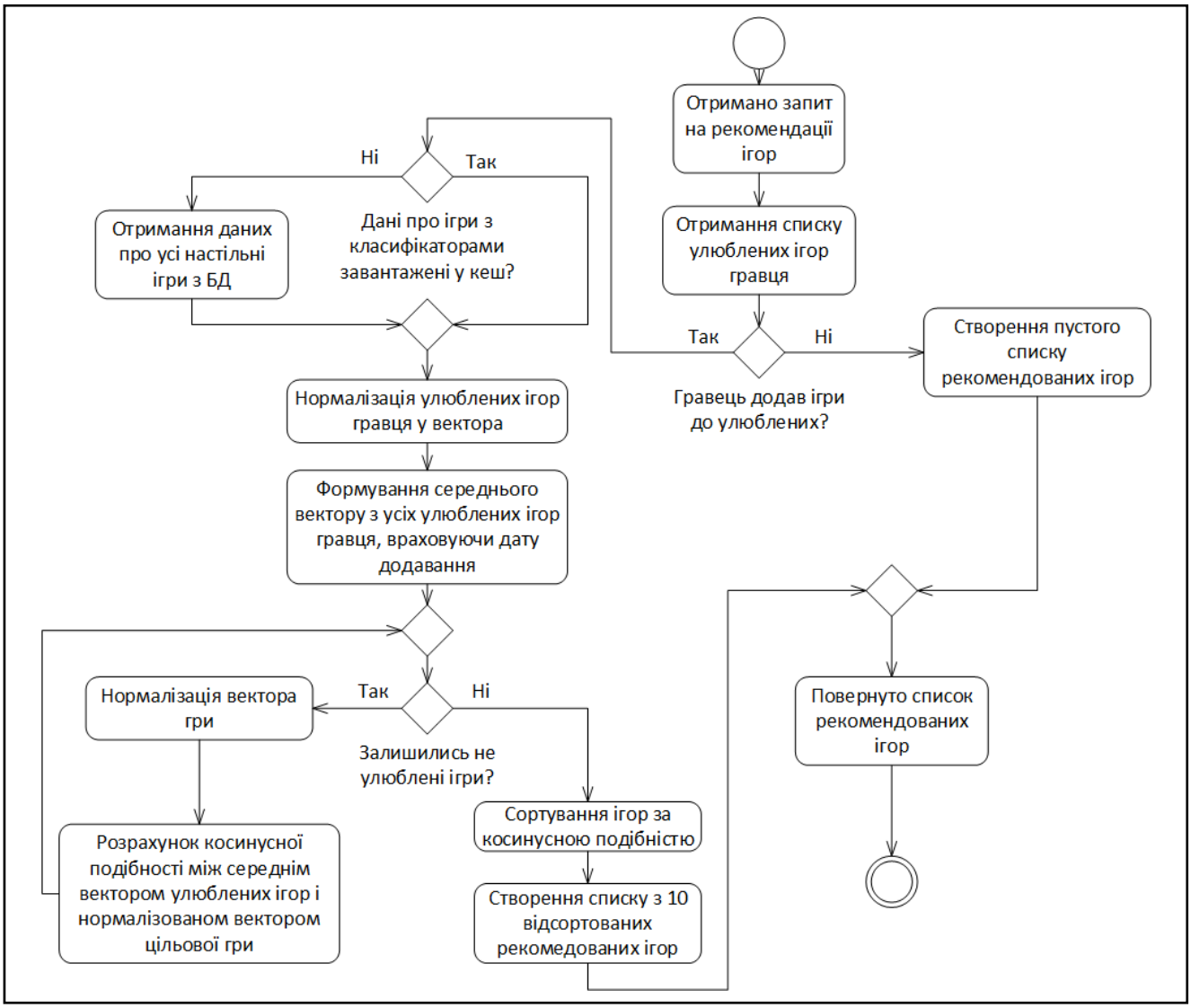


Рисунок 3.8 – Діаграма активності з візуалізацією алгоритму рекомендації настільних ігор (рисунок виконано самостійно)

3.4.3 Релевантне сортування подій

Система рекомендацій подій реалізована через принцип рейтингування (scoring) кожної події з урахуванням кількох важливих факторів, які визначають її релевантність для конкретного гравця. Далі події сортуються за рейтингом у порядку спадання, і повертаються найрелевантніші. Місцезнаходження гравця: отримується з клієнтської частини, зокрема довготу (Longitude) та широту (Latitude). Місцезнаходження події: зберігається в базі даних; при запиті до бази даних обчислюється відстань від гравця до кожної події (якщо воно проводиться не онлайн). Інша інформація про подію також береться з бази: чи подія онлайн, чи прив'язана до клубу, яка настільна гра представлена, скільки гравців вже зареєстровано, тощо.

Для кожної події обчислюється рейтинг (score) — це зважена сума факторів, елементи якої наведено у таблиці 3.1

Таблиця 3.1 – Коефіцієнти математичної моделі оцінки релевантності подій (таблиця виконана самостійно)

Фактор	Позначення	Вага	Як враховується	Призначення
Онлайн подія	W_{online}	0.7	Додається, якщо подія онлайн	Онлайн-події гнучкіші у доступі.
Відстань до події	$W_{distance}$	1.7	1 / (1 + distance)	Чим ближча подія, тим вищий бал.
Улюблена настільна гра	W_{fav}	2.0	Якщо гра є в списку улюблених	Перевага для улюблених ігор.
Без гри (відкрита подія)	W_{nogame}	0.3	Якщо гра не вказана	Низький бал, бо невідомо, чи буде цікаво.
Прив’язка до ігрового клубу	W_{club}	1.5	Додається, якщо є клуб	Клубні події вважаються якіснішими.
Кількість зареєстрованих гравців	W_{reg}	1.0	ln(1 + N)	Більше людей — потенційно цікавіше.

В результаті фінальну оцінку релевантності подій отримуємо за формулою 3.2.

$$score = W_{online} + W_{distance} + W_{fav} + W_{nogame} + W_{club} + W_{reg} \quad (3.2)$$

Коефіцієнти для розрахунку отримані емпіричним шляхом. Події з вищою оцінкою відображаються вище у порядку сортування.

Код сервісу для релевантного сортування подій наведено у додатку Б.3.

3.5 Створення UI / UX дизайну системи

Під час розробки UI/UX дизайну системи для настільних ігор у Figma було дотримано ключових принципів зручності, простоти та візуальної цілісності. Основний акцент зроблено на мінімалістичному інтерфейсі з логічною структурою — інтерфейс легко сприймається, не перевантажений зайвими деталями, а всі інтерактивні елементи мають зрозуміле розміщення. Дизайн витриманий в єдиній кольоровій гамі, що забезпечує послідовність і впізнаваність у всіх частинах системи. Реалізовано чітку візуальну ієрархію: кнопки для входу, фільтри пошуку та основна навігація виділяються кольором і позиціонуванням, що допомагає користувачу швидко орієнтуватись у функціоналі. Приклади дизайну сторінок наведені на рисунках 3.9 – 3.11.

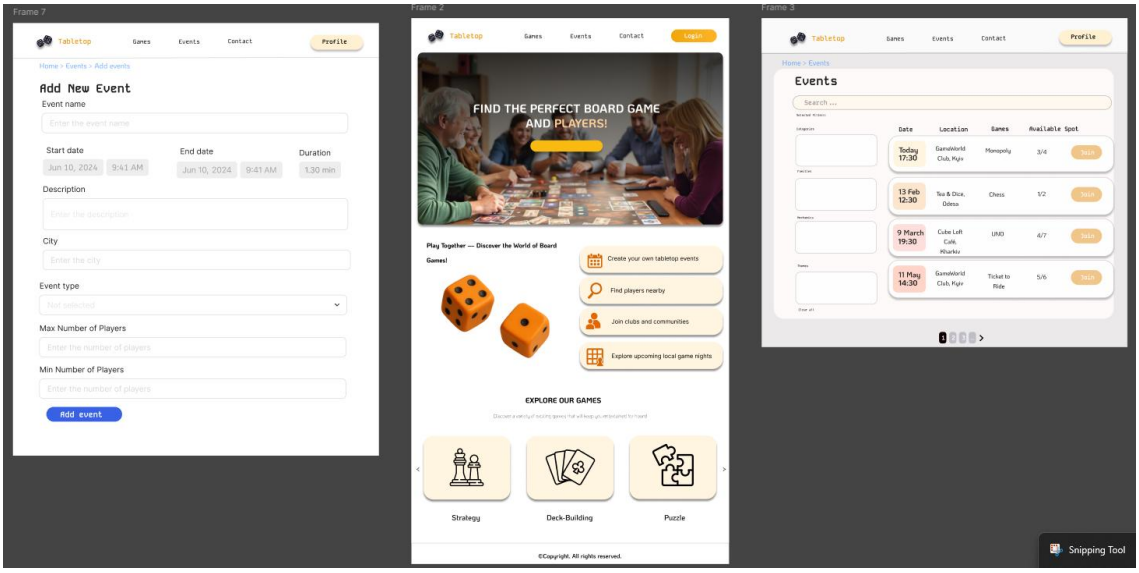


Рисунок 3.9 – Дизайни головної сторінки та сторінок з подіями (рисунок виконано самостійно)

