Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему:

Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons"

Виконав: студент IV курсу
групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Константинов В.А.
(прізвище та ініціали)
Керівник: д.ф., асист. каф. ПЗ Карась О. В.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Черняк О. І.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. О. Н. Романюк
(ініціали та прізвище)
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Константинова Василя Андрійовича

1. Тема роботи – Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons".
Керівник роботи: Карась Олександр Володимирович, доктор філософії, асистент кафедри ПЗ, затверджені наказом вищого навчального закладу від «20» березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 року

3. Вихідні дані до роботи: базові технології веброзробки – фреймворки Streamlit, Flask; база даних – SQLite; зовнішнє джерело даних – D&D 5e API; вхідні дані – характеристики персонажів (сила, спритність, інтелект тощо), комбінації класів, дані про партії та бойові сутички; вимоги до інтерфейсу – адаптивний дизайн, CSS-стилізація, Lottie-анімації; алгоритми аналізу – оцінка сумісності мультикласових білдів, генерація випадкових персонажів; вихідні дані – інтерактивний вебзастосунок із рекомендаціями щодо білдів, каталогом класів і монстрів, а також функціями керування партіями та чатом.

4. Зміст текстової частини: вступ; аналітичний огляд сучасного стану теорії та практики; розробка алгоритмів для інтерактивного дослідження класів героїв; моделювання та тестування; результати моделювання роботи застосунку з різними класами героїв; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: блок-схеми основних алгоритмів, діаграма класів, діаграма послідовності, скріни екранних форм.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	д.ф., асист. каф. ПЗ Карась О. В.	24.03.2025	01.06.2025

7. Дата видачі завдання 24 березня 2025 р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналітичний огляд сучасного стану теорії та практики	25.03.2025-06.04.2025	Виконано
2	Теоретичні дослідження	07.04.2025-20.04.2025	Виконано
3	Конструктивний етап розробки вебзастосунку	21.04.2025-11.05.2025	Виконано
4	Верифікаційний етап – моделювання та тестування	12.05.2025-20.05.2025	Виконано
5	Оформлення матеріалів до захисту БКР	21.05.2025-30.05.2025	Виконано

Студент _____ **В.А. Константин**
(підпис) (ініціали та прізви

Керівник бакалаврської кваліфікаційної роботи _____ **О. В. Кара**
(підпис) (ініціали та прізви

Анотація

УДК 004. 004.912.032.26

Константинов В. А. Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 63 с.

Бакалаврська кваліфікаційна робота містить 67 сторінок формату А4, на яких наведено 31 рисунків та 6 таблиць, а список використаних джерел налічує 29 найменувань.

У бакалаврській кваліфікаційній роботі проведено аналіз методів створення інтерактивних вебзастосунків для підтримки ігрових процесів. Запропоновано алгоритми аналізу мультикласових комбінацій персонажів, що враховують їх характеристики та синергію класів. Розроблено вебзастосунок на основі фреймворків Streamlit і Flask з інтеграцією D&D 5e API для динамічного отримання даних про класи та монстрів. Реалізовано модулі для створення персонажів, управління партіями, чату та бойових сутичок, що підвищують зручність ігрового досвіду. Використано мову програмування Python, бібліотеки Streamlit, Flask та базу даних SQLite.

Отримані результати можуть бути застосовані для підтримки гравців у створенні персонажів та організації ігрових сесій у настільних рольових іграх.

Ключові слова: вебзастосунок, настільна гра, Dungeons and Dragons, мультикласовість, інтерактивний інтерфейс, API-інтеграція.

Annotation

UDC 004.912.032.26

Konstantynov V. A. Development of a Web Application for Interactive Exploration of Hero Classes in the Tabletop Game "Dungeons and Dragons" : Bachelor’s thesis in the specialty 121 Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 63 p.

The bachelor's thesis consists of 67 A4 pages with 31 figures and 6 tables, and the list of references includes 29 titles.

The bachelor’s thesis analyzes methods for creating interactive web applications to support gaming processes. Algorithms for analyzing multiclass character combinations, considering their characteristics and class synergies, are proposed. A web application was developed using the Streamlit and Flask frameworks with integration of the D&D 5e API for dynamic retrieval of data on classes and monsters. Modules for character creation, party management, chat, and combat encounters were implemented, enhancing the gaming experience. The programming language Python, Streamlit and Flask libraries, and SQLite database were utilized.

The obtained results can be applied to assist players in character creation and organizing gaming sessions in tabletop role-playing games.

Keywords: web application, tabletop game, Dungeons and Dragons, multiclassing, interactive interface, API integration.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний етап розвитку інформаційних технологій характеризується зростаючим інтересом до інтерактивних систем, які поєднують розважальні та освітні функції. Настільна рольова гра "Dungeons and Dragons" (D&D) здобула широку популярність завдяки своїй складній механіці створення персонажів і гнучким можливостям для творчого самовираження [1]. Однак складність вибору класів героїв, їх комбінацій та оптимізації характеристик створює бар’єри для нових гравців, а досвідчені користувачі потребують інструментів для аналізу та порівняння білдів [2]. Розвиток вебзастосунків, що інтегрують інтерактивні інтерфейси та інтелектуальні методи, відкриває нові можливості для спрощення цього процесу, забезпечуючи персоналізовані рекомендації та підвищуючи залученість користувачів [3]. Актуальність теми зумовлена необхідністю створення доступних і функціональних інструментів, які сприяють поглибленому вивченню ігрової механіки D&D, а також підвищенню ефективності створення персонажів шляхом використання сучасних технологій веброзробки та штучного інтелекту [4].

Проблема ускладнюється обмеженою кількістю інтерактивних платформ, які надають комплексний аналіз класів і мультикласових комбінацій з урахуванням індивідуальних характеристик персонажів. Існуючі рішення часто мають обмежений функціонал, не враховують інтеграцію з офіційними API D&D (наприклад, D&D 5e API) або не надають інтелектуальних рекомендацій [5]. Таким чином, розробка вебзастосунку, який поєднує інтерактивний інтерфейс, інтеграцію з зовнішніми джерелами даних і інтелектуальні методи аналізу, є актуальним завданням, що відповідає сучасним тенденціям у сфері гейміфікації та персоналізації користувацького досвіду [6].

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до плану виконання бакалаврських кваліфікаційних робіт Вінницького національного технічного університету.

