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. Клієнтська частина
(тема)

Виконав:
здобувач 4 року навчання
групи ПЗПІ-21-6

Інна ПОСУКАН
(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

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
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 Посукан Інні Ігорівні
(прізвище, ім'я, по батькові)

1. Тема роботи: Програмна система для персоналізації досвіду гравців у настільні ігри. Клієнтська частина
Затверджена наказом по університету від 19.05.2025р. № 397 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 12.06.2025
3. Вихідні дані до роботи Розробити клієнтську частину програмної системи для персоналізації досвіду гравців у настільні ігри з використанням технологій та мов програмування JS, HTML, CSS, React.JS
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування 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	09.05.2025	*виконано*
2	Створення специфікації ПЗ	12.05.2025	*виконано*
3	Проектування ПЗ	15.05.2025	*виконано*
4	Розробка ПЗ	30.05.2025	*виконано*
5	Тестування ПЗ	01.06.2025	*виконано*
6	Оформлення пояснювальної записки	05.06.2025	*виконано*
7	Підготовка презентації та доповіді	06.06.2025	*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 « 8 » травня 2025р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Зоя ДУДАР
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 67 стор., 10 рис., 5 табл., 11 джерел, 5 додатків.

REACT.JS, OAuth2, MSSQL, MEDIA QUERY, SCSS, JSX, REST API, JAVASCRIPT

Об’єкт розробки – програмна система для персоналізації досвіду гравців у настільні ігри.

Метою роботи є розробка програмної системи, яка надає користувачам можливість організовувати та брати участь у подіях, присвячених настільним іграм.

Результатом розробки стала програмна система «Tabletop Connect», яка дозволяє ефективно управляти подіями настільних ігор, автоматизуючи створення івентів, реєстрацію учасників та забезпечуючи зручний зворотний зв'язок для поліпшення організації заходів.

ABSTRACT

REACT.JS, OAuth2, MSSQL, MEDIA QUERY, SCSS, JSX, REST API, JAVASCRIPT

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

The purpose of the work is to create a software system that allows users to organize and participate in events dedicated to board games.

Solution method – React.js and SCSS for the client side, OAuth2 for authentication, REST API for server communication, MSSQL for data storage, and the use of media queries and JSX for responsive and dynamic interface design.

As a result of the development, a software system called "Tabletop Connect" was created, which enables efficient management of board game events by automating event creation, participant registration, and providing convenient feedback tools to improve event organization.

ЗМІСТ

ВСТУП

У сучасному світі настільні ігри переживають нову хвилю популярності. Вони стали не лише способом відпочинку, а й ефективним засобом соціалізації, розвитку стратегічного мислення, креативності та командної взаємодії. З огляду на це, дедалі більше людей шукають можливості долучатися до тематичних івентів, турнірів і зустрічей, присвячених настільним іграм.

Проте організація таких заходів часто є складною і не має зручного цифрового інструменту для планування, пошуку або реєстрації учасників. Більшість подій рекламуються вручну через соціальні мережі або месенджери, що створює бар'єри для залучення нових учасників та ускладнює процес взаємодії між гравцями та організаторами.

Метою цієї роботи є розробка програмної системи, яка дозволить користувачам легко створювати та знаходити події, пов’язані з настільними іграми, що відбуваються в реальному житті. Платформа також передбачає можливість реєстрації на івенти, перегляд деталей подій (місце, час, тип гри, організатор тощо) та зворотний зв'язок після участі.

Розроблена система має на меті об'єднати гравців та організаторів, спрощуючи комунікацію між ними та створюючи зручну систему для популяризації хобі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Загальний аналіз предметної галузі

Останніми роками настільні ігри активно відновлюють свою популярність і частіше стають вибором не лише вузького кола ентузіастів, а й широкої аудиторії. Вони виходять за межі простого дозвілля, виконуючи важливі соціальні та освітні функції.

Настільні ігри сприяють розвитку логічного та аналітичного мислення, творчих здібностей, покращенню комунікаційних навичок, формуванню навичок командної роботи, а також створюють сприятливе середовище для спілкування та обміну ігровим досвідом. У зв’язку з цим зростає інтерес до організації тематичних подій - клубних зустрічей, ігрових вечорів, турнірів тощо.

На сьогодні навколо настільних ігор сформувалася активна спільнота гравців, яка потребує постійної взаємодії. Попит на участь у таких заходах стабільно зростає, а разом із ним - і потреба у більш ефективних засобах організації подій.

1.2 Виявлення та вирішення проблем

Незважаючи на зростання популярності настільних ігор, організаційна інфраструктура, яка обслуговує ці заходи, залишається недостатньо розвиненою. Основні труднощі виникають під час планування ігор, пошуку цільової аудиторії, реєстрації учасників та координації дій між організаторами й гравцями. Більшість подій рекламуються та обговорюються через соціальні мережі, месенджери або форуми. Ці інструменти є зручними для загального спілкування, але не забезпечують належного рівня структурованості, масштабованості та ефективності. Вони не дозволяють повноцінно управляти інформацією, автоматизувати процес реєстрації, збирати статистику або формувати спільноту на основі інтересів. У результаті потенційні учасники можуть не дізнатися про події, які їх цікавлять, або пропустити можливість взяти участь у них через відсутність зручного застосунку.

1.3 Постановка задачі

Для вирішення вказаних проблем доцільно розробити програмну систему, яка дозволить керувати інформацією про ігрові події та забезпечить зручну взаємодію між усіма учасниками процесу.

Такий програмний продукт має надавати функціонал зручного створення, редагування та перегляду подій, забезпечувати швидку й доступну реєстрацію користувачів, а також містити рекомендаційну систему, що орієнтується на інтереси користувачів та їхню активність у подіях. Важливо, щоб система надавала повну інформацію про захід - назву гри, опис, категорію, місце й час проведення, контактні дані організаторів, кількість вільних місць тощо. Крім того, передбачається, що платформа повинна мати механізми фільтрації подій за інтересами користувача, геолокацією та жанрами ігор.

Розробка такого інструменту дозволить не лише полегшити процес організації та участі в настільних іграх, але й сприятиме подальшій популяризації цього хобі, розширенню аудиторії та покращення комунікації між учасниками спільноти.

1.4 Цільова аудиторія

Основною цільовою аудиторією цього проєкту стануть молоді люди віком від 16 до 30 років, які прагнуть відкривати нові ігри та розширювати своє коло знайомств. Для них важлива зручна та швидка реєстрація на заходи, а також можливість без зайвих зусиль дізнатися про майбутні зустрічі, не витрачаючи багато часу на пошук інформації в соціальних мережах.

Також платформою будуть активно користуватися досвідчені гравці віком від 25 до 45 років, які регулярно беруть участь у турнірах та спеціалізованих заходах, що стосуються класичних ігор. Вони прагнуть тестувати нові релізи та прототипи, а також активно обмінюватися відгуками.

Організатори заходів потребують потужного інструменту для публікації всіх необхідних деталей івенту, таких як місце, час, формат і вимоги до учасників.

Комерційні майданчики, такі як кафе-ігрові клуби, коворкінги та молодіжні центри, використовують платформу для залучення нових відвідувачів у завантажені дні тижня, аналізу популярності різних ігор та збору відгуків клієнтів з метою покращення своїх послуг.

Видавці та автори настільних ігор - від малих студій до незалежних дизайнерів - бачать у цій платформі можливість презентувати свої новинки безпосередньо зацікавленій аудиторії, проводити демонстрації прототипів і отримувати «живі» відгуки.

Поєднання потреб індивідуальних гравців, досвідчених хобі-гравців, організаторів, комерційних майданчиків та видавців дозволяє створити універсальну платформу з інтуїтивно зрозумілим інтерфейсом для реєстрації, що значно спростить організацію та популяризацію настільного хобі.

1.5 Аналіз аналогів

У процесі проєктування програмної системи особливо важливо приділити увагу ґрунтовному аналізу конкурентів. Це дозволить виявити слабкі та сильні сторони конкурентів та сформувати унікальну цінність продукту.

У сфері настільних ігор існує ряд платформ, які забезпечують підтримку спільнот гравців. Першим аналогом програмної системи з створення івентів для настільних ігор є BoardGameGeek (https://boardgamegeek.com/). Це програмна система, яка створена з метою забезпечення користувачів ресурсами для обміну інформацією, оглядами і рецензіями щодо настільних ігор (див.рис.1.1).