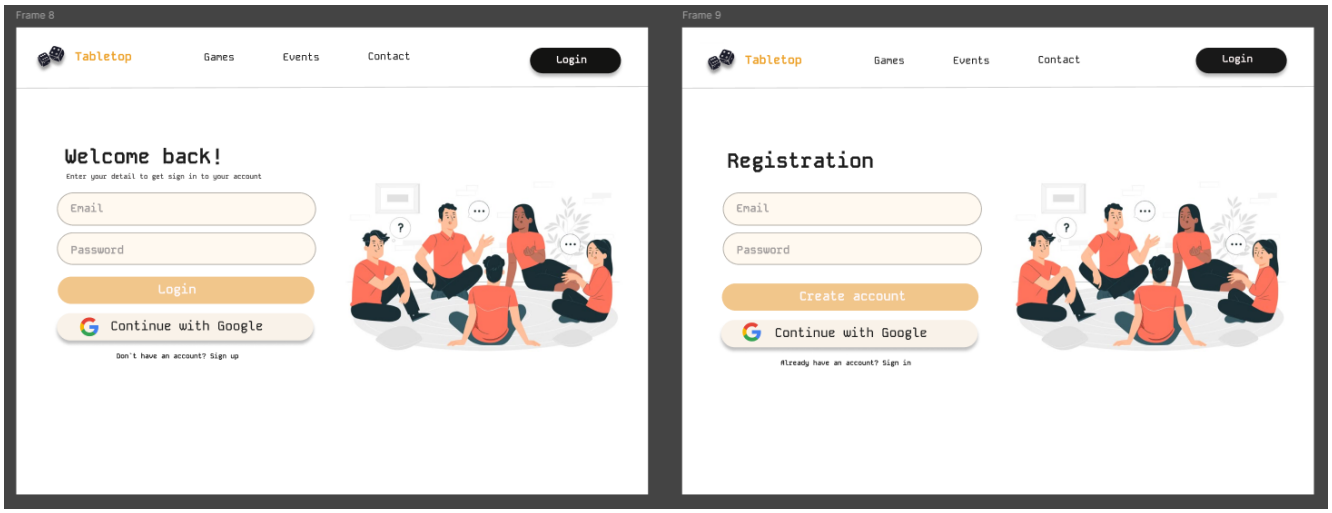


Рисунок 3.10 – Дизайни сторінок з реєстрацією та авторизацією (рисунок виконано самостійно)

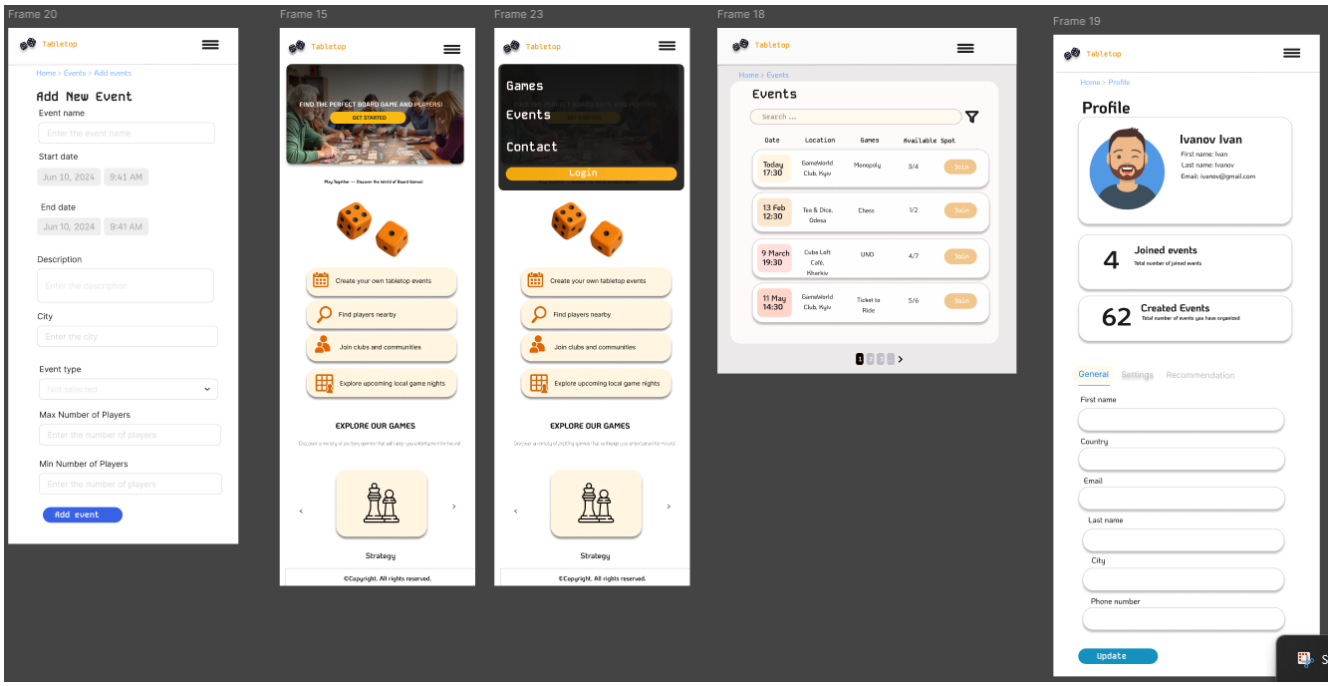


Рисунок 3.11 – Дизайни сторінок з адаптацією під мобільні пристрої (рисунок виконано самостійно)

Додатково створено дизайн електронних листів, які будуть відправлятися користувачу для підтвердження пошти, зміни паролю, оновлення ігрової події та інших системних повідомлень. Приклади дизайну електронних листів наведено на рисунку 3.12.

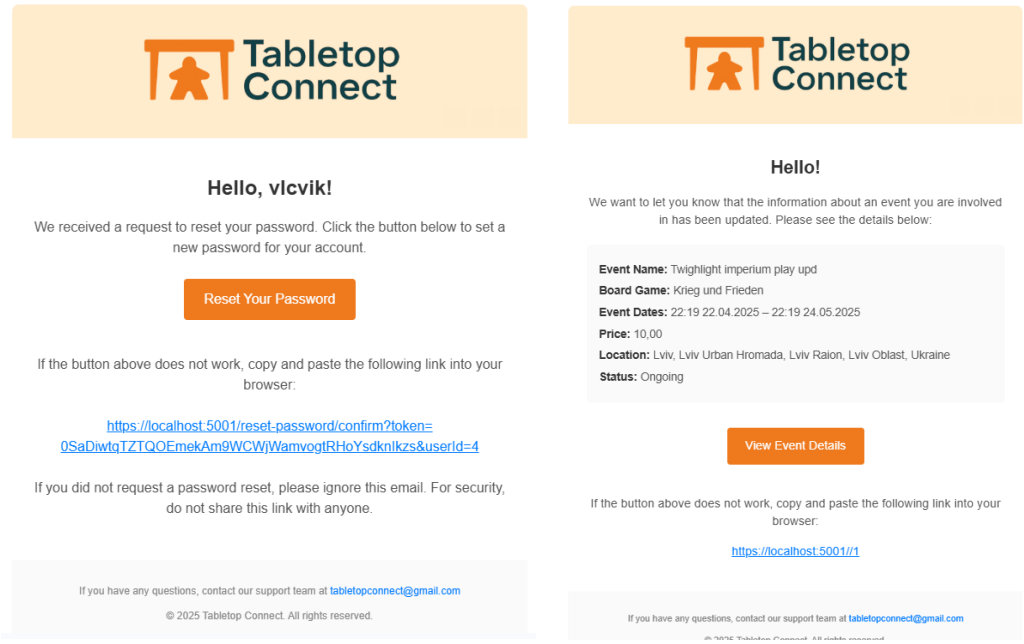


Рисунок 3.12 – Дизайн електронних листів (рисунок виконано самостійно)

В результаті було створено дизайн системи, який вирізняється простотою, зручністю та візуальною узгодженістю. Завдяки мінімалістичному підходу, логічній структурі та єдиній кольоровій гамі, користувач легко орієнтується в системі, а ключові елементи — такі як навігація, фільтри та кнопки — чітко виділені та інтуїтивно зрозумілі. Дизайн забезпечує комфортну взаємодію з користувачем і підтримує візуальну впізнаваність на всіх сторінках.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Використання Domain Driven Design при проєктуванні предметної області

Під час проектування та реалізації програмної системи було прийнято рішення використовувати підхід Domain Driven Design (DDD), який дозволяє відокремити частину бізнес-логіки від інфраструктурного коду, чітко структурувати предметну область та спростити майбутню підтримку і розвиток системи [6].

У предметній області було виокремлено кілька агрегатів, що представляють логічно цілісні сутності з власними інваріантами:

- BoardGameAggregate — представляє настільну гру разом із класифікаторами, що її описують: категорії, механіки, дизайнери, видавці тощо;
- EventAggregate — відповідає за організацію ігрових подій, зберігає інформацію про час, місце, організатора, запрошених гравців та пов’язані ігри;
- GameClubAggregate — моделює ігровий клуб з його назвою, місцем знаходження, членами клубу та організованими подіями;
- PlayerProfileAggregate — агрегує інформацію про користувача: його улюблені ігри, участь у подіях, історію оцінок та персональні уподобання.

Зазвичай, кожна основна сутність агрегату містить посилання на суміжні ValueObjects, в які винесено окремі властивості та бізнес-логіку, що притаманні тільки її окремій частині. Наприклад, для сутності BoardGame виділено окремо BggData (дані з сервісу BoardGameGeek), Players (дані про кількість гравців), PlayTime (дані про час ігрової сесії), RecommendedAge (дані про вік гравців). Таким чином можна розвантажити основну центральну сутність та розділити обов’язки, дотримавшись принципу Single Responsibility.

Окремим Bounded Context виділено підсистему IdentityAccessManagement, яка відповідає за авторизацію та аутентифікацію користувачів. У її межах реалізовані сутності User та Role, що відповідають за зберігання облікових даних, ролей та доступу до функцій системи відповідно до їх прав.

Для уніфікації географічних даних у системі було створено Value Object Location, який використовується в агрегатах GameClub та Event для позначення місця проведення події чи адреси клубу. Об’єкт значення є незмінним (immutable) та порівнюється за значенням, а не за ідентифікатором.

У системі також присутні класифікатори настільних ігор: Category, Mechanic, Designer, Publisher, Theme тощо. Вони представляють довідкову інформацію й пов’язані з агрегатом BoardGame.