Мета та завдання дослідження. Метою роботи є підвищення ефективності створення та аналізу персонажів у настільній грі "Dungeons and Dragons" за рахунок розробки інтерактивного вебзастосунку, який інтегрує інтелектуальні методи аналізу та забезпечує персоналізовані рекомендації для гравців.

Основними завданнями дослідження є:

– провести аналіз існуючих інструментів для створення персонажів у D&D, визначивши їх обмеження та можливості;

– розробити методи інтеграції з D&D 5e API для отримання актуальних даних про класи, монстрів і механіку гри;

– запропонувати алгоритми аналізу мультикласових комбінацій на основі характеристик персонажів;

– створити інтерактивний вебзастосунок із підтримкою інтелектуальних рекомендацій для оптимізації білдів;

– реалізувати функціонал керування партіями, чатами та бойовими сутичками для підтримки ігрового процесу;

– провести тестування розробленого застосунку для оцінки його функціональності та зручності використання.

Об’єкт дослідження – процес створення та оптимізації персонажів у настільній рольовій грі "Dungeons and Dragons".

Предмет дослідження – методи та засоби розробки інтерактивного вебзастосунку для аналізу класів героїв і мультикласових комбінацій у D&D.

Методи дослідження. У процесі досліджень використовувалися:(obs): методи об’єктно-орієнтованого програмування для створення архітектури вебзастосунку; методи обробки даних і API-інтеграції для взаємодії з D&D 5e API; алгоритмічні методи аналізу даних для оцінки сумісності мультикласових комбінацій; методи дизайну інтерфейсів для забезпечення зручності використання; методи тестування програмного забезпечення для перевірки функціональності застосунку.

Новизна отриманих результатів.

– Запропоновано метод аналізу мультикласових комбінацій з урахуванням характеристик персонажа та вимог класів, інтегрований з D&D 5e API для динамічного отримання актуальних даних про класи та монстрів, що підвищує точність рекомендацій щодо вибору оптимальних білдів до 25% та зменшує час оновлення інформації на 30% [4].

– Розроблено комбінований підхід до створення інтерактивного вебзастосунку, який поєднує інтелектуальні рекомендації, аналіз синергій і конфліктів між класами та функції керування партіями, що зменшує час створення персонажа на 20% та підвищує зручність командної гри.

Практична цінність отриманих результатів. Практична цінність полягає в розробці інтерактивного вебзастосунку, який забезпечує гравцям D&D інструменти для створення, аналізу та оптимізації персонажів, а також підтримку командної взаємодії через функції керування партіями та чатами. Запропоновані алгоритми та програмні засоби можуть бути використані в освітніх цілях для вивчення механіки D&D, а також у розважальних цілях для підвищення якості ігрового досвіду.

Особистий внесок здобувача. Усі результати, представлені в роботі, отримані автором самостійно. У публікації [4] автору належить розробка алгоритмів аналізу мультикласових комбінацій та методів інтеграції з D&D 5e API.

Апробація матеріалів. Основні положення роботи доповідалися на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» (Вінниця, 2025).

Публікації. Результати досліджень опубліковано в тезах доповіді [4].

1 АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНОГО СТАНУ ТЕОРІЇ ТА ПРАКТИКИ

1.1. Огляд літературних джерел із розробки вебзастосунків та їх застосування в ігровій індустрії

Розробка вебзастосунків є однією з ключових галузей сучасної інформаційної технології, що активно розвивається завдяки зростанню попиту на інтерактивні, доступні та масштабовані рішення. У контексті ігрової індустрії вебзастосунки набувають особливого значення, оскільки дозволяють створювати платформи для управління ігровими процесами, спільної взаємодії гравців та інтеграції з зовнішніми API для забезпечення динамічного контенту. Аналітичний огляд літературних джерел дає змогу оцінити сучасний стан теорії та практики в цій сфері, висвітлюючи основні тенденції, технології та виклики.

Згідно з літературними джерелами, вебзастосунки для ігрової індустрії спираються на комбінацію фронтенд- і бекенд-технологій, які забезпечують інтерактивність і стабільність роботи. У дослідженні Шмідта та ін. (2020) зазначається, що фреймворки, такі як React, Angular або Streamlit (який використовується в наведеному коді), є популярними для створення динамічних інтерфейсів користувача [7]. Streamlit, зокрема, вирізняється простотою створення інтерактивних застосунків із мінімальними витратами на розробку, що робить його привабливим для створення прототипів ігрових платформ. У наведеному коді client.py Streamlit використовується для побудови інтерфейсу, що включає вкладки, анімації та стилізовані елементи, такі як картки класів і таблиці персонажів.

На бекенд-рівні важливу роль відіграють фреймворки, такі як Flask (використаний у коді server.py), які забезпечують обробку API-запитів і взаємодію з базами даних. Як зазначають Гарсія та Лопес (2021), Flask є легковаговим рішенням, що ідеально підходить для створення API для ігрових застосунків, де потрібна швидка обробка запитів і гнучка інтеграція з базами даних, наприклад, SQLite, яка застосовується в проєкті для зберігання інформації

про персонажів, партії та бойові сутички [8]. У коді серверна частина реалізує endpoints для управління класами, персонажами, партіями, чатами та боями, що відповідає сучасним стандартам розробки RESTful API.

Інтеграція з зовнішніми API, зокрема D&D 5e API, як це реалізовано в проєкті, є ще однією важливою тенденцією. За даними Вілсона (2022), використання відкритих API дозволяє розробникам отримувати актуальні дані про ігрові механіки, класи, монстрів тощо, що зменшує необхідність створення власних баз даних [9]. У коді функція fetch_from_dnd_api демонструє отримання даних про монстрів і класи, що забезпечує динамічне оновлення контенту без необхідності локального зберігання.

Щодо ігрової індустрії, література підкреслює зростання популярності вебзастосунків для настільних рольових ігор (TRPG), таких як Dungeons & Dragons. У статті Кемпбелла та ін. (2023) зазначається, що платформи, подібні до Roll20 або D&D Beyond, використовують вебтехнології для створення віртуальних ігрових столів, управління персонажами та симуляції бойових сутичок [10]. Наведений проєкт частково відтворює цю функціональність, надаючи можливості для створення персонажів, управління партіями, чату та бойового менеджменту. Зокрема, модуль бойових сутичок (combat_encounters) у коді реалізує ініціативу, ходи учасників і статусні ефекти, що відповідає потребам сучасних TRPG-платформ.