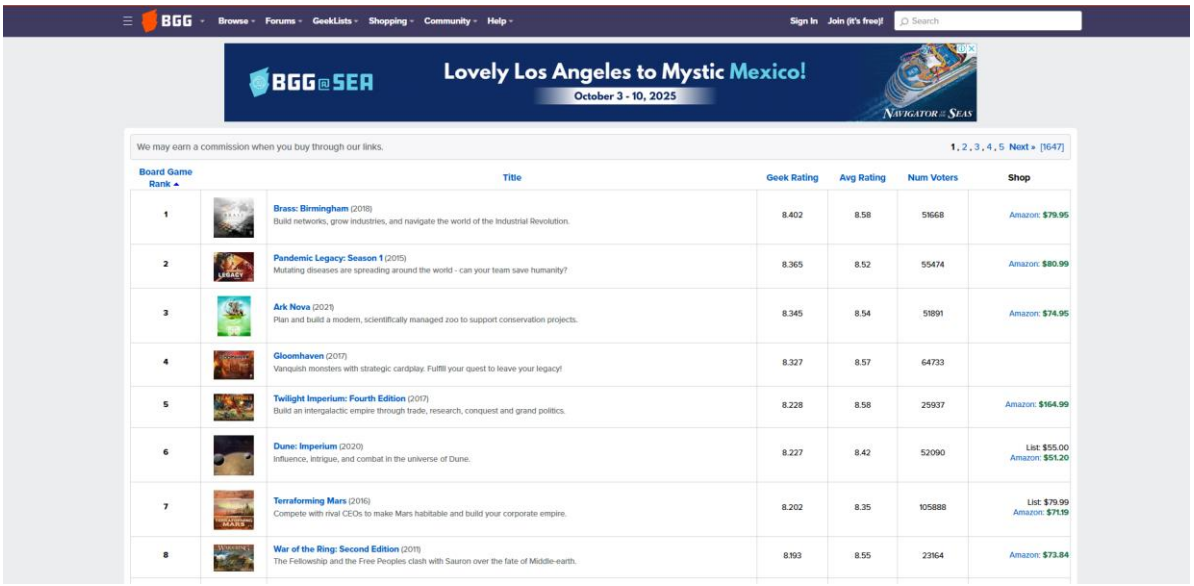

Рисунок 1.1 – Інтерфейс сервісу BGG

Серед сильних сторін цього сайту, варто відзначити велику базу даних, яка включає сотні тисяч різноманітних ігор, їх детальні описи, механіки, правила та рейтинг. Це дозволяє знайти всю необхідну інформацію про гру, що значно спрощує вибір під час покупки або організації заходу. Окрім того, BGG має активну спільноту користувачів, де можна обмінюватися досвідом, ділитися стратегіями та тактиками, а також брати участь в обговореннях на форумах.

Однак, ця платформа має й кілька суттєвих недоліків. Однією з основних проблем є застарілий інтерфейс сайту, який не є надто зручним для користувачів і вимагає додаткових зусиль для пошуку потрібних функцій.

Іншою значною вадою є обмежена функціональність для організації подій. Хоча платформа дозволяє створювати заходи та надає інструменти для реєстрації учасників, функції управління списками та взаємодії з гравцями мають суттєві обмеження. Організатори подій не мають спеціалізованих інструментів для ефективного управління процесом. Крім того, сайт не містить функцій для пошуку подій за географічним розташуванням, що ускладнює знаходження заходів поблизу.

Отже, для цієї програмної системи є можливості для вдосконалення, особливо в аспектах користувацького інтерфейсу.

Наступним аналогом є сайт Try These Games (https://trythesegames.com/), який зосереджений на пошуку настільних ігор на основі вподобань користувачів. Платформа пропонує добре розвинуту рекомендаційну систему, що базується на вподобаннях, зокрема через категоризацію ігор за типами, механіками та рівнем складністю (див.рис.1.2).

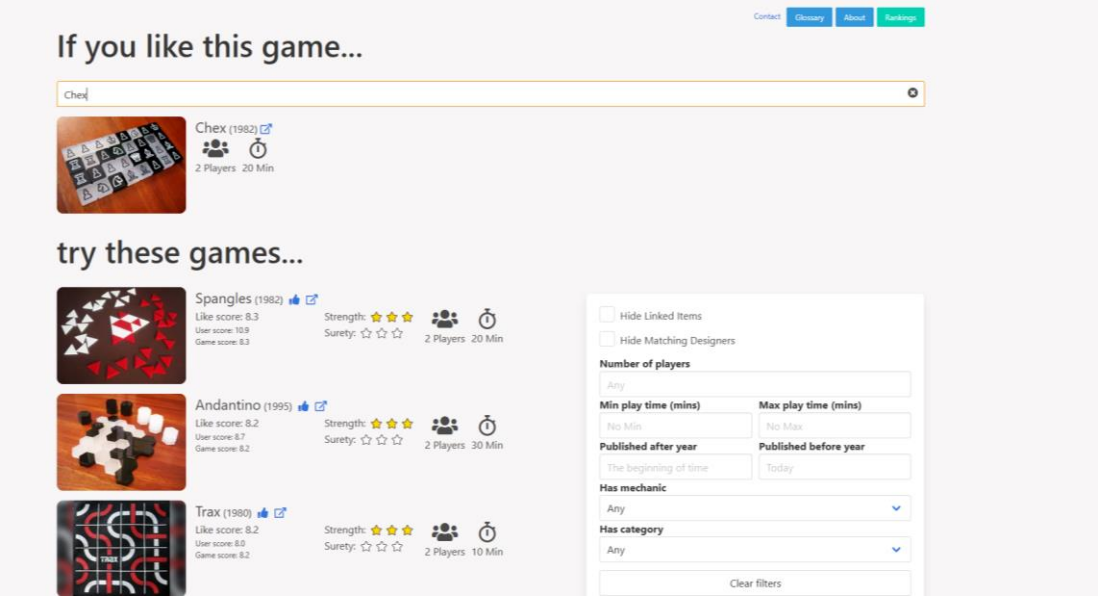

Рисунок 1.2 – Інтерфейс сервісу Try These Games

Дана платформа пропонує інтуїтивно зрозумілий інтерфейс, що є перевагою для новачків в світі настільних ігор. Водночас, сайт надає обмежену кількість інформації: відсутність оглядів, рецензій та форумів для обговорення. Для отримання більш детальної інформації про гру користувачам доводиться звертатися до зовнішніх джерел, таких як BoardGameGeek.

Крім того, сайт виконує виключно інформаційну функцію - можливості для організації або участі в ігрових сесіях відсутні, що зменшує його цінність для тих, хто бажає не лише ознайомитися з іграми, а й брати участь у них.

Ще один конкурентний сервіс - Geeker (https://geekerapp.com/). Цей ресурс надає користувачам можливість як створювати власні настільні ігри, так і долучатися до вже існуючих. Важливою перевагою є доступність сервісу як у вебверсії, так і на мобільних пристроях, що значно підвищує зручність користування.

Також, завдяки можливості вказати своє місцезнаходження, користувач може знаходити ігри та події поблизу, що сприяє активній участі у місцевих ігрових спільнотах.

Незважаючи на це, сервіс має і певні обмеження. Зокрема, база ігор тут значно менша порівняно з BGG, а також відсутні рецензії та розгорнуті відгуки користувачів, що ускладнює оцінку гри перед її вибором (див.рис.1.3).

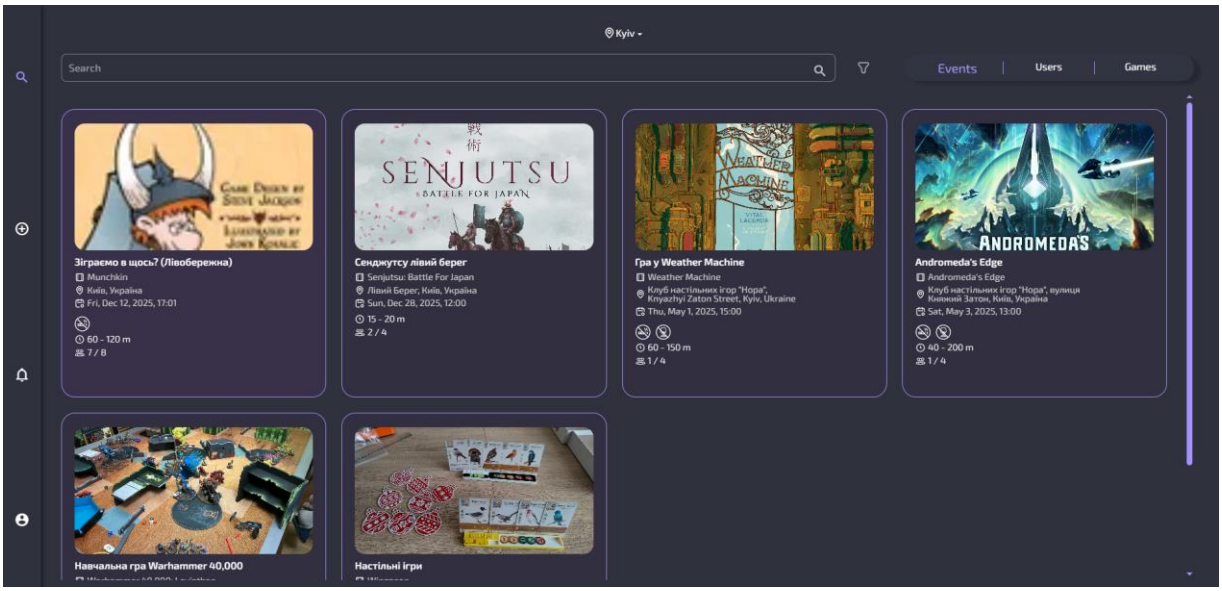

Рисунок 1.3 – Інтерфейс сервісу Geeker

Згідно аналізу аналогів, була сформована порівняльна таблиця розглянутих прикладів та майбутньої програмної системи (див.табл.1.1).

Таблиця 1.1 – Результат аналізу конкурентів

Назва критерію	Назва ресурсу			
	Tabletop (наша платформа)	BGG	Try These Games	Geeker
Кількість ігор	Висока	Дуже висока	Середня	Середня

Продовження таблиці 1.1

Рекомендації	Висока	Низька	Низька	Низька
Можливість грати онлайн	Ні	Ні	Ні	Ні
Організація подій	Так	Ні	Ні	Так
Підтримка багатомовності	Так	Так	Ні	Так
Спільнота	Активна	Дуже активна	Не активна	Активна
Пошук ігор за місцезнаходженням	Так	Ні	Ні	Так

Аналізуючи таблицю порівняння чотирьох платформ для настільних ігор - Tabletop, BGG, Try These Games та Geeker, можна зробити висновок, що Tabletop вигідно вирізняється серед інших платформ завдяки високій кількості ігор, наявності рекомендаційної системи та підтримці багатомовності. Крім того, Tabletop надає можливість організовувати події, чого не можна сказати про BGG, Try These Games чи Geeker. Проте, всі ці платформи мають спільну проблему - відсутність можливості грати онлайн, що обмежує їхню привабливість для сучасних користувачів.