Нижче приведено фрагмент код сутності PlayerProfile, яка являє собою представлення ігрового профілю гравця та інкапсулює його основну бізнес-логіку.

```
public class PlayerProfile : Entity<int>
{
    /* Конструктори та автоматичні властивості не відображено */
    public const string DefaultName = "User";

    public string FullName => $"{FirstName} {LastName}".Trim();

    public static PlayerProfile CreateRegular(int userId, string
nickname)
        => new(userId, null, null, nickname, null, null, null);

    public static PlayerProfile CreateExtended(
        int userId, string firstName, string lastName,string
nickname, string? avatarUrl, AvatarStorageType? storageType)
        => new(userId, firstName, lastName, nickname, null,
avatarUrl, storageType);

    public void UpdateIfNotNull(
        string? firstName = null,
        string? lastName = null,
        string? nickname = null,
        string? bio = null,
        string? avatarUrl = null,
        AvatarStorageType? avatarStorageType = null)
    {
        FirstName = FirstName ?? firstName;
        LastName = LastName ?? lastName;
        Nickname = Nickname ?? nickname;
        Bio = Bio ?? bio;

        if (AvatarUrl == null)
        {
            AvatarUrl = avatarUrl;
            AvatarStorageType = avatarStorageType;
        }
    }
    public void UpdateAvatar(
        string avatarUrl,
        AvatarStorageType avatarStorageType)
    {
```

```
        AvatarUrl = avatarUrl;
        AvatarStorageType = avatarStorageType;
    }
    public void RemoveAvatar()
    {
        AvatarUrl = null;
        AvatarStorageType = null;
    }
    public void UpdateBaseInfo(
        string? firstName,
        string? lastName,
        string? nickname,
        string? bio)
    {
        FirstName = firstName;
        LastName = lastName;
        Nickname = nickname;
        Bio = bio;
    }
}
```

4.2 Реалізація доступу до даних

У розробленій програмній системі для доступу до бази даних використано Entity Framework Core (EF Core) — сучасний ORM-фреймворк, що забезпечує зручний спосіб роботи з реляційними даними за допомогою об'єктно-орієнтованого підходу [10].

Для забезпечення чіткого розділення логіки конфігурації сутностей у базі даних усі налаштування винесено в окремі класи конфігурації, які реалізують узагальнені класи EntityTypeConfiguration<T> (в нашому випадку створено окремий абстрактний клас IdentifiableEntityConfiguration, в який винесено конфігурацію первинного ключа).

Наприклад, для сутності BoardGame конфігурування має наступний вигляд:

```
internal class BoardGameEntityConfiguration :
IdentifiableEntityConfiguration<BoardGame, int>
{
    public override void Configure(EntityTypeBuilder<BoardGame>
builder)
    {
        builder.ToTable("BoardGames");

        base.Configure(builder);

        builder.Property(e => e.GameComplexity)
            .HasPrecision(18, 2);
```

```
        builder.OwnsOne(e => e.BggData, bggData =>
        {
            bggData.Property(d => d.AverageRating)
                .HasPrecision(18, 2);

            bggData.Property(d => d.BggScore)
                .HasPrecision(18, 2);
        });

        builder.OwnsOne(e => e.PlayTime);
        builder.OwnsOne(e => e.Players, players =>
        {
            var converter = new ValueConverter<List<int>?, string>(
                v => JsonSerializer.Serialize(v,
(JsonSerializerOptions?)null),
                v => JsonSerializer.Deserialize<List<int>?>(v,
(JsonSerializerOptions?)null));

            var comparer = new ValueComparer<List<int>?>(
               (c1, c2) =>
                   (c1 == null && c2 == null) ||
                   (c1 != null && c2 != null &&
c1.SequenceEqual(c2)),
               c => c == null ? 0 : c.Aggregate(0, (a, v) =>
HashCode.Combine(a, v.GetHashCode())),
               c => c == null ? null : new List<int>(c));


           players.Property<List<int>?>("_goodPlayers")
                .HasConversion(converter)
                .Metadata.SetValueComparer(comparer);
        });
/* Налаштування інших зв'язків */
}
```

Такий підхід дозволив звести до мінімуму конфігурування контексту бази даних, через який ORM взаємодіє з базою даних. Код конфігурування контексту бази даних:

Для управління структурою бази даних використовувалися міграції EF Core, які автоматично створюють SQL-скрипти на основі змін у доменній моделі. Команди Add-Migration, Update-Database та Remove-Migration дозволяли версіонувати схему бази даних та підтримувати її актуальність протягом усього життєвого циклу проєкту.

Для інкапсуляції доступу до даних було реалізовано патерн Repository, що дозволяє відокремити логіку взаємодії з БД від бізнес-логіки. Базовий клас

репозиторію реалізує інтерфейс IRepository<TEntity, TKey> і надає універсальні методи роботи з сутностями. Сигнатура класу виглядає наступним чином:

```
    internal abstract class Repository<TEntity, TKey> :
IRepository<TEntity, TKey>
        where TEntity : Entity<TKey>
        where TKey : struct
```

Також реалізовано варіант SoftRepository для підтримки логічного видалення (Soft Delete). У цьому випадку сутності мають прапорець IsDeleted, який не видаляє дані фізично з БД, але виключає їх із запитів за замовчуванням.

Застосовано також патерн Unit of Work, який інкапсулює збереження змін у БД та дозволяє координувати роботу кількох репозиторіїв в одному запиті. Клас реалізує інтерфейс IUnitOfWork і надає метод для збереження даних і підтримки транзакційності:

```
    internal class UnitOfWork : IUnitOfWork
    {
        private readonly TabletopDbContext _dbContext;

        public UnitOfWork(TabletopDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        public async Task<bool> SaveChangesAsync(CancellationToken
cancellationToken)
        {
            try
            {
                return await
_dbContext.SaveChangesAsync(cancellationToken) > 0;
            }
            catch (DbUpdateException)
            {
                return false;
            }
        }
    }
```

4.3 Реалізація авторизації та аутентифікації

Для забезпечення безпеки у програмній системі реалізовано дві основні форми аутентифікації: традиційна аутентифікація за email та паролем, а також аутентифікація через Google-акаунт. Незалежно від методу входу, серверна частина формує JWT-токен, який використовується для подальшої авторизації користувача. Для зберігання інформації про користувачів використовується сутність User. Вона містить основні поля, зокрема email, хеш паролю (у разі класичної реєстрації), Google Id (у разі авторизації через Google), а також інші атрибути, як-от мова, номер телефону, роль тощо.

Класична форма аутентифікації реалізована через збереження хешованого паролю. Для хешування використовується алгоритм PBKDF2 з SHA-256. Впроваджено власну реалізацію IPasswordHasher, яка генерує соль та забезпечує безпечне зберігання паролю:

```
    internal class PasswordHasher : IPasswordHasher
    {
        private const int SaltSize = 16;
        private const int KeySize = 32;
        private const int Iterations = 100000;
        private static readonly HashAlgorithmName HashAlgorithm =
HashAlgorithmName.SHA256;
        public string HashPassword(string password)
        {
            byte[] salt = new byte[SaltSize];
            using (var rng = RandomNumberGenerator.Create())
            {
                rng.GetBytes(salt);
            }
            byte[] hash = Rfc2898DeriveBytes.Pbkdf2(
                Encoding.UTF8.GetBytes(password),
                salt,
                Iterations,
                HashAlgorithm,
                KeySize
            );
            return
$"{Convert.ToBase64String(salt)}.{Convert.ToBase64String(hash)}";
        }

        public bool VerifyPassword(string password, string storedHash)
        {
            string[] parts = storedHash.Split('.');
            if (parts.Length != 2)
                return false;
```

```
            byte[] salt = Convert.FromBase64String(parts[0]);
            byte[] storedHashBytes = Convert.FromBase64String(parts[1]);
            byte[] computedHash = Rfc2898DeriveBytes.Pbkdf2(
                Encoding.UTF8.GetBytes(password),
                salt,
                Iterations,
                HashAlgorithm,
                KeySize
            );
            return
CryptographicOperations.FixedTimeEquals(storedHashBytes, computedHash);
        }
    }
```

Перевірка паролю виконується шляхом повторного обчислення хешу з тією ж сіллю та порівняння результатів збереженим значенням.

Аутентифікація через Google реалізована за допомогою бібліотеки Google.Apis.Auth. Клас GoogleAuthService відповідає за валідацію токену, виданого клієнтом.

Після валідації зчитується інформація про користувача з payload (ID, email, підтвердження пошти, тощо) та створюється або оновлюється запис користувача у системі.

Підтвердження email та функція скидання паролю реалізовані через токени, які надсилаються на пошту. Після переходу за посиланням або введення токену користувач може підтвердити свою пошту або встановити новий пароль.

Після успішної аутентифікації система генерує JWT-токен, який клієнт використовує для авторизації у подальших запитах. Для цього реалізований сервіс JwtTokenService, який використовує симетричний ключ із конфігурації:

```
    internal class JwtTokenService : IJwtTokenService
    {
        private readonly IConfiguration _config;

        public JwtTokenService(IConfiguration config)
        {
            _config = config;
        }

        public string GenerateToken(User user, int playerProfileId)
        {
            var secret = _config["Jwt:Secret"];
            if (secret == null)
```

```
                throw new InvalidOperationException("Jwt secret is not
set");

            var key = Encoding.UTF8.GetBytes(secret);
            var credentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256);

            var claims = new[]
            {
                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
                new Claim(ClaimTypes.Email, user.Email),
                new Claim("PlayerProfileId", playerProfileId.ToString()),
                new Claim(ClaimTypes.Role, user.Role.ToString())
            };

            var token = new JwtSecurityToken(
                issuer: _config["Jwt:Issuer"],
                audience: _config["Jwt:Audience"],
                claims: claims,
                expires: DateTime.UtcNow.AddDays(1),
                signingCredentials: credentials);

            return new JwtSecurityTokenHandler().WriteToken(token);
        }
    }
```

За допомогою цих JWT токенів реалізовано авторизації. Клієнт при кожному запиті на сервер передає (за наявності аутентифікації користувача) цей токен, який в свою чергу перевіряється сервером через middleware. Система підтримує розмежування доступу за ролями, такими як RegularUser, Admin тощо. Це дозволяє ефективно обмежувати доступ до окремих API-методів за допомогою атрибутів Authorize.

4.4 Інтеграції з сервісами для отримання геоданих

Для реалізації функціональності, пов’язаної з локаціями, у розробленій програмній системі реалізована інтеграція з двома безкоштовними сторонніми сервісами: Nominatim [11] та GeoNames [12]. Їх використання дозволяє покращити користувацький досвід, забезпечити точну прив’язку до місцевості, а також реалізувати персоналізовані рекомендації на основі географічної близькості.