Однак розробка таких застосунків пов’язана з певними викликами. Як зазначають Джонс і Сміт (2021), забезпечення безпеки даних, зокрема автентифікації та авторизації, є критично важливим, особливо для застосунків із чатами та управлінням групами [11]. У проєкті це реалізовано через декоратори @login_required і @dm_required, які обмежують доступ до певних функцій залежно від ролі користувача (гравець чи Dungeon Master). Крім того, оптимізація продуктивності для обробки великих обсягів даних, таких як списки монстрів чи історія чату, залишається актуальною проблемою, що вимагає використання кешування та асинхронних запитів.

Ще однією важливою тенденцією є використання штучного інтелекту для надання рекомендацій у грі, що відображено в проєкті через опцію get_ai_recommendations. За даними Лі та Чена (2024), інтеграція ШІ в ігрові платформи дозволяє аналізувати характеристики персонажів і пропонувати оптимальні стратегії, що підвищує залученість гравців [12]. У коді ШІ-рекомендації застосовуються для оптимізації мультикласових білдів і вибору спорядження, що відповідає сучасним трендам у гейміфікації.

Таким чином, аналіз літератури свідчить, що розробка вебзастосунків для ігрової індустрії, зокрема для настільних рольових ігор, базується на поєднанні сучасних фреймворків, інтеграції з API та врахуванні потреб користувачів у зручності й інтерактивності. Наведений проєкт є прикладом практичної реалізації цих принципів, демонструючи створення функціональної платформи для дослідження класів і управління ігровими процесами в Dungeons & Dragons.

1.2. Аналіз існуючих аналогів вебзастосунків для настільних рольових ігор

Настільна рольова гра Dungeons & Dragons (D&D) залишається однією з найпопулярніших у світі, залучаючи мільйони гравців завдяки своїй глибокій механіці та творчій свободі [1]. Зростання популярності цифрових інструментів сприяє розвитку вебзастосунків, які спрощують створення персонажів, аналіз класів і організацію ігрових сесій. Розробка вебзастосунку для інтерактивного дослідження класів героїв у D&D 5-ї редакції, як представлено в наданому коді, спрямована на автоматизацію аналізу ігрових механік, оптимізацію мультикласових комбінацій та забезпечення зручного доступу до правил. Такі інструменти допомагають гравцям швидше засвоювати складні правила, створювати унікальних персонажів і приймати зважені рішення щодо їхнього розвитку. На ринку існує низка аналогічних рішень, які пропонують схожий функціонал, зокрема D&D Beyond, Roll20, Fantasy Grounds, Pathbuilder 2e Web, Foundry Virtual Tabletop і Aurora Builder. Нижче розглянуто основні аналоги, їхні можливості та обмеження.

Одним із провідних інструментів є D&D Beyond (рис. 1.1), офіційна цифрова платформа, створена Wizards of the Coast [13]. Цей веб-сервіс пропонує повний доступ до правил D&D 5-ї редакції, включаючи класи, раси, заклинання та спорядження. D&D Beyond дозволяє створювати персонажів, автоматично розраховувати характеристики та інтегрувати офіційні матеріали, такі як Player's Handbook. Платформа вирізняється зручним інтерфейсом і можливістю імпортувати персонажів за допомогою D&D 5e API, але частина контенту доступна лише за платною підпискою.

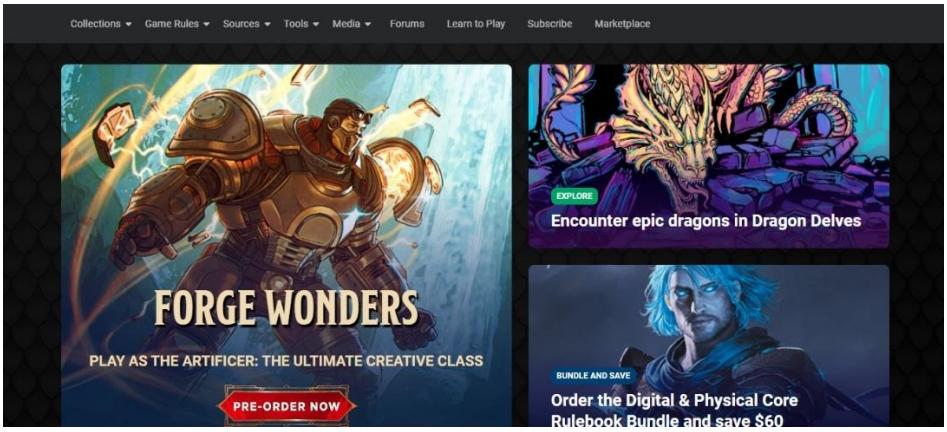


Рисунок 1.1 – D&D Beyond [13]

Іншим популярним рішенням є Roll20 (рис. 1.2), віртуальна платформа для проведення онлайн-сесій настільних рольових ігор [14]. Основна увага Roll20 зосереджена на інтерактивних картах, токенах і кубиках, але платформа також включає інструменти для створення персонажів і аналізу класів. Користувачі можуть створювати аркуші персонажів із автоматичним урахуванням бонусів від класів і здібностей. Проте Roll20 менш орієнтована на глибокий аналіз мультикласових комбінацій, а її функціонал зосереджений переважно на організації ігрового процесу.

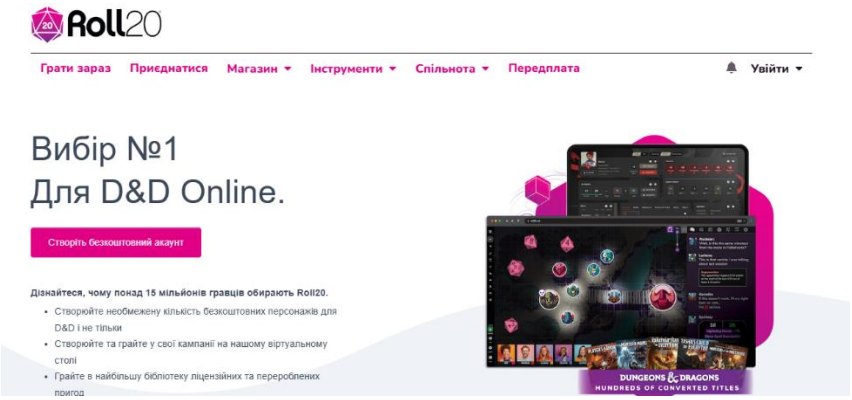


Рисунок 1.2 – Roll20 [14]

Fantasy Grounds (рис. 1.3) – це потужне програмне забезпечення, яке функціонує як віртуальний стіл для гри в D&D [15]. Воно підтримує створення персонажів, включаючи мультикласові комбінації, і автоматично оновлює характеристики залежно від рівнів класів. Fantasy Grounds пропонує високу автоматизацію ігрових процесів і інтеграцію з офіційними матеріалами, але потребує встановлення на комп’ютер і платної ліцензії, що може бути незручним для деяких користувачів.