1.6 Монетизація

Монетизація проєкту передбачає як напрями для звичайних користувачів і організаторів подій (B2C), так і співпрацю з ігровими клубами та видавцями настільних ігор (B2B). Для звичайних користувачів можна запропонувати недорогу підписку з базовими перевагами, а саме доступ до розширених персоналізованих рекомендацій настільних ігор.

Організатори подій можуть оформити окрему підписку, що надає можливість виділяти свої івенти у списках подій, а також отримувати доступ до

розширеної аналітики. Така аналітика може включати інформацію про аудиторію, рівень залученості учасників і статистику успішності попередніх заходів.

Ще один важливий напрям - це співпраця з ігровими клубами. Для них передбачено створення спеціального типу акаунту з розширеним функціоналом: власна сторінка клубу з окремим відображенням у сервісі, можливість створювати події від імені клубу з пріоритетним показом у пошуку, а також розміщення інформації про клуб у тематичних розділах. Це дозволяє клубам просувати себе безпосередньо серед цільової аудиторії.

У довгостроковій перспективі планується додати можливість оплати участі у подіях безпосередньо через платформу. Це відкриє нові можливості для організаторів, а платформа при цьому братиме невелику комісію (3–5% від суми оплати). Реалізація цього механізму вимагатиме додаткових технічних рішень - інтеграції платіжної системи, захисту транзакцій, гарантій повернення тощо.

Окремо розглядається взаємодія з видавцями настільних ігор. У разі зацікавленості їм буде запропонована можливість рекламування нових продуктів через платформу - наприклад, шляхом розміщення банерів, блоків з новинками або анонсів на сторінках ігор. Це може стати стабільним джерелом рекламного доходу та стимулювати додатковий інтерес до ігор серед користувачів.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Вимоги до програмної системи

Метою проєкту є розробка адаптивної програмної системи, призначеної для полегшення організації, пошуку та участі в заходах, присвячених настільним іграм. Програмна система повинна бути зручною та інтуїтивно зрозумілою як для організаторів, так і для звичайних користувачів. Передбачається баланс між функціональними можливостями та простотою у використанні.

Система дозволятиме реєстрацію користувачів, перегляд каталогу настільних ігор, створення ігрових подій, реєстрацію на події, отримання рекомендацій та формування звітів. Усі ці функції мають бути реалізовані з урахуванням вимог до зручності, безпеки, масштабованості та адаптивності інтерфейсу.

2.2 Функціональні вимоги

Для досягнення поставленої мети передбачено реалізація наступних функціональних вимог:

- реєстрація нових користувачів у системі;
- авторизація зареєстрованих користувачів;
- відновлення пароля через електронну пошту;
- перегляд каталогу настільних ігор;
- фільтрація ігор за жанром, віковою категорією, кількістю гравців, рейтингом тощо;
- сортування ігор за популярністю, новизною або алфавітом;
- додавання настільних ігор до списку “Обране”;
- створення ігрових подій з визначенням назви, дати, часу, місця проведення та вибраної гри;
- редагування та видалення створених подій організатором;
- перегляд доступних подій усіма зареєстрованими користувачами;
- реєстрація користувача на вибрану подію;

- отримання підтвердження про участь у події;
- генерація персоналізованих рекомендацій для користувачів на основі історії активності.

2.3 Нефункціональні вимоги

Після визначення функціональних вимог, які описують, що саме має робити система, важливо також врахувати нефункціональні вимоги :

- інтерфейс системи повинен бути інтуїтивно зрозумілим та зручним у користуванні для різних користувачів.
- забезпечення конфіденційності та безпеки обробки персональних даних користувачів (використання HTTPS, шифрування, хешування паролів).
- розмежування доступу до функціоналу відповідно до ролі користувача;
- сумісність із сучасними браузерами, включаючи Google Chrome, Mozilla Firefox, Microsoft Edge та Safari;
- адаптивність веб-інтерфейсу для коректного відображення на комп’ютерах, планшетах і мобільних телефонах;
- підтримка стабільної роботи системи при збільшенні кількості користувачів і подій без втрати продуктивності;
- швидкість завантаження сторінки не повинна перевищувати 2 секунд при середньому навантаженні;
- реакція інтерфейсу на дії користувача має бути миттєвою або з мінімально можливою затримкою;
- наявність механізмів обробки помилок.

Усі вищезазначені функції повинні бути враховані на етапах проєктування, реалізації та тестування веб-сайту, з дотриманням принципів зручності (usability), безпеки, масштабованості та адаптивності інтерфейсу.

2.4 Допущення та залежності

Розробка програмного забезпечення, як і будь-який технічний процес, передбачає необхідність врахування певних допущень і визначення залежностей, які впливають на успішне функціонування та реалізацію проєкту.

Після проведеного аналізу було сформульовано наступні допущення:

- наявність стабільного інтернет-підключення;
- сумісність із сучасними браузерами.

Крім того, були визначені наступні залежності:

- залежність від новітніх браузерів, а саме : Google Chrome, Microsoft Edge, Mozhila Firefox.

2.5 Середа оточення

У процесі аналізу предметної області було визначено необхідність створення програмної системи з чітким розділенням клієнтської та серверної частин, що забезпечить високу гнучкість, масштабованість і зручність у супроводі системи.

Клієнтська частина повинна бути реалізована за допомогою бібліотеки React, що дає змогу будувати динамічний і інтерактивний інтерфейс за допомогою компонентів, що дозволяють ефективно обробляти оновлення сторінок без перезавантаження. Для розробки клієнтської частини обрано Visual Studio Code як редактор, що підтримує роботу з JavaScript, а також має розширення для роботи з React.

Vite було вибрано для оптимізації процесу розробки завдяки його здатності миттєво відображати зміни без затримок, підтримці гарячого оновлення модулів і швидкому завантаженню в продакшн-режимі.

Основні технології для клієнтської частини: JavaScript - для реалізації логіки взаємодії, HTML - для структурування сторінок, SCSS - для стилізації інтерфейсу, що дозволяє використовувати змінні, вкладені стилі та інші переваги CSS-препроцесора для полегшення підтримки коду.

Для реалізації серверної частини обрано ASP.NET Core, оскільки він надає високу продуктивність, підтримує сучасні підходи до побудови веб-сервісів, включаючи REST API, а також інтегрується з іншими компонентами Microsoft.

Для розробки серверної частини використовуватиметься Microsoft Visual Studio, що надає всі необхідні інструменти для налаштування серверних компонентів, обробки запитів, налаштування безпеки та аутентифікації.

Як система управління базами даних було обрано Microsoft SQL Server (MSSQL), що забезпечує надійне зберігання і обробку даних, а також інтеграцію з платформою .NET через Entity Framework Core. Це дозволить ефективно організувати роботу з реляційною базою даних, забезпечивши стабільність і безпеку системи, а також підтримку масштабування при необхідності.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 UML проєктування програмного забезпечення

На етапі початкового проєктування програмної системи, призначеної для персоналізації досвіду гравців, було створено діаграму використання. Ця діаграма дозволяє візуалізувати взаємодію користувачів (акторів) із системою, показуючи, які функціональні можливості вона повинна надавати.

У системі існує три основні ролі користувачів, а саме: користувач, адміністратор, власник клубу. Для цих ролей були створені діаграми (див.рис.3.1-3.3).