Кожна подія та ігровий клуб у системі прив’язуються до певної географічної локації. Для цього зберігається така інформація: OsmId, OsmType, Class, Logitude, Latitude. Ці дані дозволяють унікально ідентифікувати місце за допомогою

OpenStreetMap (через OSM ID і тип); визначати точне місцезнаходження об'єкта на мапі; розраховувати відстані між об'єктами; перевіряти і зберігати детальні адресні дані, такі як країна, місто, вулиця, номер будинку тощо.

Nominatim — це безкоштовний сервіс геокодування, який дозволяє шукати місця за назвою, координатами або OSM-ідентифікатором. У системі він використовується для:

- пошуку точних координат за назвою населеного пункту чи адреси (Forward Geocoding);
- отримання детальної адресної інформації за OSM ID (Reverse Geocoding або Lookup).

Клієнтська бібліотека Nominatim.API використовується для спрощеної інтеграції із сервісом через HTTP.

Наприклад, для пошуку користувачем локацій за текстовим запитом, використовується метод:

```
public async Task<List<LocationDetailsDto>> SearchLocationsAsync
(LocationSearchDto searchDto)
```

Для отримання інформації про конкретне місце — за OSM ID:

```
public async Task<LocationDetailsDto?> GetByOsmIdAsync(long osmId,
string osmType, string? prefferedLanguage)
```

GeoNames — це відкритий географічний сервіс, що надає структуровану інформацію про країни, регіони та населені пункти. У системі GeoNames використовується для:

- завантаження списку країн під час реєстрації або редагування профілю;
- динамічного пошуку міст у вибраній країні (наприклад, при виборі домашнього міста користувача).

Для цього реалізований власний HTTP-клієнт GeoNamesClient, який виконує запити до API. Використовується клас HttpClient, який отримуємо з контейнеру залежностей.

```
    internal class GeoNamesClient : IGeoNamesClient
    {
        private readonly HttpClient _httpClient;
        private readonly string _username;
        private readonly string _baseUrl = "http://api.geonames.org";
        private readonly int itemsLimit = 1000;

        public GeoNamesClient(HttpClient httpClient,
IOptions<GeoNamesOptions> options)
        {
            _httpClient = httpClient;
            _username = options.Value.Username;
        }

        public async Task<List<GeoNamesCountryDto>>
GetCountriesAsync(CancellationToken cancellationToken)
        {
            var url = $"{_baseUrl}/countryInfoJSON?username={_username}";
            var response = await _httpClient.GetAsync(url,
cancellationToken);
            if (!response.IsSuccessStatusCode)
            {
                throw new HttpRequestException($"Request to GeoNames API
error: {response.StatusCode}");
            }
            var json = await response.Content.ReadAsStringAsync();
            var geoNamesResponse =
JsonSerializer.Deserialize<GeoNamesResponse>(json, new
JsonSerializerOptions { PropertyNameCaseInsensitive = true });
            return geoNamesResponse?.Geonames ?? new
List<GeoNamesCountryDto>();
        }

        public async Task<List<GeoNamesPopulatedPlaceDto>>
GetCitiesByCountry(string country, int limit, string searchQuery,
CancellationToken cancellationToken)
        {
            var url =
$"{_baseUrl}/searchJSON?country={country}&username={_username}&featureClass
=P&maxRows={limit}&q={searchQuery}";
            var response = await _httpClient.GetAsync(url,
cancellationToken);
            if (!response.IsSuccessStatusCode)
            {
                throw new HttpRequestException($"Request to GeoNames API
error: {response.StatusCode}");
            }
            var json = await response.Content.ReadAsStringAsync();
            var geoNamesResponse =
JsonSerializer.Deserialize<GeoNamesPoplatedPlacesResponse>(json, new
JsonSerializerOptions { PropertyNameCaseInsensitive = true });

            return geoNamesResponse?.Geonames ?? new
List<GeoNamesPopulatedPlaceDto>();
        }
    }
```

Усі необхідні методи для геопошуку винесені в окремий сервіс, до якого через клієнтський застосунок отримує через контролер.

4.5 Інтеграція системи з зовнішнім API BoardGameGeek

Для забезпечення користувачів актуальною інформацією про настільні ігри в інформаційній системі було реалізовано інтеграцію з сервісом BoardGameGeek — найбільшою міжнародною базою даних настільних ігор [2]. Завдяки цій інтеграції адміністратор системи має можливість оперативно оновлювати інформацію про ігри, використовуючи зовнішнє джерело даних.

На жаль, BoardGameGeek надає лише XML API, що значно ускладнює взаємодію, оскільки сучасні інтерфейси здебільшого орієнтуються на JSON. Проте в рамках проєкту було реалізовано спеціальний HTTP-клієнт BggXmlClient, який виконує запити до BGG API, обробляє XML-відповіді за допомогою XmlSerializer та перетворює їх у DTO-моделі, зручні для подальшої обробки.

Для пошуку настільних ігор за ключовим словом виконується запит за адресою /search?query=назва&type=boardgame. У відповіді повертається перелік ігор, який фільтрується (наприклад, ігри без назви ігноруються), і формується список DTO-об'єктів із полями: BggId, назва гри та рік публікації.

Для отримання повної інформації про гру використовується запит /thing?id={bggId}&stats=1. У відповіді містяться детальні атрибути гри: назва, опис, рейтинг, мінімальний/максимальний вік і кількість гравців, середня вага (складність гри), час гри, посилання на дизайнерів, механіки, видавців, категорії, теми, інформацію про залежність від мови, сімейство гри тощо.

HTTP клієнт для запитів до API BGG створено аналогічно до клієнту для отримання геоданих з сервісу GeoNames.

Оскільки структура XML-відповідей BGG є складною та містить надлишкову або неоднозначну інформацію (наприклад, у вигляді голосувань спільноти), було реалізовано детальний парсинг полів з використанням регулярних виразів та логіки агрегації даних. Наприклад:

- рекомендована кількість гравців визначається шляхом обчислення, скільки голосів було за "Best" або "Recommended", порівняно з "Not Recommended";
- рекомендований вік вираховується як середньозважене значення з урахуванням кількості голосів;
- залежність від мови зіставляється з відповідним перерахуванням у системі, використовуючи мапінг із текстових значень, що надає API.

Крім того, реалізовано відокремлення тем (themes) від підкатегорій (subcategories) та основних категорій (categories), що дозволяє правильно структурувати інформацію в локальній базі даних. Для кожної категорії з рейтингом визначається позиція гри в межах відповідного сімейства (наприклад, Strategy, Family тощо).

Метод для перетворення даних, отриманих з BGG у DTO-модель, яка підтримується у системі:

```
     private BoardGameDetailsReturnDto
ConvertBggResponseBoardGameModel(BggGameDetailsItemResponseDto
bggGameDetails)
          {
              var isReimplementation = bggGameDetails.Links.Any(e => e.Type
== "boardgameimplementation");

              var communityRecommendedMinAge = bggGameDetails.Polls
                  .FirstOrDefault(e => e.Name == "suggested_playerage")?
                  .Results
                  .FirstOrDefault()?
                  .ResultList
                  .Select(e => new
                  {
                      e.NumVotes,
                      Age = int.Parse(Regex.Replace(e.Value,
CleanupIntegerRegex, ""))
                  })
                  .Aggregate(
                      (sum: 0, totalVotes: 0),
                      (acc, x) => (
                          sum: acc.sum + x.Age * x.NumVotes,
                          totalVotes: acc.totalVotes + x.NumVotes
                      ),
                      acc => acc.totalVotes == 0 ? 0 : (double)acc.sum /
acc.totalVotes
                  ) ?? null;

              var bestPlayers = bggGameDetails.Polls
                  .FirstOrDefault(e => e.Name == "suggested_numplayers")?
                  .Results
```

```
                .Select(r => new
                {
                    NumPlayers = r.NumPlayers,
                    BestVotes = r.ResultList
                        .FirstOrDefault(res => res.Value ==
"Best")?.NumVotes ?? 0
                })
                .OrderByDescending(x => x.BestVotes)
                .FirstOrDefault()?.NumPlayers;

            var goodPlayers = bggGameDetails.Polls
                .FirstOrDefault(e => e.Name == "suggested_numplayers")?
                .Results
                .Where(r =>
                {
                    int best = r.ResultList.FirstOrDefault(x => x.Value
== "Best")?.NumVotes ?? 0;
                    int recommended = r.ResultList.FirstOrDefault(x =>
x.Value == "Recommended")?.NumVotes ?? 0;
                    int notRecommended = r.ResultList.FirstOrDefault(x =>
x.Value == "Not Recommended")?.NumVotes ?? 0;

                    return (best + recommended) > notRecommended;
                })
                .Select(r => int.Parse(Regex.Replace(r.NumPlayers,
CleanupIntegerRegex, "")))
                .Distinct()
                .OrderBy(e => e)
                .ToList();

            var languageDependence = bggGameDetails.Polls
                .FirstOrDefault(e => e.Name == "languager_dependence")?
                .Results
                .FirstOrDefault()?
                .ResultList
                .OrderByDescending(r => r.NumVotes)
                .FirstOrDefault()?.Value;

            var languageDependenceValue = languageDependence == null ?
LanguageDependence.Unknown :
                LanguageDependenceMapping.TryGetValue(languageDependence,
out var value)
                ? value : LanguageDependence.Unknown;

            var familyName = bggGameDetails.Links
                .FirstOrDefault(e => e.Type == "boardgamefamily" &&
e.Value.StartsWith("Game:"))?
                .Value
                .Replace("Game: ", "");

            var boardGameReturnDto = new BoardGameReturnDto(
                0,
                bggGameDetails.Names.FirstOrDefault(n => n.Type ==
"primary")?.Value ?? bggGameDetails.Names.FirstOrDefault()?.Value ?? "",
                bggGameDetails.Description,
                bggGameDetails.YearPublished.Value,
                bggGameDetails.Statistics.Ratings.AverageWeight.Value,
                isReimplementation,
```

```
                    bggGameDetails.Image,
                    bggGameDetails.PlayingTime.Value,
                    bggGameDetails.MinPlayTime.Value,
                    bggGameDetails.MaxPlayTime.Value,
                    bggGameDetails.MinAge.Value,
                    communityRecommendedMinAge == null ? null :
(int)Math.Round(communityRecommendedMinAge.Value, 0),
                    bggGameDetails.MinPlayers.Value,
                    bggGameDetails.MaxPlayers.Value,
                    bestPlayers == null ? null :
int.Parse(Regex.Replace(bestPlayers, CleanupIntegerRegex, "")),
                    goodPlayers,
                    bggGameDetails.Id,
                    bggGameDetails.Statistics.Ratings.BayesAverage.Value,
                    bggGameDetails.Statistics.Ratings.UsersRated.Value,

bggGameDetails.Statistics.Ratings.Ranks.RankList.FirstOrDefault(r => r.Name
== "boardgame")?.Value ?? null,
                    languageDependenceValue,
                    languageDependenceValue.ToString(),
                    null,
                    null);

                var designers = bggGameDetails.Links
                    .Where(e => e.Type == "boardgamedesigner")
                    .Select(e => new ClassifierReturnDto(0, e.Value))
                    .ToList();

                var mechanics = bggGameDetails.Links
                    .Where(e => e.Type == "boardgamemechanic")
                    .Select(e => new ClassifierReturnDto(0, e.Value))
                    .ToList();

                var publishers = bggGameDetails.Links
                    .Where(e => e.Type == "boardgamepublisher")
                    .Select(e => new ClassifierReturnDto(0, e.Value))
                    .ToList();

                var subcategories = bggGameDetails.Links
                    .Where(e => e.Type == "boardgamecategory" &&
Subcategories.Contains(e.Value))
                    .Select(e => new ClassifierReturnDto(0, e.Value))
                    .ToList();

                var themes = bggGameDetails.Links
                    .Where(e => e.Type == "boardgamecategory" &&
!Subcategories.Contains(e.Value))
                    .Select(e => new ClassifierReturnDto(0, e.Value))
                    .ToList();

                var categories =
bggGameDetails.Statistics.Ratings.Ranks.RankList
                    .Where(e => e.Type == "family" &&
CategoriesMapping.ContainsKey(e.Name))
                    .Select(e => new CategoryWithPositionReturnDto(e.Value,
new ClassifierReturnDto(0, CategoriesMapping[e.Name])))
                    .ToList();
```

```
            var family = familyName == null ? null : new
ClassifierReturnDto(0, familyName);

            return new BoardGameDetailsReturnDto(
                boardGameReturnDto,
                categories,
                designers,
                mechanics,
                publishers,
                subcategories,
                themes,
                family);
        }
```