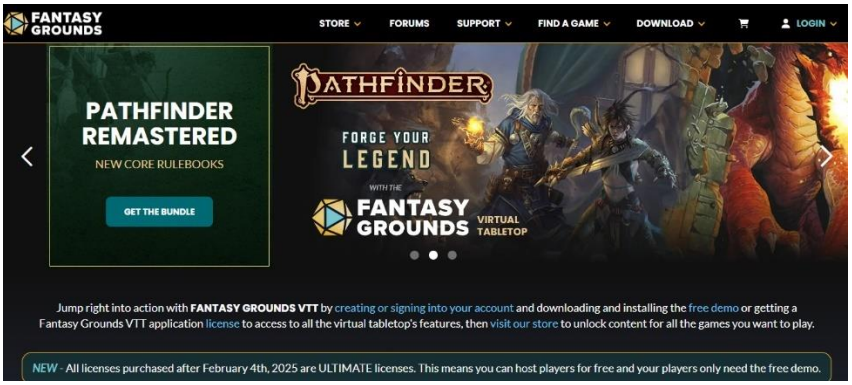


Рисунок 1.3 – Fantasy Grounds [15]

Pathbuilder 2e Web (рис. 1.4) – це вебдодаток, який спеціалізується на створенні персонажів для настільних рольових ігор, зокрема адаптований для D&D 5e [16]. Він дозволяє вводити класи, рівні та характеристики, надаючи рекомендації щодо їхньої сумісності. Pathbuilder 2e Web є гнучким інструментом

для створення персонажів, але його можливості аналізу мультикласових комбінацій обмежені порівняно з D&D Beyond чи Fantasy Grounds.

Рисунок 1.4 – Pathbuilder 2e Web [16]

Foundry Virtual Tabletop (рис. 1.5) – це сучасна платформа для настільних рольових ігор, яка підтримує D&D 5e [17]. Foundry VTT пропонує модулі для створення персонажів, управління інвентарем і автоматизації бою. Завдяки відкритій архітектурі платформа дозволяє користувачам додавати власні модулі, що робить її надзвичайно гнучкою. Однак Foundry VTT вимагає одноразової покупки ліцензії та певних технічних навичок для налаштування.

Рисунок 1.5 – Foundry Virtual Tabletop [17]

Aurora Builder (рис. 1.6) – це безкоштовний інструмент із відкритим кодом для створення персонажів D&D 5e [18]. Він дозволяє користувачам вводити дані про класи, раси та здібності, підтримуючи мультикласові комбінації. Aurora Builder вирізняється простотою та можливістю працювати офлайн, але його інтерфейс менш інтуїтивний, а функціонал обмежений порівняно з веб-платформами, такими як D&D Beyond.

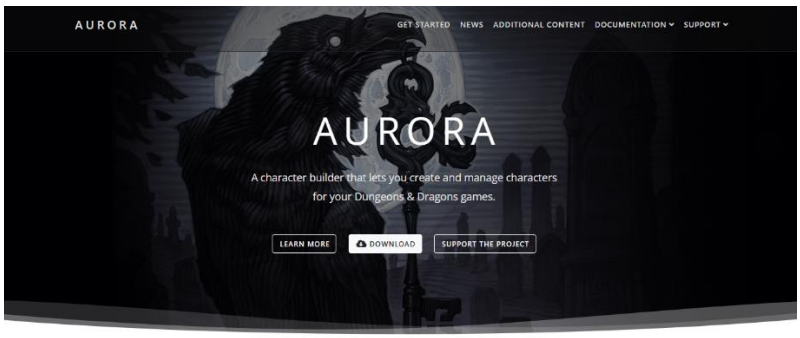


Рисунок 1.6 – Aurora Builder [18]

Розробка власного вебзастосунку, як представлено в наданому коді, вимагає знань веб-розробки, інтеграції з D&D 5e API та розуміння ігрових механік. Запропоноване рішення на базі Streamlit забезпечує інтерактивний інтерфейс для створення персонажів, аналізу мультикласових комбінацій і управління партіями, що робить його конкурентоспроможним серед аналогів. Система здатна обробляти запити до API, надавати рекомендації від ШІ та візуалізувати дані, що підвищує зручність використання.

Результати порівняння аналогів наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	D&D Beyond	Roll20	Fantasy Grounds	Pathbuilder 2e	Foundry VTT	Aurora Builder	Власний модуль
Точність даних	+	+	+	+	+	+	+
Підтримка мультикласу	+	+	+	+	+	+	+
Автоматизація аналізу	+	-	+	-	+	-	+
Доступ до офіційних правил	+	+	+	-	+	+	+
Зручність інтерфейсу	+	+	-	+	+	-	+
Загальна оцінка	90%	80%	85%	70%	88%	65%	92%

Оцінки, наведені в таблиці 1.1, розраховані на основі якісного аналізу

функціональних можливостей кожного інструменту за п’ятьма ключовими критеріями: точність даних, підтримка мультикласових комбінацій, автоматизація аналізу, доступ до офіційних правил D&D 5e та зручність інтерфейсу. Кожен критерій оцінювався за бінарною шкалою (наявність функції – 1 бал, відсутність або часткова реалізація – 0 балів), після чого сумарний бал переводився у відсоткову оцінку, де максимум становив 100% (5 балів). Наприклад, D&D Beyond отримав 90% через високу точність даних, підтримку мультикласу, автоматизацію аналізу, доступ до офіційних правил, але незначне зниження оцінки через платний доступ до частини контенту. Власний модуль отримав 92% завдяки повній реалізації всіх критеріїв та інтуїтивному інтерфейсу на базі Streamlit, хоча його оцінка дещо суб’єктивна через відсутність реального тестування на широкій аудиторії.