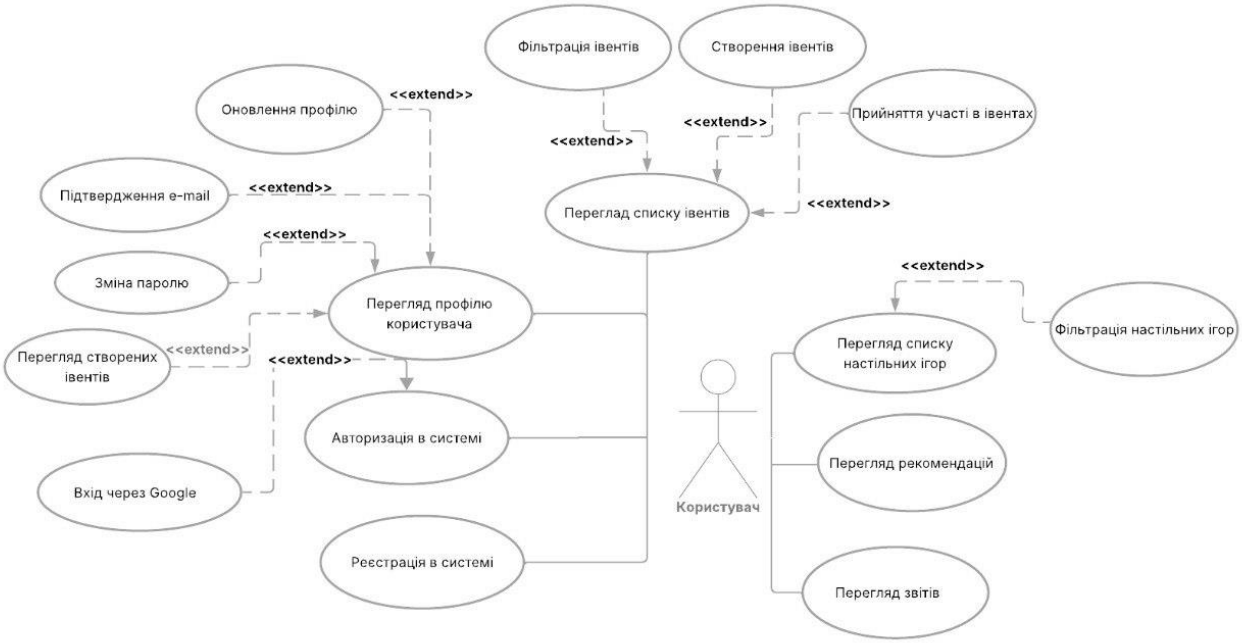

Рисунок 3.1 – Use-case діаграма для користувача

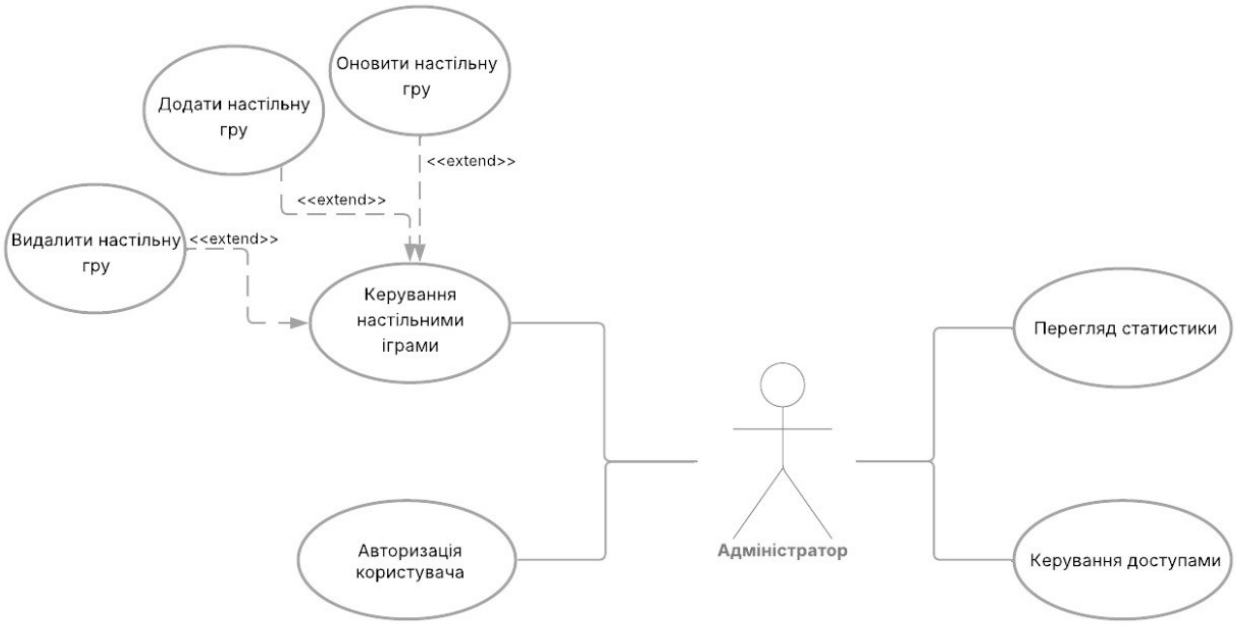

Рисунок 3.2 – Use-case діаграма для адміністратора

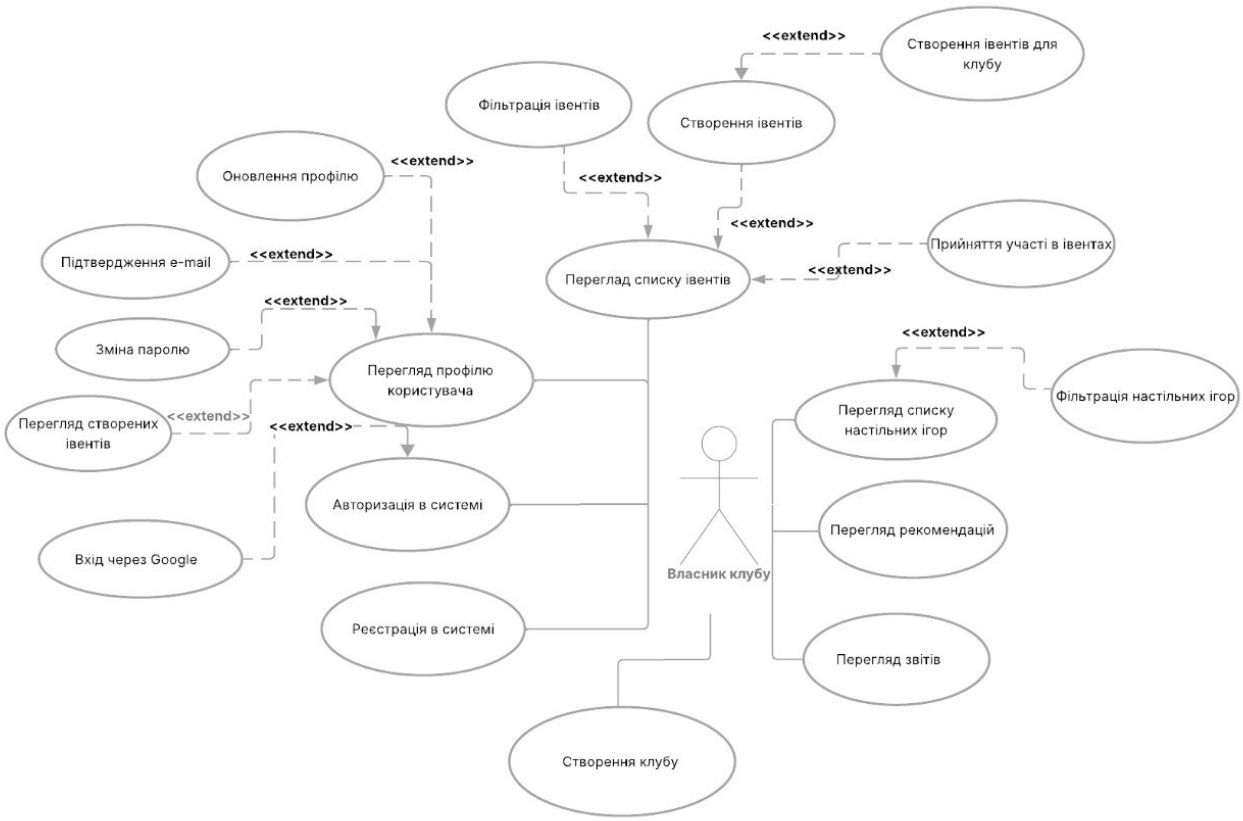

Рисунок 3.3 – Use-case діаграма для власника клубу

У межах розробленої програмної системи для персоналізації досвіду гравців у настільні ігри передбачено поділ користувачів за ролями. Основні ролі включають:

– звичайний користувач – це користувач, який взаємодіє з основними функціями веб-сайту.
– власник клубу – це користувач, який має розширені можливості, щодо організації події від імені ігрового клубу
– адміністратор – це користувач з повними правами доступу, який відповідає за загальне налаштування та підтримку системи. До його обов’язків входить управління контентом платформи, зокрема додавання, редагування та видалення настільних ігор, жанрів, категорій та іншої довідкової інформації.

Розподіл ролей у програмній системі дозволяє забезпечити гнучкість доступу до функціоналу, підвищити безпеку, та оптимізувати взаємодію користувачів з платформою. Кожна роль має чітко визначений список повноважень, що відповідає її цілям використання системи.

3.2 Взаємодія між частинами програмної системи

Користувач взаємодіє з веб-застосунком через браузер, в якому завантажується інтерфейс, створений за допомогою React. Цей інтерфейс дозволяє здійснювати запити до серверної частини через HTTP-протокол за допомогою RESTful API, реалізованого на платформі ASP.NET Core. Після отримання запиту сервер обробляє його, виконуючи відповідні операції, зокрема взаємодію з базою даних Microsoft SQL Server, де зберігаються основні дані системи. Для доступу до даних сервер використовує ORM-технологію Entity Framework Core, яка дозволяє працювати з даними у вигляді об'єктів. Результати запитів повертаються на сервер, де формуються у форматі JSON та надсилаються назад клієнтському застосунку.

Крім того, програмна система взаємодіє з зовнішніми API, такими як Google Auth API для аутентифікації користувачів, Nominatim API для роботи з географічними даними та картами з OpenStreetMap, а також з GeoNames API для доступу до географічної інформації, включаючи дані про міста, країни, координати та інші географічні об'єкти.

Для більш детального розуміння взаємодії програмної системи розглянемо діаграму компонентів (див.рис.3.4).