4.6 Інтеграція з електронною поштою

У рамках розробленої системи реалізовано повноцінну інтеграцію з електронною поштою. Вона використовується для автоматичного надсилання повідомлень користувачам з приводу підтвердження електронної пошти, скидання паролю або оновлень подій. Система дозволяє гнучко формувати HTML-листи з динамічним вмістом та додаванням зображень безпосередньо у тіло листа.

Для створення вмісту листів використовується бібліотека Scriban, яка дозволяє динамічно рендерити HTML-шаблони на основі переданих даних. Шаблони зберігаються у вигляді окремих HTML файлів, структура яких може містити спеціальні плейсхолдери для зображень та змінних. Зображення вставляються у шаблон через спеціальний плейсхолдер, який під час рендерингу замінюється на відповідне CID-зображення.

Формування листа виконується у класі EmailTemplateProvider. Він відповідає за:

- пошук відповідного шаблону;
- підстановку даних у шаблон;
- підключення зображень;
- обробку помилок рендерингу.

В результаті обробки формується HTML-вміст листа та список повних шляхів до вбудованих зображень.

Надсилання листів реалізовано за допомогою бібліотек MailKit та MimeKit, що забезпечують гнучку роботу з протоколом SMTP. Вся логіка відправки зосереджена у класі MailKitEmailSender, який:

- створює MIME-повідомлення;
- додає HTML-вміст листа;
- вбудовує зображення як inline attachments із відповідними Content-ID;
- підключається до SMTP-сервера з автентифікацією;
- надсилає лист і коректно завершує з'єднання.

Параметри підключення до поштового сервера (хост, порт, логін, пароль, SSL) зберігаються в конфігураційному файлі у розділі Email.

Код методу SMTP-клієнта для відправки листа:

```
      public async Task SendEmailAsync(
          IEnumerable<string> recipients,
          List<string> ccRecepients,
          string subject,
          string htmlBody,
          List<string> inlineImageFilePaths)
      {
          var message = new MimeMessage();
          message.From.Add(new MailboxAddress(_settings.Username,
_settings.Username));
          foreach (var to in recipients)
              message.To.Add(MailboxAddress.Parse(to));
          message.Subject = subject;

          var builder = new BodyBuilder { HtmlBody = htmlBody };
          foreach (var filePath in inlineImageFilePaths)
          {
              var image = await
builder.LinkedResources.AddAsync(filePath);
              image.ContentId = filePath;
              image.ContentType.MediaType = "image/png";
          }

          message.Body = builder.ToMessageBody();

          using var client = new MailKit.Net.Smtp.SmtpClient();
          await client.ConnectAsync(_settings.Host, _settings.Port,
_settings.UseSsl);
          await client.AuthenticateAsync(_settings.Username,
_settings.Password);
          await client.SendAsync(message);
          await client.DisconnectAsync(true);
      }
```

Щоб ізолювати логіку створення шаблонів та надсилання повідомлень від решти частин застосунку, реалізовано сервіс EmailService. Він виступає єдиною точкою для ініціації надсилання листів та дозволяє викликати метод SendEmailAsync, вказуючи лише тип повідомлення (EmailType), список одержувачів та словник параметрів для шаблону.

4.7 Підтримка інтернаціоналізації

З метою забезпечення зручного користувацького досвіду для користувачів, які спілкуються різними мовами, у системі реалізовано базову підтримку інтернаціоналізації. Це дозволяє динамічно адаптувати частину функціоналу програми під обрану користувачем мову інтерфейсу.

У кожного користувача в профілі зберігається налаштування мови у вигляді поля LanguageIso, що містить код мови за стандартом ISO (наприклад, "en", "uk"). Це значення можна отримати разом з іншими даними профілю користувача через API-запити з клієнтського застосунку. Користувач також має змогу змінити мову в особистому кабінеті — при редагуванні профілю.

Для визначення мови, яка має використовуватись у межах поточного запиту, реалізовано окремий сервіс доступу до поточного користувача. Сервіс містить метод:

```
     public async Task<CultureInfo> GetUserLanguageOrDefaultAsync(
CancellationToken cancellationToken)
     {
          var userClaims = _httpContextAccessor.HttpContext?.User;
          if (userClaims == null || !userClaims.Identity?.IsAuthenticated
== true)
               return new CultureInfo(LanguageConstants.DefaultLanguage);
          var userIdClaim =
userClaims.FindFirst(ClaimTypes.NameIdentifier)?.Value;
          if (!int.TryParse(userIdClaim, out var userId))
               return CultureInfoHelpers.GetDefaultCulture();
          var user = await _usersService.GetUserByIdAsync(userId,
cancellationToken);
          if (user == null)
               return CultureInfoHelpers.GetDefaultCulture();
          return CultureInfoHelpers.GetCultureInfoOrDefault(user.Language);
     }
```

Цей метод витягує дані про користувача з токена авторизації, перевіряє, чи користувач автентифікований, отримує мовний код, вказаний у профілі користувача, повертає відповідний об’єкт CultureInfo, або значення за замовчуванням, якщо користувач не вказав мову або не автентифікований.

Цей метод використовується у різних сервісах або контролерах, де потрібна мовна специфікація. Наприклад, при зверненні до зовнішнього геосервісу Nominatim, який підтримує багатомовність, у параметрах запиту можна вказати потрібну мову.

Це дозволяє повертати адреси або географічні дані (наприклад, назви вулиць чи населених пунктів) тією мовою, яку обрав користувач, що значно покращує якість користування застосунком (рисунки 4.1 – 4.2).

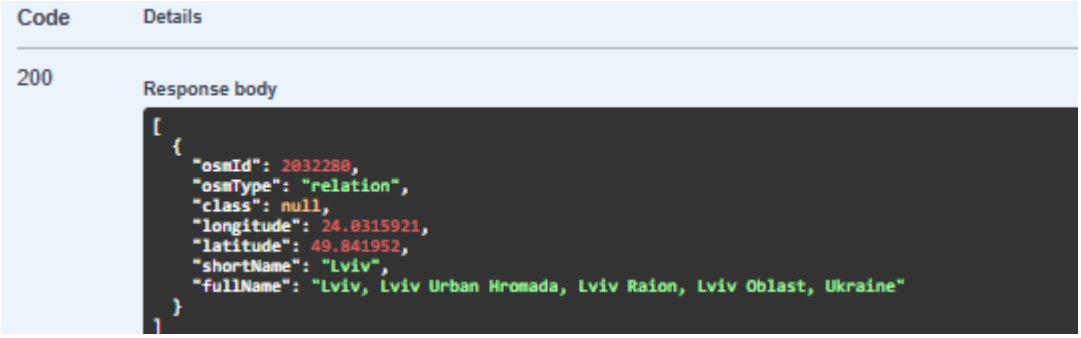


Рисунок 4.1 – Приклад отримання даних з зовнішнього API англійською мовою (рисунок виконано самостійно)

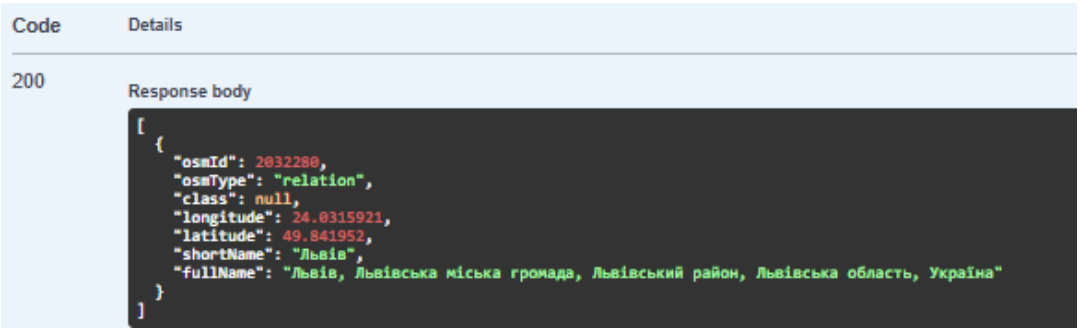


Рисунок 4.2 – Приклад отримання даних з зовнішнього API українською мовою (рисунок виконано самостійно)