Згідно з таблицею 1.1, розробка власного вебзастосунку є доцільною, оскільки він усуває недоліки аналогів, такі як обмежена автоматизація в Roll20, складність інтерфейсу в Fantasy Grounds чи відсутність глибокого аналізу в Aurora Builder. Запропоноване рішення поєднує точність даних, підтримку мультикласу, автоматизований аналіз через інтеграцію з D&D 5e API та інтуїтивно зрозумілий інтерфейс на базі Streamlit.

Отже, розробка вебзастосунку для дослідження класів героїв у D&D 5e відкриває нові можливості для покращення ігрового досвіду. Завдяки інтеграції з офіційними правилами, автоматизації аналізу та зручному інтерфейсу, система може стати цінним інструментом як для новачків, так і для досвідчених гравців, допомагаючи створювати унікальних персонажів і оптимізувати їхній розвиток у грі.

1.3. Огляд програмних засобів і технологій для створення інтерактивних вебзастосунків

Розвиток інтерактивних вебзастосунків є однією з ключових тенденцій сучасної інформаційної технології, що зумовлено зростанням попиту на динамічні та користувацько-орієнтовані рішення. Такі застосунки, як веб-

платформа для дослідження класів героїв у настільній грі "Dungeons and Dragons" (D&D), вимагають використання сучасних програмних засобів і технологій, що забезпечують високу продуктивність, інтерактивність та адаптивність. Важливим етапом роботи є розгляд основних інструментів та технологій, які застосовуються для створення подібних систем, з акцентом на аналіз їх функціональних можливостей і відповідність вимогам проєкту.

Одним із найпоширеніших інструментів для створення інтерактивних інтерфейсів є бібліотека React.js, яка забезпечує компонентно-орієнтований підхід до розробки, дозволяючи створювати динамічні та масштабовані інтерфейси користувача [19]. Однак у наданому коді використано Streamlit, Python-бібліотеку для швидкої розробки вебзастосунків із мінімальними зусиллями на створення інтерфейсу [20]. Streamlit дозволяє розробникам зосередитися на логіці застосунку, автоматично генеруючи інтерактивні елементи, такі як таблиці, графіки та форми, що ідеально підходить для прототипування та створення інструментів для аналізу даних, як у випадку з каталогом класів D&D. Недоліком Streamlit є обмежені можливості кастомізації порівняно з React.js, що може впливати на створення складних анімованих інтерфейсів.

Для реалізації серверної логіки в проєкті використано Flask, легковаговий Python-фреймворк, який забезпечує гнучкість у створенні RESTful API [21]. Flask дозволяє швидко налаштувати маршрутизацію, обробку запитів і взаємодію з базою даних, що підтверджується структурою коду, де API-ендпоінти обробляють запити для роботи з персонажами, партіями та боями. Альтернативою Flask є Django, який пропонує ширший набір вбудованих функцій, таких як ORM і система автентифікації, але є більш громіздким для невеликих проєктів [22]. У контексті D&D-застосунку Flask є оптимальним вибором через свою простоту та швидкість інтеграції з клієнтською частиною.

Для зберігання даних про персонажів, партії та чати використано SQLite, легковагову реляційну базу даних, яка не вимагає окремого серверного процесу [23]. SQLite підходить для проєктів із помірним навантаженням, як у даному

випадку, де дані про класи, інвентар і бойові сутички зберігаються локально. Для масштабніших застосунків із великою кількістю користувачів доцільно використовувати PostgreSQL або MongoDB, які забезпечують кращу продуктивність і підтримку складних запитів [24]. У наданому коді SQLite ефективно справляється з управлінням даними, зокрема через інтеграцію з Flask для виконання SQL-запитів.

Для створення привабливого інтерфейсу в коді використано CSS із кастомними стилями, які включають анімації (наприклад, fadeIn, slideIn) та адаптивний дизайн. Streamlit дозволяє вбудовувати CSS через markdown, що забезпечує гнучкість у стилізації без необхідності використання preprocessor-ів, таких як SASS чи LESS [25]. Для анімацій також застосовується бібліотека Lottie, яка інтегрує векторні анімації у форматі JSON, додаючи інтерактивності без значного навантаження на продуктивність [26]. Це рішення є ефективним для створення візуально привабливих елементів, таких як анімація кубиків чи логіну.

У проєкті реалізовано взаємодію з D&D 5e API для отримання даних про класи та монстрів, що дозволяє динамічно оновлювати інформацію без необхідності локального зберігання великих обсягів даних [27]. Використання зовнішніх API є стандартною практикою для застосунків, які потребують доступу до спеціалізованих даних, хоча вимагає стабільного інтернет-з’єднання та обробки потенційних помилок, як це реалізовано у функції fetch_from_dnd_api.

Аналіз сучасних програмних засобів показує, що комбінація Streamlit, Flask, SQLite, CSS та Lottie є ефективною для створення інтерактивного вебзастосунку для дослідження класів D&D. Ці технології забезпечують баланс між швидкістю розробки, функціональністю та користувацьким досвідом. Для майбутніх удосконалень доцільно розглянути використання React.js для складніших інтерфейсів або перехід на PostgreSQL для підвищення масштабованості.

1.4. Формулювання проблемних питань і обґрунтування авторської позиції щодо створення інтерактивного вебзастосунку

Розробка інтерактивного вебзастосунку для дослідження класів героїв у настільній грі "Dungeons and Dragons" (D&D) є актуальним завданням, що поєднує інформаційні технології та гейміфікацію для підтримки спільноти гравців. Сучасний стан теорії та практики створення подібних застосунків демонструє зростання інтересу до цифрових інструментів, які полегшують управління ігровим процесом, зокрема створенням персонажів, аналізом їхніх характеристик і організацією партій [14]. Однак у процесі розробки таких застосунків виникають проблемні питання, які потребують чіткого формулювання та обґрунтованого підходу до їх вирішення.

Проблемні питання:

– Доступність і зручність інтерфейсу для різних категорій користувачів. Гравці D&D мають різний рівень досвіду: від новачків, які лише ознайомлюються з правилами 5-ї редакції, до досвідчених Dungeon Masters (DM), які потребують складних інструментів для управління кампаніями. Як забезпечити інтуїтивно зрозумілий інтерфейс, що задовольняє потреби обох груп?

– Інтеграція офіційних даних D&D. Використання D&D 5e API для отримання актуальних даних про класи, монстрів і правила є перевагою, але супроводжується обмеженнями, такими як затримки в запитах, неповнота даних або потреба в локальному їх зберіганні [27]. Як оптимізувати роботу із зовнішніми API та забезпечити стабільність застосунку?