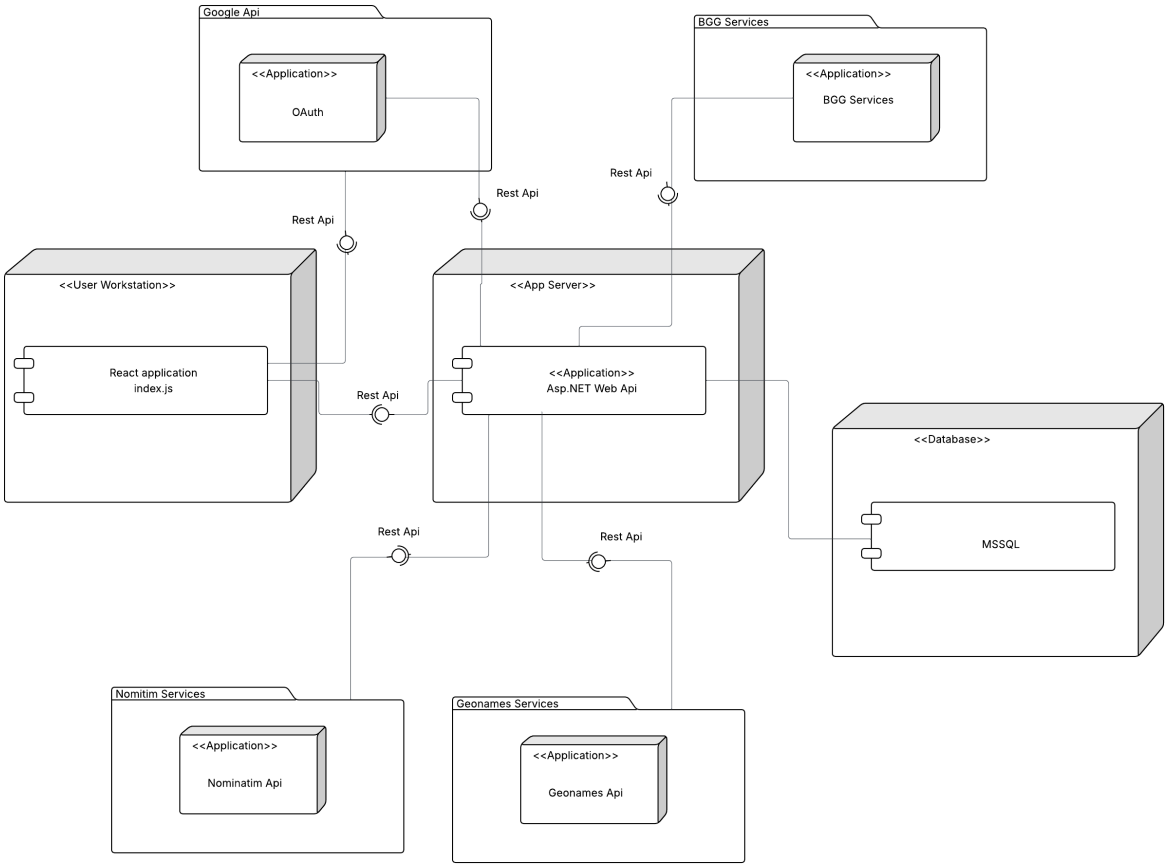

Рисунок 3.4 – Діаграма компонентів

Згідно рисунку 3.4 усі компоненти об'єднуються у єдину інфраструктуру, яка дозволяє реалізувати функціональність сучасного веб-застосунку, забезпечуючи ефективну взаємодію між користувачем, сервером, базою даних та зовнішніми сервісами.

3.3 Вибір архітектури програмної системи

Розроблювана система повинна використовувати клієнт-серверну архітектуру, яка є поширеним підходом для розробки веб-застосунків. Ця архітектура базується на чіткому розподілі завдань між двома компонентами: клієнтською та серверною частинами (див.рис.3.5).

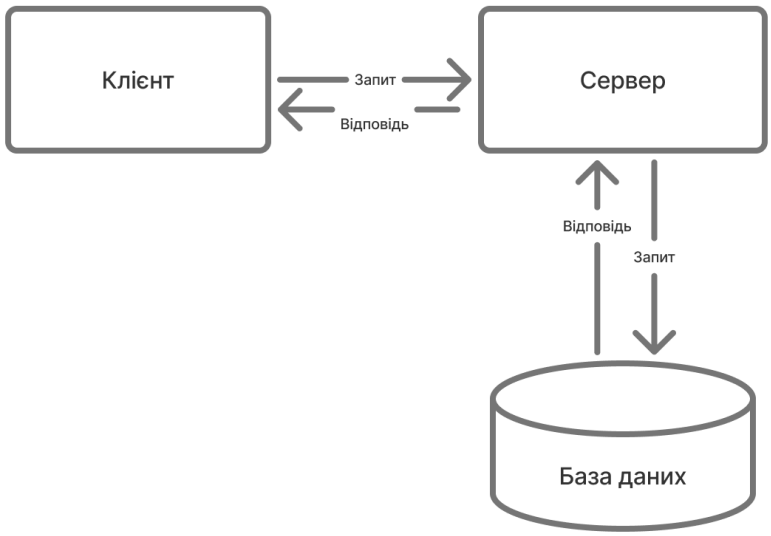

Рисунок 3.5 – Схематичне представлення клієнт-серверної архітектури

Клієнтська частина відповідає за взаємодію з користувачем, надаючи інтерфейс для перегляду, введення та редагування інформації. Основною її задачею є надсилання запитів до сервера та представлення даних у зрозумілому й зручному форматі для користувачів.

Серверна частина відповідає за обробку запитів, реалізацію бізнес-логіки та взаємодію з базою даних. Крім того, сервер виконує критично важливі функції, такі як автентифікація та авторизація користувачів, валідація даних, логування подій тощо.

Взаємодія між клієнтом і сервером відбувається переважно за допомогою HTTP/HTTPS протоколу, а дані передаються у форматі JSON. Така розподілена архітектура забезпечує високу гнучкість, масштабованість, повторне використання компонентів і можливість розширення функціоналу в майбутньому.

3.3.1 Архітектура клієнтської частини

Основою архітектури React є компонентний підхід, що передбачає розбиття інтерфейсу на незалежні частини - компоненти. Кожен компонент відповідає за свою частину UI та може мати свою власну логіку і стан. Компоненти можуть бути використані багаторазово в різних частинах програми, що значно знижує дублювання коду та покращує читабельність.

3.3.2 Патерн Container-Presenter

У клієнтській частині застосунку використовується архітектурний патерн Container–Presenter, який дозволяє чітко розділити відповідальність між логікою взаємодії з даними та їх візуальним представленням. Цей патерн передбачає поділ компонентів на дві категорії: контейнерні (Container) та презентерні (Presenter).

Контейнерні компоненти відповідають за всю логіку отримання, обробки та управління даними. Саме вони взаємодіють із зовнішніми джерелами інформації - API, глобальним станом або іншими сервісами. Контейнер отримує дані, перетворює їх за потреби та передає до презентера у вигляді властивостей.

Таким чином, контейнер несе відповідальність за бізнес-логіку та координує взаємодію між джерелами даних і відображенням. Презентер-компоненти, натомість, фокусуються виключно на візуалізації даних. Вони не містять логіки отримання інформації чи управління станом. Їхнє завдання — відображати ті дані, які вони отримали ззовні, згідно з дизайном та функціональними вимогами. Це робить такі компоненти передбачуваними, простими у повторному використанні та легкими для тестування.

3.3.3.Архітектура серверної частини

Серверна частина розроблюваної системи буде реалізована з використанням Onion архітектури та підходу Domain-Driven Design (DDD). Така організація дозволяє чітко структуризувати код, що значно полегшує підтримку, тестування

та масштабування системи в майбутньому, а також покращує взаємодію між різними компонентами програми.

Onion архітектура є патерном проєктування, що має на меті чітке розмежування відповідальностей між різними рівнями програми. У цій архітектурі система організована в кілька рівнів, кожен з яких має свою чітко визначену роль. Ядро системи містить основну бізнес-логіку, правила і сутності. Сервіси домену, які оперують цими сутностями, виконують ключові бізнес-операції, такі як управління подіями, рекомендаціями або реєстрацією користувачів на заходи. Це рівень, на якому виконуються основні бізнес-процеси, і він забезпечує виконання операцій, не залежачи від зовнішніх компонентів системи.

Інтерфейси та адаптери є рівнем, який забезпечує зв'язок між внутрішньою бізнес-логікою та зовнішніми компонентами, такими як бази даних. Він відповідає за обробку запитів від клієнтів, збереження та отримання даних, а також за взаємодію з іншими сервісами.

Зовнішні залежності знаходяться на останньому рівні архітектури і включають компоненти, які забезпечують реальну роботу системи в реальному середовищі, такі як з'єднання з базою даних, мережеві запити або відправка електронної пошти. Цей рівень дозволяє абстрагувати складнощі взаємодії з зовнішніми сервісами та зберігати незалежність бізнес-логіки від конкретних технологій.

Використання підходу Domain-Driven Design (DDD) дозволяє розвинути цю архітектуру ще більше, фокусуючись на глибокому розумінні бізнес-домену. В DDD система моделюється навколо проблем, які вирішує бізнес, а не навколо технічних рішень.

Ключовими концепціями в DDD є сутності, об'єкти-значення, агрегати, репозиторії, служби домену та події домену. Сутності є об'єктами з унікальною ідентичністю, які можуть змінювати свої властивості протягом свого життєвого циклу, але їх ідентичність залишається сталою.

Агрегати складаються з взаємопов'язаних сутностей і функціонують як єдиний логічний об'єкт. Репозиторії надають абстракцію для доступу до даних і зберігають агрегати, ізолюючи бізнес-логіку від складнощів роботи з базою даних.

Служби домену реалізують бізнес-логіку, яка не належить безпосередньо до конкретної сутності, а події домену сигналізують про важливі зміни в системі, що дає можливість ефективно організувати комунікацію між компонентами.

Інтеграція Onion архітектури та DDD створює потужну основу для розробки масштабованих і стійких до змін програмних систем. У такій архітектурі бізнес-логіка розміщується в самому центрі, і зовнішні залежності не можуть впливати на її роботу. Це дозволяє легко адаптувати систему до змін у технологічному середовищі, таких як заміна бази даних або оновлення API. Одночасно, система залишається гнучкою та підтримуваною завдяки чітко визначеним інтерфейсам і добре організованим рівням, що дозволяє ізолювати зміни в одній частині програми від впливу на інші частини.

Завдяки такій архітектурі досягається висока модульність і тестованість системи, що дозволяє ефективно проводити юніт-тестування та інтеграційне тестування, а також забезпечує гнучкість у масштабуванні програми.

3.4 Проєктування дизайну веб-сайту у Figma

Проєктування користувацького інтерфейсу веб-сайту (UI) здійснювалося за допомогою хмарного інструменту Figma, який на сьогодні є одним із найпопулярніших засобів для створення макетів для програмних систем.