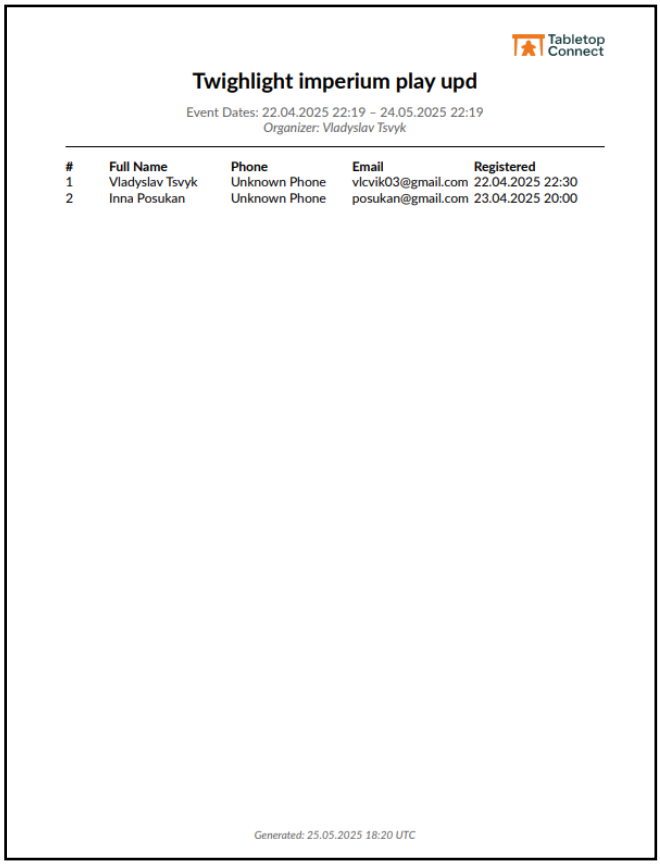
4.8 Генерація звітів у форматі PDF

У системі присутня можливість генерації звітів з форматі PDF для надання користувачам зручних і візуально структурованих даних. Це дозволяє легко зберігати, друкувати або надсилати звіти електронною поштою у вигляді готових документів.

Генерація звітів виконується за допомогою бібліотеки QuestPDF, яка забезпечує побудову PDF-документів на основі декларативного опису структури та стилю. QuestPDF має зручний API, підтримує адаптивне компонування елементів сторінки, дозволяє вставляти таблиці, зображення, текстові блоки та стилізувати їх відповідно до потреб додатку. Для ліцензування використовується Community License, що є безкоштовною для некомерційного використання.

Реалізовано два основних типи звітів.

- звіт зі списком учасників подій містить інформацію про назву події, дати початку і завершення, ім’я організатора, таблицю з даними кожного учасника: номер, повне ім’я, номер телефону, електронну пошту та дату реєстрації (рисунок 4.3);
- звіт про події користувача та популярні ігри на подіях містить інформацію про перелік організованих подій, перелік подій, у яких користувач брав участь та перелік найпопулярніших настільних ігор, у які грали на подіях за участю користувача (рисунок 4.4).


Tabletop Connect

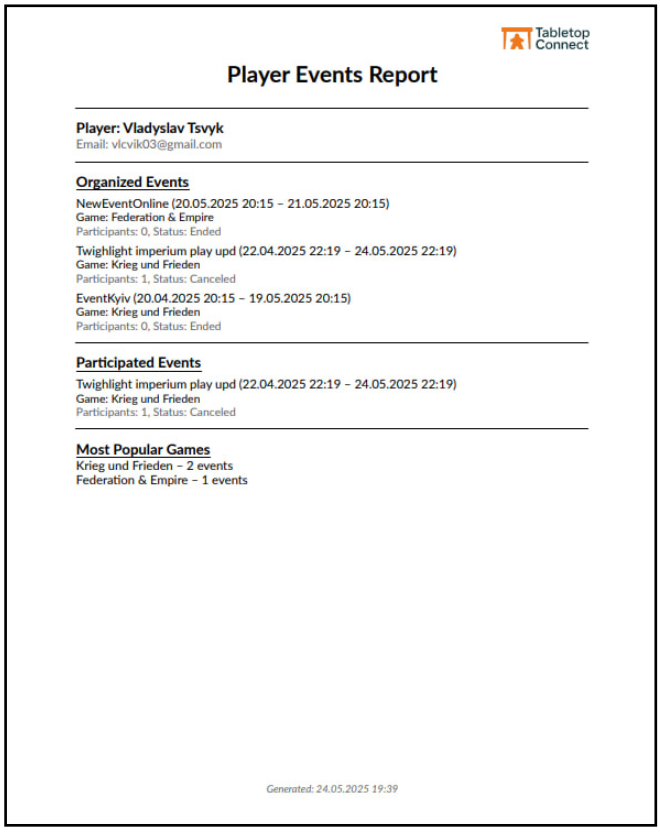
Twighlight imperium play upd

Event Dates: 22.04.2025 22:19 – 24.05.2025 22:19
Organizer: Vladyslav Tsvyk

#	Full Name	Phone	Email	Registered
1	Vladyslav Tsvyk	Unknown Phone	vlcvik03@gmail.com	22.04.2025 22:30
2	Inna Posukan	Unknown Phone	posukan@gmail.com	23.04.2025 20:00

Generated: 25.05.2025 18:20 UTC

Рисунок 4.3 – Приклад звіту зі списком учасників події (рисунок виконано самостійно)


Tabletop Connect

Player Events Report

Player: Vladyslav Tsvyk
Email: vlcvik03@gmail.com

Organized Events

NewEventOnline (20.05.2025 20:15 – 21.05.2025 20:15)
Game: Federation & Empire
Participants: 0, Status: Ended

Twighlight imperium play upd (22.04.2025 22:19 – 24.05.2025 22:19)
Game: Krieg und Frieden
Participants: 1, Status: Canceled

EventKyiv (20.04.2025 20:15 – 19.05.2025 20:15)
Game: Krieg und Frieden
Participants: 0, Status: Ended

Participated Events

Twighlight imperium play upd (22.04.2025 22:19 – 24.05.2025 22:19)
Game: Krieg und Frieden
Participants: 1, Status: Canceled

Most Popular Games

Krieg und Frieden – 2 events
Federation & Empire – 1 events

Generated: 24.05.2025 19:39

Рисунок 4.4 – Приклад звіту про події гравця (рисунок виконано самостійно)

Для генерації звітів використовується окремий сервіс QuestPdfGenerator, який реалізує інтерфейс IPdfGenerator. Усі методи для генерації звітів повертають масив байтів, що є готовим PDF-файлом, який можна зберігати або повертати клієнту в HTTP-відповіді з відповідним MIME-типом.

Приклад методу для генерації звіту:

```
    public byte[] GeneratePlayerEventsReport(PlayerEventsReportDto
playerEventsReportDto, DateTime reportGenerationDate)
    {
        var document = Document.Create(container =>
        {
            container.Page(page =>
            {
                page.Margin(50);

                page.Header().Column(column =>
                {

column.Item().AlignRight().Height(20).Image("PdfGenerator/Assets/logo.png",
ImageScaling.FitHeight);

                    column.Item().AlignCenter().Padding(10).Text("Player
Events Report")
                        .FontSize(22).Bold();

                    column.Item().PaddingVertical(10).LineHorizontal(1);
                });

                page.Content().Column(content =>
                {
                    content.Item().Text($"Player:
{playerEventsReportDto.PlayerDisplayName}").FontSize(14).Bold();
                    content.Item().Text($"Email:
{playerEventsReportDto.PlayerEmail}").FontSize(12).FontColor(Colors.Grey.Da
rken2);
                    content.Item().PaddingVertical(10).LineHorizontal(1);
                    content.Item().Text("Organized
Events").FontSize(14).Bold().Underline();
                    if (playerEventsReportDto.OrganizedEvents.Any())
                    {
                        foreach (var ev in
playerEventsReportDto.OrganizedEvents)
                        {
                            content.Item().PaddingTop(5).Text($"{ev.Name}
({ev.StartDate:dd.MM.yyyy HH:mm} – {ev.EndDate:dd.MM.yyyy HH:mm})")
                                .FontSize(12);
                            content.Item().Text($"Game: {ev.BoardGameName
?? "N/A"}").FontSize(11);
                            content.Item().Text($"Participants:
{ev.TotalParticipantsCount}, Status:
{ev.EventStatus}").FontSize(11).FontColor(Colors.Grey.Darken2);
                        }
                    }
```

```
                    else
                    {
                        content.Item().Text("No organized
events.").FontSize(12).Italic().FontColor(Colors.Grey.Darken2);
                    }

                    content.Item().PaddingVertical(10).LineHorizontal(1);
                    content.Item().Text("Participated
Events").FontSize(14).Bold().Underline();
                    if (playerEventsReportDto.ParticipatedEvents.Any())
                    {
                        foreach (var ev in
playerEventsReportDto.ParticipatedEvents)
                        {
                            content.Item().PaddingTop(5).Text($"{ev.Name}
({ev.StartDate:dd.MM.yyyy HH:mm} - {ev.EndDate:dd.MM.yyyy HH:mm})")
                                .FontSize(12);
                            content.Item().Text($"Game: {ev.BoardGameName
?? "N/A"}").FontSize(11);
                            content.Item().Text($"Participants:
{ev.TotalParticipantsCount}, Status:
{ev.EventStatus}").FontSize(11).FontColor(Colors.Grey.Darken2);
                        }
                    }
                    else
                    {
                        content.Item().Text("No participated
events.").FontSize(12).Italic().FontColor(Colors.Grey.Darken2);
                    }

                    content.Item().PaddingVertical(10).LineHorizontal(1);

                    content.Item().Text("Most Popular
Games").FontSize(14).Bold().Underline();
                    if (playerEventsReportDto.BoardGames.Any())
                    {
                        foreach (var game in
playerEventsReportDto.BoardGames)
                        {
                            content.Item().Text($"{game.BoardGameName} -
{game.Count} events").FontSize(12);
                        }
                    }
                    else
                    {
                        content.Item().Text("No data about popular event
board games.").FontSize(12).Italic().FontColor(Colors.Grey.Darken2);
                    }
                });
                page.Footer().AlignCenter().Text($"Generated:
{reportGenerationDate:dd.MM.yyyy HH:mm} UTC")

.FontSize(10).Italic().FontColor(Colors.Grey.Darken2);
            });
        });
        return document.GeneratePdf();
}
```

4.9 Реалізація наскрізної функціональності

Під час розробки було впроваджено низку наскрізних (cross-cutting) функціональностей, які підвищують якість коду, полегшують підтримку додатку та забезпечують стабільність у роботі системи. До таких функціональностей належать: логування, кешування, глобальна обробка винятків, валідація даних та централізоване налаштування конфігурації [13].