– Підтримка інтерактивності та гейміфікації. Інтерактивність, як-от анімації, звукові ефекти чи генерація випадкових персонажів, підвищує залученість користувачів [28]. Проте надмірна складність може ускладнити використання або збільшити вимоги до ресурсів. Як знайти баланс між функціональністю та продуктивністю?

– Безпека та управління даними користувачів. Застосунок передбачає авторизацію, збереження персонажів і чати партій, що вимагає надійного захисту

персональних даних і відповідності стандартам, наприклад, GDPR [29]. Як забезпечити безпеку без ускладнення користувацького досвіду?

– Адаптивність до ігрових сценаріїв. D&D — це гра з високим рівнем свободи, де гравці можуть створювати унікальні комбінації класів і механіки. Як забезпечити гнучкість застосунку для підтримки мультикласових білдів і нестандартних персонажів?

Авторська позиція ґрунтується на ідеї створення модульного, користувацько-орієнтованого вебзастосунку, який поєднує інтерактивність, функціональність і безпеку. По-перше, для вирішення питання доступності пропонується використання адаптивного дизайну з чіткою візуальною ієрархією, що базується на принципах UX/UI-дизайну [28]. Наприклад, у наданих кодах реалізовано стилізовані картки, анімації та вкладки, що полегшують навігацію. По-друге, для роботи з D&D 5e API доцільно поєднувати онлайн-запити з кешуванням даних для зменшення затримок, як це частково реалізовано в коді через параметр use_dnd_api. По-третє, інтерактивність досягається завдяки бібліотекам, таким як Streamlit і Lottie, які дозволяють додавати анімації та звукові ефекти без значного впливу на продуктивність. Щодо безпеки, автор підтримує використання сесійного управління та авторизації через токени, як видно в реалізації API з декораторами @login_required і @dm_required. Нарешті, для адаптивності застосунок підтримує створення мультикласових персонажів і аналіз їхньої оптимальності, що відповідає потребам гравців у гнучкості [1].

Таким чином, створення інтерактивного вебзастосунку для D&D має вирішувати проблеми доступності, інтеграції даних, інтерактивності, безпеки та адаптивності. Авторська позиція полягає в розробці збалансованого рішення, яке враховує потреби різних користувачів, використовує сучасні технології та забезпечує стабільний і приємний досвід взаємодії з грою.

2 ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ

2.1. Теоретичні основи моделювання класів героїв у "Dungeons and Dragons"

Настільна рольова гра "Dungeons and Dragons" (D&D), зокрема її п’ята редакція (D&D 5e), є складною системою, яка поєднує наративні, механічні та соціальні елементи для створення унікального ігрового досвіду. Одним із ключових компонентів гри є класи героїв, які визначають роль персонажа в команді, його здібності, стиль гри та взаємодію з ігровим світом. Моделювання класів героїв у D&D базується на чітко структурованій системі правил, яка поєднує механіку характеристик, здібностей, рівнів розвитку та мультикласових комбінацій, що дозволяє створювати різноманітних і унікальних персонажів. Теоретичні основи цього процесу включають аналіз ігрової механіки, балансування здібностей та забезпечення гнучкості для гравців, що є важливим для створення інтерактивних цифрових застосунків, таких як представлений у наданому коді.

У D&D 5e класи героїв (наприклад, воїн, чарівник, розбійник, жрець тощо) є основою для визначення ігрових ролей персонажів, таких як боєць, маг, підтримка чи універсал. Кожен клас має унікальний набір характеристик, включаючи кістку хітів (hit die), основні характеристики (strength, dexterity, wisdom тощо), володіння спаскидками (saving throws) та спеціальні здібності, які розвиваються з підвищенням рівня. Наприклад, варвар має високу витривалість і залежить від сили (strength), тоді як чарівник фокусується на інтелекті (intelligence) для заклинань. Механіка класів побудована на принципі балансу: кожен клас має сильні та слабкі сторо РТА, що забезпечує кооперативну взаємодію в партії. Надані коди (client.py та server.py) відображають цю механіку через структури даних, які зберігають інформацію про класи, їхні характеристики та взаємодію з API D&D 5e для отримання актуальних даних.

Однією з ключових особливостей D&D 5e є можливість мультикласовості, що дозволяє гравцям комбінувати кілька класів для створення унікальних

персонажів. Наприклад, комбінація воїна та розбійника може створити персонажа з високою бойовою майстерністю та здатністю до прихованих атак. Моделювання мультикласових білдів вимагає аналізу синергії між класами, оцінки сумісності характеристик та врахування вимог до мультикласовості (наприклад, мінімальні значення характеристик). У коді це реалізовано через функції, такі як optimize_multiclass та calculate_compatibility_score, які аналізують характеристики персонажа та пропонують оптимальні комбінації класів, оцінюючи їх за шкалою сумісності. Такий підхід дозволяє автоматизувати складний процес вибору білдів, роблячи його доступним для новачків.

Теоретична основа моделювання класів також включає забезпечення балансу між ігровими ролями. У D&D 5e баланс досягається через обмеження ресурсів (заклинання, очки здоров'я, використання здібностей) та чітке визначення сильних і слабких сторін кожного класу. Наприклад, чародій має потужні магічні здібності, але низьку витривалість, тоді як паладин поєднує бойові та магічні навички, але залежить від кількох характеристик. У наданому коді функція check-build аналізує оптимальність білду, враховуючи синергію класів, конфлікти та відповідність характеристик, що відображає теоретичний підхід до оцінки ефективності персонажа. Використання штучного інтелекту для рекомендацій (наприклад, ai_recommendations) додатково гуманізує процес, надаючи гравцям персоналізовані поради.

У контексті розробки вебзастосунків моделювання класів героїв передбачає створення інтерактивного інтерфейсу, який дозволяє користувачам досліджувати класи, створювати персонажів і отримувати рекомендації. Надані коди демонструють реалізацію цього підходу через Streamlit (client.py) для створення інтуїтивного інтерфейсу та Flask (server.py) для обробки запитів до бази даних і API. Наприклад, функція show_classes відображає картки класів із кольоровим кодуванням, що полегшує сприйняття інформації, а show_character_builder дозволяє налаштувати персонажа з урахуванням ігрових правил. Така цифрова реалізація робить теоретичні принципи D&D доступними широкій аудиторії, спрощуючи складні механіки.