Вибір саме цього інструменту був обґрунтований його зручністю для спільної роботи, можливістю швидкого прототипування та наявністю вбудованих засобів для швидкої розробки.

Після проведення аналізу цільової аудиторії були визначені ключові вимоги до дизайну програмного продукту та розроблено візуальні макети основних сторінок веб-сайту Tabletop (див.рис. 3.6).

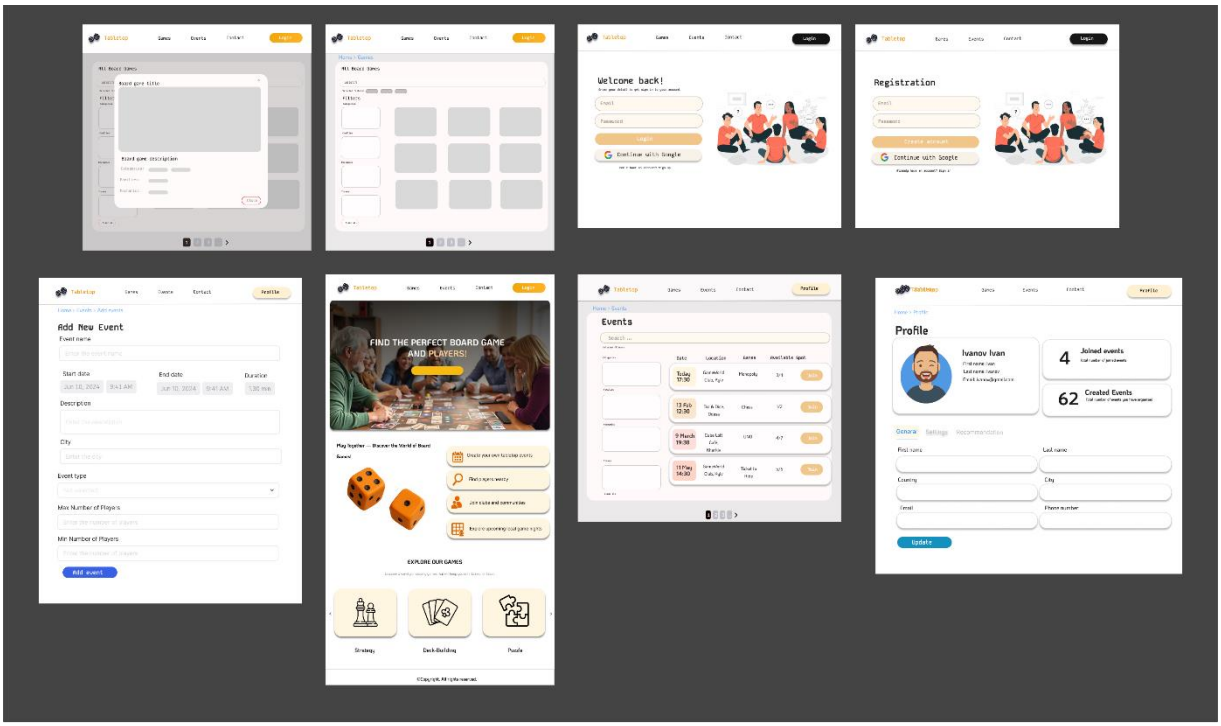

Рисунок 3.6 – Дизайн веб-сайту у Figma для десктопної версії

Особливу увагу варто приділити мобільній версії інтерфейсу, оскільки значна частина цільової аудиторії взаємодіє із сайтом саме з мобільних пристроїв. З огляду на це, інтерфейс було адаптовано також для невеликих екранів (див. рис. 3.7).

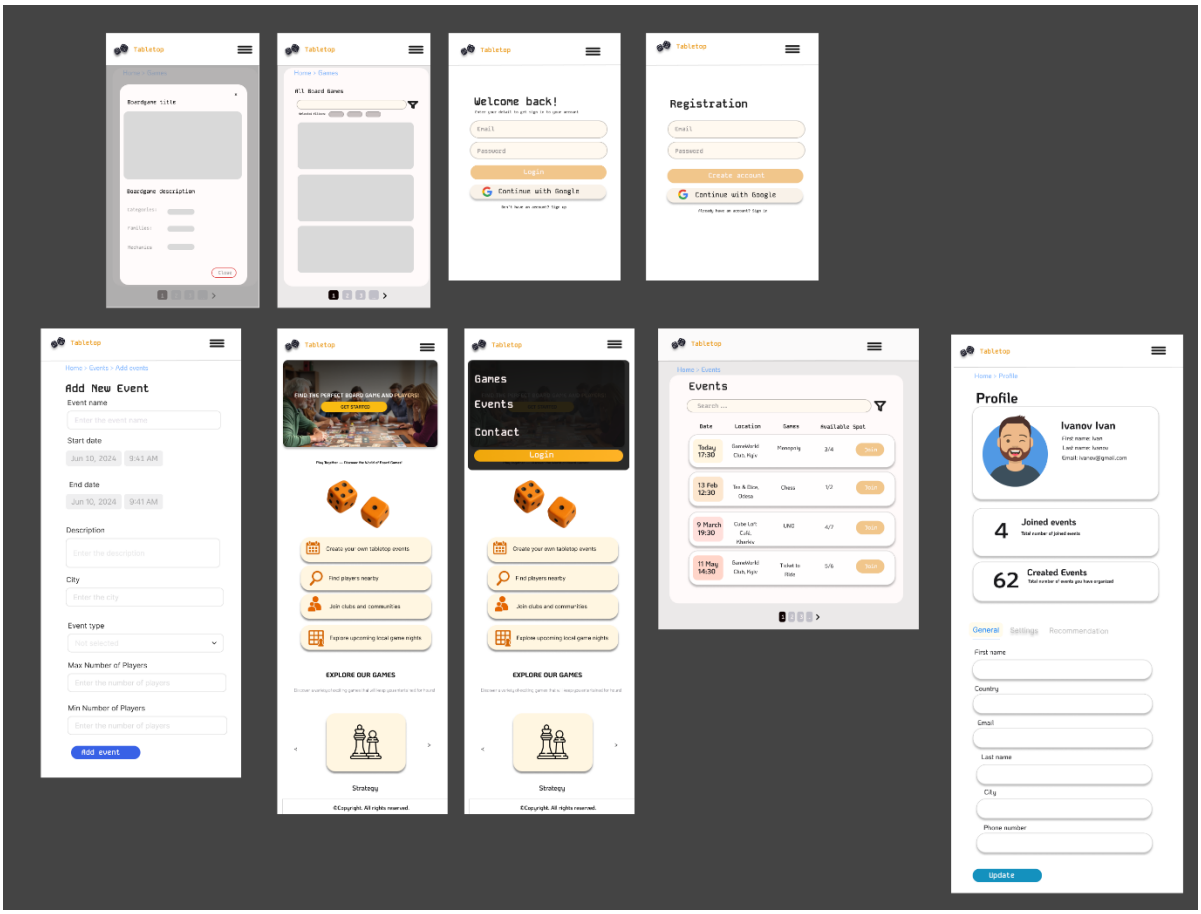

Рисунок 3.7 – Дизайн веб-сайту у Figma для мобільної версії

Як видно з рисунка 3.7, мобільна версія зберігає всі ключові функціональні елементи, водночас забезпечуючи простоту навігації та зручність взаємодії.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Менеджер пакетів npm

У процесі розробки клієнтської частини застосунку було використано менеджер пакетів npm (Node Package Manager). Це найважливіший інструмент, який забезпечує ефективну роботу з зовнішніми бібліотеками, залежностями та скриптами в JavaScript.

npm автоматизує процес інсталяції, оновлення, конфігурації та видалення пакетів, що значно прискорює розробку, робить її структурованішою та керованою. Менеджер npm працює разом з середовищем Node.js, що дозволяє запускати JavaScript-код не лише у браузері, а й на сервері. Це робить npm універсальним інструментом як для клієнтської, так і для серверної частини застосунку.

Під час реалізації проєкту через npm є можливість додавати такі залежності, як react, react-router-dom, react-bootstrap, @react-oauth/google, axios, jsonwebtoken тощо. Всі ці бібліотеки після встановлення будуть автоматично записані у файл package.json, який виступає головним конфігураційним документом проєкту. Він містить перелік усіх встановлених пакетів, скриптів запуску, інформацію про версію веб-застосунку та дозволяє швидко відновити робоче середовище проєкту на іншому пристрої.

4.2 Створення веб-застосунку за допомогою React

Розробка веб-застосунку починається з налаштування робочого середовища, для чого зазвичай використовується інструмент Create React App. Він дозволяє швидко ініціалізувати проєкт без потреби вручну налаштовувати конфігурації таких інструментів, як Webpack чи Babel. Після виконання команди створення нового проекту, структура папок і файлів генерується автоматично, включаючи базову конфігурацію і необхідні залежності.

4.3 Використання React Bootstrap

Під час створення клієнтської частини застосунку було вирішено використовувати бібліотеку React Bootstrap - сучасну реалізацію класичного CSS-фреймворку Bootstrap, яка інтегрована з React.

Ця бібліотека надає набір компонентів, які можуть бути безпосередньо використані в React-проєкті, замість того, щоб застосовувати класичні HTML-елементи з класами, як у традиційному Bootstrap. На відміну від звичайного Bootstrap, де елементи оформлюються за допомогою CSS-класів, React Bootstrap надає повноцінні React-компоненти, які підтримують декларативний підхід розробки та без проблем налаштовуються через пропси.

Такий підхід зменшує обсяг кастомної верстки та прискорює створення адаптивного, зручного та функціонального інтерфейсу. Крім того, використання React Bootstrap допомагає підтримувати чистоту і читабельність коду, оскільки компоненти React Bootstrap мають зрозумілу структуру і легко інтегруються з іншими частинами проекту. Компоненти React Bootstrap є модульними, що дозволяє швидко додавати чи змінювати елементи інтерфейсу, без порушення логіки програми.