Для забезпечення збирання інформації про роботу системи, виникнення помилок та спрощення діагностики, в проекті реалізовано логування за допомогою вбудованого механізму ILogger<T>. Логування використовується в тому числі й у проміжному програмному забезпеченні (middleware) для глобального перехоплення помилок.

Приклад логування:

```
_logger.LogError(exception, "An unexpected error occurred");
```

Для зменшення навантаження на джерела даних та пришвидшення відповіді сервісу, реалізовано механізм кешування на основі IMemoryCache. Було створено обгортку над інтерфейсом IBoardGamesProvider, яка реалізує кешування у пам’яті (In-Memory Cache). Фрагмент коду для кешування списку настільних ігор з пов’язаними класифікаторами:

```
    public Task<List<BoardGame>>
GetBoardGamesWithRelatedClassifiersReadonlyAsync(CancellationToken
cancellationToken)
    {
        return _cache.GetOrCreateAsync(
            CacheKey,
            async entry =>
            {
                entry.SetOptions(cacheOptions);
                return await
_inner.GetBoardGamesWithRelatedClassifiersReadonlyAsync(cancellationToken);
            })!;
    }
```

Для валідації даних у бізнес-логіці було створено єдину модель ValidationResultDto, яка інкапсулює результати перевірки. У разі помилки кидається DomainValidationException, що дозволяє інтегруватися з глобальною обробкою.

Усі необроблені винятки перехоплюються централізовано за допомогою проміжного програмного забезпечення (ExceptionHandlingMiddleware). Це забезпечує єдиний підхід до обробки помилок, формування відповідей у форматі JSON та уникнення дублювання коду [14].

У випадку помилки валідації на рівні домену повертається повідомлення про конкретну помилку у форматі:

```
{
  "errors": [
    {
      "message": "Validation error message",
      "propertyName": "Property"
    }
  ]
}
```

Для зберігання параметрів середовища, таких як адреси API, SMTP-сервери чи ключі доступу, використано механізм опцій (IOptions<T>). Конфігураційні класи описують відповідні налаштування, які автоматично заповнюються при старті додатку.

Приклад реєстрації:

```
services.Configure<GeoNamesOptions>(configuration.GetSection(GeoNames
Options.ConfigurationSection));
```

4.10 Реалізація REST інтерфейсу

Серверна частина надає REST-інтерфейс, який забезпечує взаємодію з клієнтськими додатками. Усі запити обробляються через HTTP-протокол із використанням відповідних методів (GET, POST, PUT, DELETE) та повертають відповіді у форматі JSON.

Усі кінцеві точки (endpoints) розділені між контроллерами – Auth, Bgg, BoardGames, Classifiers, Events, Geo, PlayerProfiles, Reports, Users, GameClubs.

REST специфікація наведена у таблиці 4.1 (окрім кінцевих точок для роботи з класифікаторами).

Таблиця 4.1 – REST-специфікація серверної частини програмної системи (таблиця виконана самостійно)

POST /api/Auth/register	Реєстрація нового користувача
POST /api/Auth/login	Авторизація користувача
POST /api/Auth/google-login	Вхід через Google акаунт
POST /api/Auth/link-google	Прив'язка Google акаунту до профілю
POST /api/Auth/email-confirmation/send	Надсилання листа для підтвердження електронної пошти
POST /api/Auth/email-confirmation/confirm/{token}	Підтвердження електронної пошти за токеном
GET /api/Auth/settings	Отримання налаштувань користувача
PUT /api/Auth/settings	Оновлення налаштувань користувача
POST /api/Auth/{email}/password-reset/send	Надсилання листа для скидання пароля
POST /api/Auth/password-reset/confirm	Підтвердження скидання пароля
GET /api/Bgg/search	Пошук ігор на BGG
GET /api/Bgg/{bggId}	Отримання інформації про гру BGG за ідентифікатором
POST /api/BoardGames/filtered	Фільтрація списку настільних ігор
POST /api/BoardGames/names	Отримання назв настільних ігор
GET /api/BoardGames/{id}	Отримання інформації про настільну гру за ID
DELETE /api/BoardGames/{id}	Видалення настільної гри за ID
GET /api/BoardGames/recommendations	Отримання рекомендацій настільних ігор
GET /api/BoardGames/{bggId}/update-details	Оновлення деталей гри з BGG за ідентифікатором

Продовження таблиці 4.1

POST /api/BoardGames/{bggId}/create-or-update	Створення або оновлення гри за BGG ідентифікатором
POST /api/Events	Створення нового заходу
PUT /api/Events/{id}	Оновлення інформації про захід
DELETE /api/Events/{id}	Видалення заходу
GET /api/Events/{id}	Отримання інформації про захід
POST /api/Events/{id}/cancel	Скасування заходу
POST /api/Events/{eventId}/participations/{playerId}	Додавання учасника до заходу
DELETE /api/Events/{eventId}/participations/{playerId}	Видалення учасника з заходу
GET /api/Events/{id}/participations	Отримання списку учасників заходу
GET /api/Events/{id}/participations/details	Отримання детальної інформації про учасників заходу
POST /api/Events/filtered	Фільтрація заходів
GET /api/Geo/locations?search=&limit=	Отримання списку локацій за пошуковою строкою
GET /api/Geo/locations/{type}/{osmId}	Отримання конкретної локації за типом і OSM ID
GET /api/Geo/countries	Отримання списку країн
GET /api/Geo/cities?countryCode=&searchQuery= &limit=	Отримання списку міст за пошуковими даниими
GET /api/PlayerProfiles/favourite-games	Отримання улюблених ігор користувача
POST /api/PlayerProfiles/favourite-games/{boardGameId}	Додавання гри до улюблених
DELETE /api/PlayerProfiles/favourite-games/{boardGameId}	Видалення гри з улюблених
PUT /api/PlayerProfiles	Оновлення профілю користувача
GET /api/PlayerProfiles/participated-events	Отримання заходів, у яких користувач брав участь
GET /api/PlayerProfiles/created-events	Отримання заходів, створених користувачем
GET /api/PlayerProfiles/calendar-events	Отримання заходів у календарі користувача
GET /api/PlayerProfiles/my	Отримання даних власного профілю

Кінець таблиці 4.1

POST /api/PlayerProfiles/profile-picture	Завантаження фото профілю
GET /api/Reports/events/{eventId}/participants	Отримання звіту учасників заходу
GET /api/Reports/player/{playerId}/events	Отримання звіту заходів, у яких брав участь гравець
GET /api/Statistics/event-types	Отримання статистики щодо типів ігрових подій, що були створені
GET /api/Statistics/event-players	Отримання статистики щодо зареєстрованих на подію гравців
GET /api/Statistics/overall	Отримання загальної статистики щодо кількості подій, ігор та гравців
PUT /api/Users/{id}/role	Зміна ролі користувачу
GET /api/Users	Отримання загальної інформації про усіх користувачів системи
GET /api/GameClubs	Отримання даних про усі ігрові клуби
GET /api/GameClubs/my	Отримання даних про ігрові клуби користувача-власника клубів
POST /api/GameClubs	Створення ігрового клубу
PUT /api/GameClubs/{id}	Оновлення даних про ігровий клуб за ID
DELETE /api/GameClubs/{id}	Видалення ігрового клубу за ID

REST API інтерфейс системи є логічно структурованим, охоплює всі основні функціональні можливості — від автентифікації та управління користувачем до взаємодії з настільними іграми, заходами, геолокацією та генерацією звітів. Кінцеві точки побудовані відповідно до принципів REST, мають зрозумілу та послідовну URL-структуру, що полегшує інтеграцію з клієнтськими додатками та сприяє масштабованості системи [14].

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Обґрунтування вибору типів тестування

Для розробленої серверної частини системи проведено аналіз різних видів тестування, що дозволяє комплексно оцінити функціональність, стабільність і продуктивність розробленого REST API.

Існує багато типів тестування, які поділяються на кілька основних категорій: unit-тестування, інтеграційне тестування, функціональне тестування, навантажувальне тестування, безпекове та регресійне. Кожен з цих типів виконує свою роль у забезпеченні якості і надійності програмного продукту. Вибір конкретних типів тестування базувався на аналізі архітектури серверної частини, особливостей бізнес-логіки та очікуваних сценаріїв використання.

Для виконання тестування було обрано unit-тестування, інтеграційне тестування та функціональне тестування.

Unit-тестування зосереджується на перевірці окремих компонентів — класів, методів і сервісів. У контексті багаторівневої архітектури ASP.NET Core це дозволяє детально перевірити кожну бізнес-функцію, окремі обробники логіки та репозиторії без необхідності запуску всієї системи. Таке тестування забезпечує раннє виявлення помилок, дозволяє автоматизувати процес перевірки та підтримувати якість коду при подальшому розвитку проекту.