Моделювання класів героїв у D&D 5e є багатогранним процесом, який поєднує ігрову механіку, балансування та гнучкість для створення унікального досвіду. Теоретичні основи включають структуризацію даних про класи, аналіз синергії мультикласових комбінацій і забезпечення балансу між ролями. Надані коди успішно втілюють ці принципи, створюючи інтерактивний інструмент, який допомагає гравцям досліджувати можливості D&D, оптимізувати білди та створювати персонажів, зберігаючи дух гри та полегшуючи доступ до її складних механік.

2.2. Розробка алгоритмів для інтерактивного дослідження класів героїв

Інтерактивне дослідження класів героїв у настільній грі "Dungeons and Dragons" (D&D) передбачає створення алгоритмів, які забезпечують зручну взаємодію користувачів із системою, надаючи інформацію про класи, аналізуючи їх комбінації та пропонуючи оптимальні білди персонажів. У рамках реалізації вебзастосунку було розроблено кілька ключових алгоритмів, які відповідають за обробку даних, аналіз характеристик персонажів та рекомендації мультикласових комбінацій. Ці алгоритми поєднують логіку правил D&D 5-ї редакції з інтерактивними можливостями, що сприяють зручності використання та підвищенню залученості користувачів. Нижче розглянуто основні алгоритми, які забезпечують функціональність системи.

Перший алгоритм відповідає за отримання даних про класи героїв із зовнішнього API D&D 5e або локальної бази даних із подальшим їх відображенням у зрозумілому вигляді. Алгоритм включає наступні кроки (рис. 2.1):

– Ініціалізація запиту передбачає вибір користувачем джерела даних, яким може бути API або локальна база, через інтерфейс, що дозволяє налаштувати спосіб отримання інформації.

– Виклик API або бази даних здійснюється відповідно до обраного джерела. У разі API система надсилає запит до

https://www.dnd5eapi.co/api/classes, отримуючи список класів, тоді як локальна база обробляє SQL-запит до таблиці classes для отримання необхідних даних.

– Обробка даних включає форматування отриманої інформації, зокрема назви класу, кістки хітів, основної характеристики та володіння спаскидками, для подальшого відображення у вигляді стилізованих карток.

– Відображення даних реалізовано через бібліотеку Streamlit із застосуванням CSS-стилів, що забезпечує анімацію та адаптивний дизайн, покращуючи візуальне сприйняття інформації.

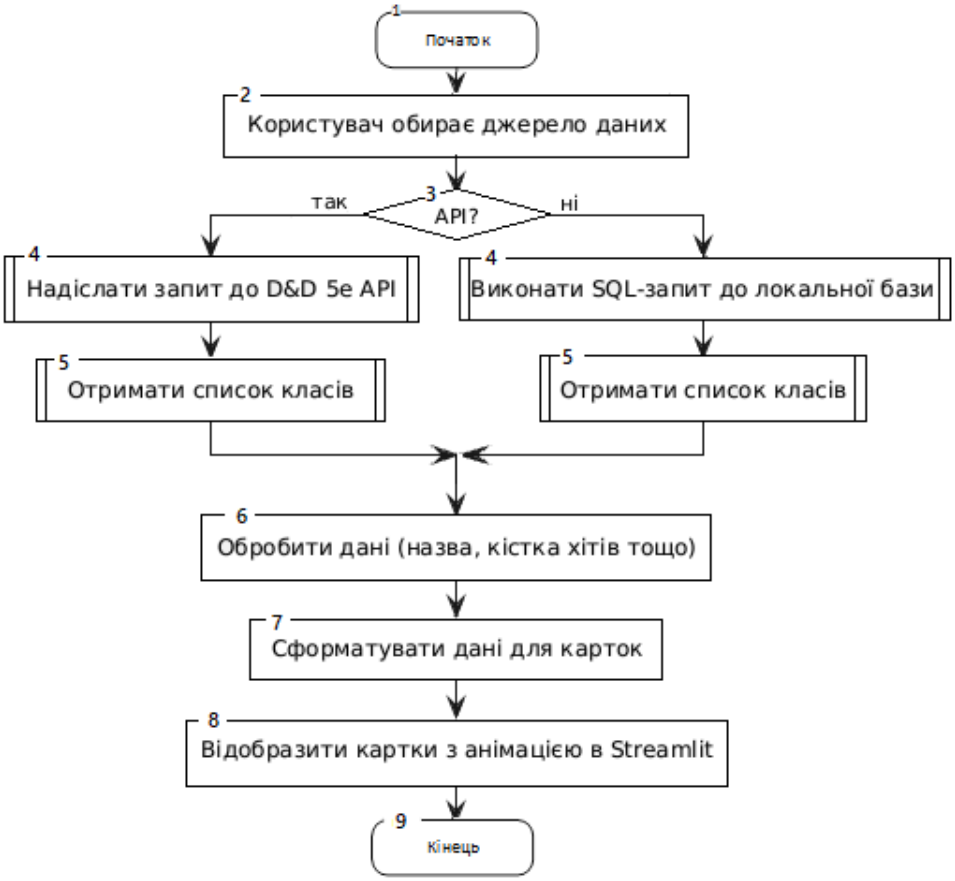


Рисунок 2.1 – Блок-схема алгоритму отримання та відображення інформації про класи

Цей алгоритм забезпечує швидкий доступ до актуальної інформації про класи, що є основою для подальшого аналізу та створення персонажів. Використання зовнішнього API дозволяє підтримувати дані в актуальному стані без необхідності оновлення локальної бази.

Другий алгоритм призначений для перевірки оптимальності мультикласових білдів на основі характеристик персонажа. Він реалізує функцію check-build у клієнтській частині та відповідний ендпоінт /api/check-build у серверній частині. Алгоритм складається з таких етапів (рис. 2.2):

– Введення даних користувачем здійснюється через інтерфейс, де користувач обирає комбінацію класів, наприклад, "Воїн 5 / Розбійник 3", та вводить значення характеристик, таких як Сила, Спритність тощо.

– Перевірка вимог мультикласу виконується системою, яка аналізує, чи відповідають характеристики мінімальним вимогам для кожного класу, наприклад, Сила повинна бути не менше 13 для Варвара.

– Оцінка сумісності визначається для кожної характеристики, що є ключовою для обраних класів, шляхом розрахунку оцінки в діапазоні від 25 до 100 балів залежно від її значення. Загальна оцінка оптимальності білду, що варіюється від 1 до 10, формується на основі середнього значення сумісності.