Щоб підключити React Bootstrap до проєкту, в першу чергу необхідно встановити бібліотеку за допомогою npm. Після цього імпортувати стилі Bootstrap у файл, де ініціалізується додаток. Тепер можна починати використовувати компоненти React Bootstrap, змінюючи стандартні HTML-елементи на вже готові компоненти, такі як Button, Navbar, Card, Modal та інші.

4.4 Використання Context API для аутентифікації

У межах розробки клієнтської частини було прийнято рішення використовувати контекст React Context API для керування автентифікацією користувача. Контекст дозволяє створити глобальне сховище даних, доступне з будь-якої точки програми, минаючи необхідність передавати дані через численні рівні вкладених компонентів. Це особливо корисно у випадках, коли певна

інформація - зокрема, стан автентифікації, токен доступу чи ідентифікатор користувача - має бути доступною одночасно у багатьох частинах інтерфейсу (наприклад, у хедері, , налаштуваннях профілю тощо).

У створеному контексті відбувається зберігання JWT-токена, витягування ідентифікатора користувача з токена, оновлення інформації профілю та управління статусом входу/виходу. Наприклад, при вході в систему токен зберігається в localStorage, передається в контекст, а з нього автоматично оновлюється глобальний стан. Це дозволяє уникнути дублювання логіки в різних компонентах, а також спрощує повторне використання компонентів у майбутньому.

Крім того, використання Context API дозволяє легко масштабувати систему: наприклад, у разі додавання авторизації на основі ролей, багатомовності або іншого глобального стану, його також можна розмістити в окремих контекстах. Контекст добре інтегрується з React Router, що дозволяє будувати захищені маршрути, які автоматично перевіряють автентифікацію перед переходом. Таким чином, використання контексту в React значно підвищує структурованість коду, зменшує ризик помилок, пов’язаних із передачею стану, та забезпечує гнучкість у керуванні автентифікацією, що робить його ефективним і доцільним рішенням у проєкті.

4.5 Використання хуків та пропсів

React хук та пропси є двома основними концепціями, які дозволяють ефективно працювати зі станом компонентів і передавати між ними дані Хуки дозволяють інтегрувати функціональність у функціональні компоненти, надаючи їм можливість керувати станом, виконувати побічні ефекти та взаємодіяти з іншими частинами додатка. Одним з найпопулярніших хуків є useState, який дозволяє зберігати значення стану у компоненті. Використовуючи його, можна легко змінювати стан компонента та оновлювати інтерфейс. Наприклад, за

допомогою useState можна створити форму, де введені користувачем дані зберігаються у стані компонента.

Іншим важливим хукем є useEffect, який використовується для виконання побічних ефектів, таких як запити до серверу, підписки на події чи оновлення DOM. Хук useEffect запускається після кожного рендеру компонента, або лише при зміні значень. Це дозволяє контролювати, коли саме потрібно виконувати певні операції, не порушуючи логіку рендерингу.

Пропси (від "properties") використовуються для передачі даних від батьківського компонента до дочірнього. Пропси можуть містити будь-яку інформацію, таку як значення, функції або навіть інші компоненти, які потрібні для рендеру дочірнього компонента. Вони є важливою частиною архітектури React, дозволяючи створювати гнучкі компоненти. Хуки та пропси значно спрощують процес розробки в React, роблячи компоненти більш чистими та гнучкими. Вони дозволяють створювати адаптивні та масштабовані інтерфейси без зайвої складності, забезпечуючи при цьому високу продуктивність та можливість для подальшого розширення.

4.6 Реалізація навігації по веб-сайту

Навігація у веб-застосунку є важливою складовою, що забезпечує зручність переміщення між сторінками інтерфейсу без необхідності повного перезавантаження сторінки.

У даному проєкті для організації навігації використано бібліотеку React Router, яка є стандартним інструментом маршрутизації в React-застосунках. Вона дозволяє будувати односторінкові додатки (SPA — Single Page Application), в яких перехід між сторінками відбувається миттєво, без оновлення всієї сторінки, що позитивно впливає на продуктивність і користувацький досвід.

Приклад створення навігації в веб-застосунку Tabletop:

```
const AppRouter = () => {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
```

```
        <Route path="/auth" element={<Auth />} />
        <Route path="/events" element={<Events />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/addevent" element={<AddEvent />} />
        <Route path="/all" element={<AllBoardGames />} />
      </Routes>
    );
};

export default AppRouter;
```

Навігація реалізована за допомогою компонентів Routes і Route, які входять до складу бібліотеки react-router-dom. Кожен маршрут вказує, яка сторінка повинна відображатися за конкретним URL. Наприклад, при переході на головну сторінку веб-застосунку (шлях /), користувачеві відображається компонент Home. Аналогічно, для сторінки авторизації визначено шлях /auth, для подій - /events, для профілю користувача - /profile, додавання події - /addevent, а також для перегляду каталогу настільних ігор - /all.

Всі ці шляхи пов’язані з окремим React-компонентом, який відповідає за логіку та інтерфейс відповідної сторінки.

4.7 Відправлення запиту з клієнту на сервер

Під час створення веб-застосунку на React важливою складовою є організація взаємодії між клієнтською частиною та сервером. Це забезпечує отримання, оновлення або фільтрації даних відповідно до дій користувача. У даному проєкті відправлення запитів на сервер реалізовано з використанням бібліотеки Axios, яка є зручною та популярною для HTTP-запитів.

Приклад відправки запиту на сервер в веб-застосунку Tabletop:

```
export async function getAllBoardGames(pageNumber, pageSize,
search, filters) {
  try {
    const response = await
axios.post(`${BASE_URL}/BoardGames/filtered`, {
      pageNumber,
      pageSize,
      search,
      filter: filters,
    });

    return response.data;
```

```
    } catch (error) {
      console.error('Error fetching board games:', error);
      throw error;
    }
  }
```

У наведеному прикладі продемонстровано функцію getAllBoardGames, яка відповідає за надсилання POST-запиту до серверу за адресою /BoardGames/filtered. Ця функція приймає параметри pageNumber, pageSize, search і filters, які використовуються для реалізації пагінації, пошуку та фільтрації даних про настільні ігри. Вони формуються у вигляді тіла запиту та передаються на сервер. Використання async/await забезпечує асинхронну поведінку функції, що дозволяє не блокувати основний потік програми під час очікування відповіді від сервера.

У разі безпомилкового виконання запиту функція повертає response.data, який містить необхідні дані. Якщо ж під час запиту виникає помилка вона обробляється в блоці catch, де фіксується повідомлення про помилку і генерується виняток за допомогою throw error, щоб передати його для подальшого оброблення на рівні відповідного компонента або загального механізму обробки помилок.

4.8 Реалізація інтернаціоналізації за допомогою react-i18next

У клієнтській частині застосунку реалізована підтримка української та англійської мов для інтерфейсу користувача. Для цього використовується бібліотека react-i18next, яка є популярним і гнучким рішенням для організації локалізації у React-додатках.

У проєкті створено окремий файл з конфігурацією, у якому відбувається ініціалізація та підключення ресурсів перекладу.

```
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import en from '../locales/en.json';
import ua from '../locales/ua.json';

const resources = {
  en: { translation: en },
  uk: { translation: ua },
};

i18n
```

```
  .use(initReactI18next)
  .init({
    resources,
    lng: 'en',
    fallbackLng: 'en',
    debug: false,
    interpolation: {
      escapeValue: false,
    },
  });

export default i18n;
```

Усі переклади зберігаються у вигляді JSON-файлу в директорії locales. Наприклад файл з англійською локалізацією:

```
{
  "navbar": {
    "games": "GAMES",
    "allBoardgames": "All board games",
    "game2": "Game 2",
    "game3": "Game 3",
    "events": "EVENTS",
    "contact": "CONTACT",
    "profile": "Profile",
    "login": "Login"
  },
  "banner": {
    "welcomeTo": "Welcome to",
    "tabletop": "TABLETOP GAMES!",
    "findPerfect": "FIND THE PERFECT BOARD GAME AND",
    "description": "Gather the best players and create an
unforgettable game!",
    "getStarted": "GET STARTED",
    "words": {
      "players": "PLAYERS!",
      "fun": "FUN!",
      "friends": "FRIENDS!"
    }
  }
```

У React-компонентах переклад здійснюється за допомогою хука useTranslation. Приклад застосування useTranslation в компоненті банеру сайту (див. Додаток А розділ А.2).

Також, користувач має змогу змінити мову інтерфейсу за допомогою кнопок. Для зберігання стану та передавання в інші компоненти застосунку використовується LanguageContext.

```
import React, { createContext, useState, useEffect, useContext }
from 'react';
import { useTranslation } from 'react-i18next';

const LanguageContext = createContext();

export function LanguageProvider({ children }) {
  const { i18n } = useTranslation();
  const [language, setLanguage] = useState(() => {
    return localStorage.getItem('appLanguage') || 'en';
  });

  useEffect(() => {
    i18n.changeLanguage(language);
    localStorage.setItem('appLanguage', language);
  }, [language, i18n]);

  const changeLanguage = (lang) => {
    setLanguage(lang);
  };

  return (
    <LanguageContext.Provider value={{ language, changeLanguage }}>
      {children}
    </LanguageContext.Provider>
  );
}

export function useLanguage() {
  return useContext(LanguageContext);
}
```

Отже, підтримка багатомовності реалізована з використанням найкращих практик, що дозволяє зручно масштабувати застосунок на інші мови в майбутньому. Локалізація покращує доступність і зручність використання системи для ширшого кола користувачів.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування розробленого програмного забезпечення

Тестування програмного забезпечення є ключовим етапом розробки, який спрямований на підтвердження відповідності створеної системи функціональним вимогам, а також на забезпечення її стабільної та надійної роботи в різних експлуатаційних умовах.