Інтеграційне тестування є логічним продовженням unit-тестування. Воно спрямоване на перевірку взаємодії між різними рівнями системи — контролерами, сервісами, шаром доступу до даних і базою даних. У системі, що підтримує роботу з базою даних та складною бізнес-логікою, інтеграційне тестування гарантує, що всі компоненти коректно працюють разом, а не тільки поодинці. Це допомагає виявити проблеми, які неможливо помітити, тестуючи компоненти ізольовано, наприклад, помилки в налаштуванні зв’язку з базою або неправильне формування запитів.

Функціональне тестування, орієнтоване на перевірку REST API, дозволяє контролювати поведінку системи з точки зору кінцевого користувача або зовнішніх клієнтів. Таке тестування здійснюється через HTTP-запити до кінцевих точок, що

імітують реальні сценарії взаємодії з сервером. Воно підтверджує, що API коректно обробляє запити, виконує бізнес-правила, повертає правильні коди стану і відповіді. Важливо, що функціональне тестування перевіряє систему в цілому, включно з механізмами автентифікації, авторизації, валідації даних і обробкою помилок.

5.2 Unit-тестування компонентів серверної частини

Unit-тести призначені для перевірки найменших одиниць коду — окремих класів, методів або функцій. Вони ізолюють ці компоненти від решти системи, щоб впевнитися, що кожен працює правильно сам по собі. Основний функціонал, який повинен бути покритий unit-тестами – це бізнес-логіка доменної моделі, сервіси, валідація, утиліти та допоміжні класи.

Unit-тести написані з використанням бібліотеки xUnit для .NET. При написанні тестів використовуємо патерн Arrange-Act-Assert (AAA). Приклад одного з unit-тестів для тестування доменної бізнес-логіки подій:

```
[Fact]
public void Cancel_Should_SetIsCanceled_IfNotCanceled()
{
    // Arrange
    var evt = Event.Create(
        "Cancelable Event", null, null, null,
        DateTime.Now.AddDays(1), DateTime.Now.AddDays(2),
        false, null, EventType.Meetup, null,
null, 1, null);

    // Act
    evt.Cancel();

    // Assert
    Assert.True(evt.IsCanceled);
}
```

5.3 Інтеграційне тестування

Інтеграційні тести перевіряють взаємодію між кількома компонентами системи, щоб упевнитися, що вони коректно працюють разом. Тут вже не тестується окремий клас ізольовано, а вся підсистема або кілька шарів (наприклад, контролер, сервіс та репозиторій разом).

Для більш просунутого інтеграційного тестування створюється окремий Test Fixture, який містить налаштування ізольованого середвища для тестів – в нашому випадку базу даних InMemory для симуляції реальної поведінки бази даних без зелажності від ней, а також зручні методи заповнення тестових даних і мок-об’єкти, щоб заповнити залежності сервісу, що тестується, імітаційними об’єктами.

Приклад тесту для перевірки отримання списку учасників подій сервісу подій наведено нижче:

```
[Fact]
public async Task GetEventParticipantsDetails_ReturnsExpectedResult()
{
    // Arrange
    var fixture = new EventsServiceFixture();

    var user = User.RegisterRegular("email@gmail.com", "asdf");
    fixture.DbContext.Set<User>().Add(user);
    await fixture.DbContext.SaveChangesAsync();

    var player = PlayerProfile.CreateRegular(user.Id, "John");
    fixture.DbContext.Set<PlayerProfile>().Add(player);
    await fixture.DbContext.SaveChangesAsync();

    var eventParticipation = new EventParticipation(1, player.Id, new
DateTime(2025, 1, 1));

fixture.DbContext.Set<EventParticipation>().Add(eventParticipation);
    await fixture.DbContext.SaveChangesAsync();

    // Act
    var result = await
fixture.EventsService.GetEventParticipantsDetailsAsync(eventParticipation.I
d, CancellationToken.None);

    // Assert
    Assert.Single(result);
    Assert.Equal("John", result[0].Nickname);
    Assert.Equal(1, result[0].Id);
}
```

5.4 Функціональне тестування кінцевих точок

Ще один вид тестування, який був проведений – це функціональне тестування кінцевих точок API для перевірки правильність обробки HTTP-запитів, що надходять до серверної частини, відповідно до вимог до системи. Метою є впевнитися, що конкретні кінцеві точки працюють згідно з очікуваною бізнес-логікою — наприклад, обробляють вхідні дані, взаємодіють із базою даних, повертають очікувані статуси та структури відповідей. Тестування охоплює як позитивні сценарії (успішне виконання операцій), так і негативні (обробка помилок, валідаційних повідомлень). Для тестування використовувався Swagger [15].

Приклад одного з текст-кейсів наведено у таблиці 5.1

Таблиця 5.1 – Тест кейс перевірки скасування події організатором (таблиця виконана самостійно)

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Скасування події організатором		
Власник тесту:	Цвик Владислав Іванович		
Дата створення:	26.05.2025		
Мета тесту:	Перевірка можливості скасування події її організатором та обробки помилкових сценаріїв		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	У базі даних є створена подія, а користувач, який її створив, авторизований	Подію можна скасувати через API	Пройдено
Скасування події			
№	Опис випадку	Очікуваний результат	Висновок
1	Надіслати POST-запит до /api/Events/{id}/cancel автором події	Сервер повертає HTTP-статус 200 (OK), подія позначається як скасована	Пройдено
2	Надіслати POST-запит до /api/Events/{id}/cancel, якщо подія вже скасована	Сервер повертає HTTP-статус 400 (Bad Request) з валідаційною помилкою	Пройдено

Кінець таблиці 5.1

3	Надіслати POST-запит до /api/Events/{id}/cancel, де id — неіснуючої події	Сервер повертає HTTP-статус 404 (Not Found) з повідомленням "Подію не знайдено"	Пройдено
4	Надіслати POST-запит до `/api/Events/{id}/cancel` користувачем, який не є автором	Сервер повертає HTTP-статус 400 (Bad Request) з повідомленням "Немає прав на скасування цієї події"	Пройдено
Результати тестування			
Дата прогону тесту: 26.05.2025		Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Проведене тестування серверної частини REST API було комплексним і охоплювало різні рівні системи. Було реалізовано unit-тестування, інтеграційне тестування та функціональне тестування, що дозволило всебічно перевірити роботу ключових компонентів. Такий підхід до тестування допоміг підвищити надійність, якість і стійкість системи до помилок, а також забезпечив впевненість у правильності реалізації бізнес-логіки на всіх рівнях.

ВИСНОВКИ

В результаті виконання роботи було розроблено програмну систему для персоналізації досвіду гравців у настільні ігри. Основною метою роботи було створення зручного інструменту, який допомагає гравцям знаходити настільні ігри відповідно до їх інтересів, уподобань та ігрового досвіду, а також організовувати події, відслідковувати улюблені ігри та отримувати персоналізовані рекомендації.

У процесі проектування було створено модель багаторівневої клієнт-серверної архітектури системи. Для реалізації клієнтської частини використовується фреймворк React.js. Серверна частина реалізується з використанням ASP.NET Core на мові програмування C#, що забезпечує високу продуктивність і масштабованість. Для зберігання даних було використано MS SQL Server у поєднанні з Entity Framework Core.

У системі було реалізовано авторизацію користувачів за допомогою протоколу OAuth2. Для взаємодії між клієнтом і сервером використовувався REST API, що дозволило створити гнучку та розширювану архітектуру.

Під час розробки застосовувалися основні принципи об’єктно-орієнтованого програмування (ООП), шаблони проєктування, принципи чистої архітектури, а також принципи SOLID для підвищення гнучкості та тестованості коду. Крім того, було впроваджено алгоритм рекомендацій настільних ігор та подій відповідно до профілю користувача.

У підсумку розроблено програмна система, яка покращує взаємодію користувачів між собою та покращує їх ігровий досвід, забезпечує персоналізований підхід до рекомендацій і сприяє розвитку спільноти гравців. Отримані результати підтвердили доцільність обраної архітектури, технологій та алгоритмів, а також продемонстрували практичні навички розробки сучасних веб-сервісів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Roeder, O. Crowdfunding Is Driving A $196 Million Board Game Renaissance [Електронний ресурс]. – URL: https://fivethirtyeight.com/features/crowdfunding-is-driving-a-196-million-board-game-renaissance/ (дата звернення: 02.06.2025).

2. Board Game Geek [Електронний ресурс]. – URL: https://boardgamegeek.com/ (дата звернення: 02.06.2025).

3. Try These Games [Електронний ресурс]. – URL: https://trythesegames.com/ (дата звернення: 02.06.2025).

4. Сервіс Geeker [Електронний ресурс]. – URL: https://geekerapp.com/ (дата звернення: 02.06.2025).

5. Хмарна платформа Microsoft Azure [Електронний ресурс]. – URL: https://azure.microsoft.com/ (дата звернення: 03.06.2025).

6. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. London: Longman, 2003. 534 p.

7. A data science competition platform Kaggle [Електронний ресурс]. – URL: https://www.kaggle.com/ (дата звернення 03.06.2025).

8. Muhammad Falah, Dewi Handayani. Content-based filtering using cosine similarity algorithm for alternative selection on training programs. Journal of Soft Computing Exploration. 2023. Vol. 4. P. 204 – 212.

9. Phil Woodward, Sam Woodward. Mining the Boardgamegeek. Significance. 2019. Vol. 16, Issue 5. P. 24 – 29.

10. Entity Framework Core Documentation [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення 03.06.2025).

11. Open-source geocoding service with OpenStreetMap data Nominatim [Електронний ресурс] – URL: https://nominatim.org/ (дата звернення: 03.06.2025).

12. The geographical database GeoNames [Електронний ресурс]. – URL: https://www.geonames.org/ (дата звернення: 03.06.2025).

13. Мартін Р. Чиста архітектура: Мистецтво розроблення програмного забезпечення. Харків: Фабула, 2021. 368 с.

14. Мартін Р. Чистий код: створення і рефакторинг за допомогою AGILE. Харків: Фабула, 2019. 416 с.

15. Swagger documentation [Електронний ресурс]. – URL: https://swagger.io/docs/ (дата звернення: 28.05.2025).

16. Матеріали кваліфікаційної роботи на GitHub [Електронний ресурс]. – URL: https://github.com/NureTsvykVladyslav/2025_B_PI_PZPI-21-6_Tsvyk_V_I