Для комплексної оцінки якості розробленої рекомендаційної системи настільних ігор було використано три основні види тестування: юніт-тестування, інтеграційне тестування та мануальне (ручне) тестування.

Юніт-тестування полягає у перевірці окремих компонентів програмного забезпечення, а саме: модулів, функцій або класів з метою переконатися у їхній правильній роботі. Цей вид тестування дозволяє виявляти помилки на ранніх стадіях розробки, що значно знижує загальну кількість дефектів у системі.

Інтеграційне тестування призначене для оцінки взаємодії між різними компонентами системи, що працюють разом. Воно допомагає виявити помилки, які можуть виникати при обміні даними між модулями або при їхній спільній роботі. У проєкті було проведено інтеграційне тестування взаємодії фронтенд- і бекенд-компонентів, а також перевірку коректності обробки API-запитів і взаємодії із базою даних.

Мануальне тестування здійснюється шляхом виконання тестових сценаріїв з метою перевірки системи з позиції кінцевого користувача. Воно дозволяє оцінити зручність інтерфейсу, логіку навігації, а також стійкість системи до некоректних або нестандартних дій користувача. Завдяки мануальному тестуванню вдається виявити та усунути проблеми, пов’язані з юзабіліті, які складно виявити автоматичними методами.

5.2 Юніт-тестування

Юніт-тестування у проєкті проводилось для перевірки коректності роботи окремих функціональних модулів системи, таких як реєстрація, авторизація, пошук, фільтрація ігор та створення ігрових подій.

Для реалізації юніт-тестів було обрано фреймворк Jest, що є популярним інструментом для тестування JavaScript та React додатків. Він забезпечує зручний синтаксис, швидке виконання тестів та підтримку моків (імітацій зовнішніх залежностей).

Створимо тест-кейси для перевірки наступної функціональності системи:

- перемикання видимості пароля;
- перевірка коректного відображення повідомлення про помилку у випадку спроби виконати вхід при порожніх полях «email» або/та «password»;
- імітація успішного виклику API для аутентифікації.

Таблиця 5.1 – Тест кейс для перевірки перемикання видимості пароля

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №1		
Опис функції:	Перемикання видимості пароля		
Власник тесту:	Посукан Інна Ігорівна		
Дата створення:	27.05.2025		
Мета тесту:	Перевірити, що кнопка перемикає тип поля пароля між «password» і «text»		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Елемент з типом «password» присутній на сторінці	Поле для введення пароля відображається як тип «password»	Пройдено
Перемикання видимості пароля			
№	Опис випадку	Очікуваний результат	Висновок
1	Ініціалізація першого натискання кнопки-перемикача видимості пароля за допомогою fireEvent.click()	Тип поля змінюється з «password» на «text»	Пройдено

Продовження таблиці 5.1

2	Ініціалізація другого натискання кнопки-перемикача за допомогою fireEvent.click()	Тип поля повертається до «password»	Пройдено
Результати тестування			
Дата прогону тесту: 27.05.2025		Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Таблиця 5.2 – Тест кейс для перевірки заповнення полів

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №2		
Опис функції:	Перевірка обов’язковості полів для входу		
Власник тесту:	Посукан Інна Ігорівна		
Дата створення:	27.05.2025		
Мета тесту:	Переконатися, що при спробі виконати вхід з порожніми полями викликається функція валідації і повертається відповідна помилка		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Імітується натискання кнопки входу (без взаємодії користувача з UI).	Використання fireEvent.click() для імітаіції кліку на кнопку входу без заповнення полів.	Пройдено
Перевірка обов’язковості полів для входу			
№	Опис випадку	Очікуваний результат	Висновок
1	Виклик функції входу з порожніми полями (email = "", password = "")	Функція валідації виявляє пусті поля і повертає/виводить помилку з ключем "auth.errors.fillAllFields"	Пройдено
Результати тестування			
Дата прогону тесту: 27.05.2025		Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Таблиця 5.3 – Тест кейс для відправлення даних для аутентифікації на сервер

Інформація про тест-кейс			
Ідентифікатор тесту:	Тест-кейс №3		
Опис функції:	Перевірка обов’язковості полів для входу		
Власник тесту:	Посукан Інна Ігорівна		
Дата створення:	27.05.2025		
Мета тесту:	Переконатися, що при заповнених полях відбувається виклик функції authenticate з коректними параметрами.		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Поле «password» та «email» заповнені	Поле «password» та «email» заповнені	Пройдено
Перевірки відправлення даних для аутентифікації на сервер			
№	Опис випадку	Очікуваний результат	Висновок
1	Ініціалізація натискання кнопки Login за допомогою fireEvent.click()	Дані відправляються на сервер, статус 200 OK. Виводиться повідомлення про успішну аутентифікацію.	Пройдено
Результати тестування			
Дата прогону тесту: 27.05.2025		Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Юніт-тест для виконання тест-кейсів пов’язаних з аутентифікацією (див. Додаток В розділ В.1).

5.3 Інтеграційне тестування

Інтеграційне тестування проводиться з метою перевірки взаємодії між окремими модулями програмної системи. Проведемо інтеграційне тестування компонента “EventCalendar” за допомогою Jest.

Таблиця 5.4 – Тест кейс для відправлення даних для перевірки навігації

Інформація про тест-кейс	
Ідентифікатор тесту:	Тест-кейс №4
Опис функції:	Перевірка навігації при натисканні кнопки "Додати подію"
Власник тесту:	Посукан Інна Ігорівна

Продовження таблиці 5.4

Дата створення:	27.05.2025		
Мета тесту:	Перевірити, що при натисканні кнопки додається нова подія та відбувається навігація на сторінку створення події		
Передумова			
№	Опис випадку	Очікуваний результат	Висновок
1	Компонент EventCalendar імпортовано	Компонент EventCalendar імпортовано	Пройдено
2	Змоканий useTraslation та useNavigate	Змоканий useTraslation та useNavigate	Пройдено
Перевірка навігації при натисканні кнопки "Додати подію"			
№	Опис випадку	Очікуваний результат	Висновок
1	Відрендерити компонент “EventCalendar” у MemoryRouter	Компонент відрендерений у MemoryRouter	Пройдено
2	Натиснути на кнопку “Add Event”	Функція navigate викликається з аргументом /addevent	Пройдено
Результати тестування			
Дата прогону тесту: 27.05.2025		Результат тесту (P/F/B): ПРОЙДЕНО (P)	

Інтеграційне тестування перевірки функції навігації (див. Додаток В розділ В.2).

ВИСНОВКИ

У ході виконання звіту з передатестаційної практики було створено програмну систему для персоналізації досвіду гравців у настільні ігри. В межах даного проєкту було проведено повний цикл розробки програмного забезпечення: від аналізу предметної області та формування вимог до проєктування, реалізації, тестування та документування системи.

Головною метою роботи було спрощення процесу організації заходів, пов’язаних із настільними іграми, та створення зручного інструменту для комунікації між гравцями, організаторами й клубами. У результаті було розроблено систему «Tabletop», яка дозволяє користувачам створювати події, реєструватися на них, переглядати деталі заходів, залишати зворотний зв’язок і отримувати персоналізовані рекомендації.

Програмна система реалізована з використанням сучасного стеку технологій: React.js, ASP.NET Core, MSSQL, REST API, OAuth2, SCSS тощо. Для клієнтської частини застосовано патерн Container–Presenter, що забезпечує зручне розділення логіки та візуального представлення, а також сприяє масштабованості й підтримуваності коду. Серверна частина побудована на основі Onion-архітектури в поєднанні з підходом Domain-Driven Design, що гарантує чітку структуру проєкту, гнучкість та можливість розширення.

Також, було реалізовано низку важливих функцій: авторизація через Google OAuth, створення/редагування подій, система фільтрації та пошуку , збереження ігор у “обране”. Значну увагу було приділено адаптивному дизайну інтерфейсу, який створено у Figma як для десктопних, так і для мобільних пристроїв.

У перспективі система може бути доповнена мобільним застосунком, функціоналом онлайн-ігор та монетизаційною моделлю з платіжною системою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мартін Р. Чистий код: створення, аналіз та архітектура – Харків: Фабула, 2019 – 200 с.
2. React JS Documentation [Електронний ресурс]. – URL: https://legacy.reactjs.org/docs/getting-started.html (дата звернення: 28.03.2025).
3. .NET Documentation [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 18.04.2025).
4. Jill Butler. Principles of Design – Quarto Publishing, 2023- 324 p.
5. Figma [Електронний ресурс]. – URL: https://help.figma.com/hc/en-us (дата звернення: 05.04.2025).
6. Google Identity Platform Documentation (OAuth 2.0) [Електронний ресурс]. – URL: https://developers.google.com/identity (дата звернення 18.04.2025).
7. JWT.IO Documentation [Електронний ресурс]. – URL: https://jwt.io/introduction/ (дата звернення 13.04.2025).
8. React Bootstrap Documentation [Електронний ресурс]. – URL: https://react-bootstrap.github.io (дата звернення 05.04.2025).
9. Vite Documentation [Електронний ресурс]. – URL: https://vitejs.dev/guide/ (дата звернення 01.04.2025).
10. React Context API Documentation [Електронний ресурс] – URL: https://reactjs.org/docs/context.html/ (дата звернення: 11.04.2025).
11. Axios Documentation [Електронний ресурс]. – URL: https://github.com/axios/axios (дата звернення: 03.06.2025).