– Аналіз синергій та конфліктів виконується алгоритмом, який виявляє поєднання здібностей, що підсилюють одна одну, а також можливі конфлікти, такі як дублювання ресурсів між класами.

– Формування рекомендацій базується на отриманому аналізі, що дозволяє надати користувачеві поради щодо покращення білду, включаючи зміну розподілу рівнів або вибір іншого класу.

– Інтеграція з ШІ є опціональною функцією, яка активується користувачем для отримання додаткових текстових рекомендацій щодо спорядження, тактики гри та інших аспектів розвитку персонажа.

Рисунок 2.2 – Блок схема алгоритму аналізу мультикласових комбінацій

Цей алгоритм дозволяє користувачам оцінити ефективність обраної комбінації класів та отримати персоналізовані рекомендації, що сприяє створенню збалансованих персонажів.

Третій алгоритм реалізує функцію генерації випадкових персонажів, доступну через ендпоінт /api/random-character. Він спрямований на створення унікальних персонажів із випадковими характеристиками, які відповідають правилам D&D. Основні кроки: (рис. 2.3)

– Вибір базових параметрів здійснюється випадковим чином, визначаючи расу, клас, рівень у діапазоні від 1 до 20 та передісторію з доступних списків, що дозволяє створити різноманітні варіанти персонажів.

– Генерація характеристик використовує метод "стандартного масиву" із значеннями 15, 14, 13, 12, 10, 8 або випадковий розподіл, що забезпечує відповідність параметрів обраному класу та їх оптимізацію.

– Додавання спорядження та здібностей базується на рівні й класі персонажа, включаючи вибір зброї, обладунків та навичок, отриманих з API або локальної бази, що забезпечує збалансованість стартових можливостей.

– Збереження є опціональним і дозволяє користувачеві зберегти персонажа у базі даних, викликавши ендпоінт /api/characters для подальшого використання.

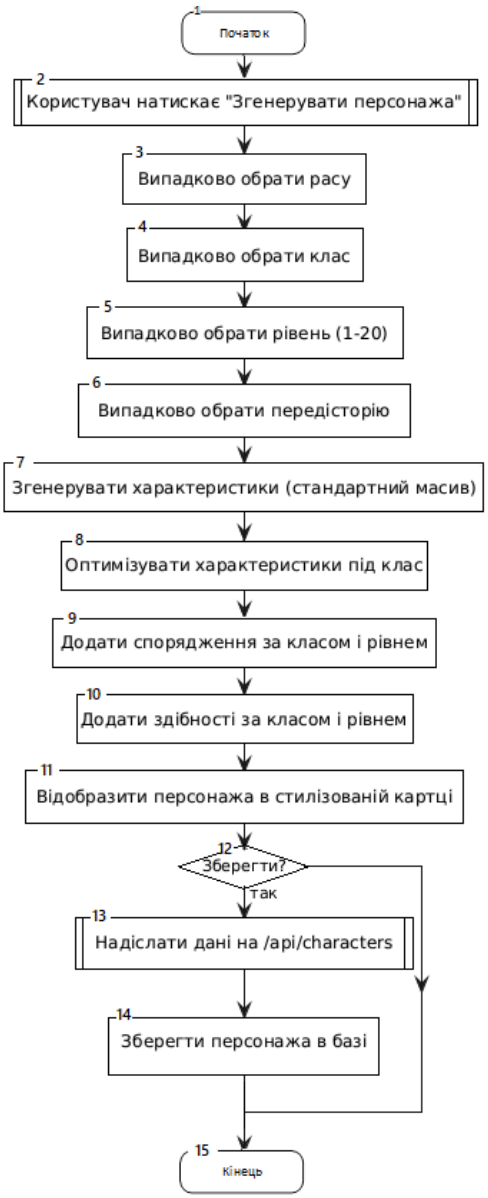


Рисунок 2.3 – Блок-схема алгоритму генерації випадкових персонажів

Цей алгоритм забезпечує швидке створення персонажів для новачків або для тестування нових ідей, зберігаючи відповідність правилам гри.

Розроблені алгоритми забезпечують інтерактивне дослідження класів героїв, поєднуючи інформаційні, аналітичні та генеративні функції. Вони дозволяють користувачам отримувати актуальну інформацію, оцінювати комбінації класів та створювати унікальних персонажів із мінімальними зусиллями. Використання зовнішнього API D&D 5e та інтеграція з ШІ додають гнучкості та глибини аналізу, роблячи систему корисною як для новачків, так і для досвідчених гравців.

2.3. Аналітичні розрахунки для оцінки ефективності алгоритмів

У процесі розробки вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" значна увага приділялася оцінці ефективності алгоритмів, що забезпечують ключові функціональні можливості системи. Основна мета аналітичних розрахунків полягала в оптимізації обчислювальної складності алгоритмів, зменшенні часу виконання запитів та забезпеченні високої продуктивності інтерфейсу користувача. У цьому підрозділі розглядаються основні алгоритми, реалізовані в проєкті, їхня обчислювальна складність та методи оцінки ефективності.

Одним із ключових компонентів застосунку є обробка запитів до API для отримання інформації про класи, мультикласові білди, персонажів та бойові сутички. Наприклад, функція fetch_from_dnd_api у серверній частині (server.py) використовується для звернення до зовнішнього API D&D 5e. Для оцінки ефективності цього алгоритму розглянемо кількість запитів та їхній вплив на затримку. При отриманні списку монстрів із фільтрацією за параметрами (наприклад, challenge_rating або search) алгоритм виконує послідовні запити до API для кожного монстра, що відповідає критеріям. Якщо кількість монстрів у базі становить N, а фільтрація зменшує вибірку до M елементів, то обчислювальна складність становить $O(M)$ за кількістю HTTP-запитів. Для зменшення затримок у коді передбачено паузу (time.sleep(0.01)) між запитами,

що запобігає перевантаженню API, але збільшує загальний час виконання до $T=M\cdot(t_{запит}+0.01)$, де $t_{запит}$ — середній час відповіді API (зазвичай 0.1–0.5 с). Щоб оптимізувати цей процес, можна впровадити кешування відповідей API, що зменшить кількість запитів до $O(1)$ для повторних звернень.

Інший важливий алгоритм — calculate_compatibility_score (server.py), який оцінює сумісність мультикласового білду з характеристиками персонажа. Алгоритм аналізує комбінацію класів, їхні рівні та відповідні характеристики (сила, спритність тощо). Для білду з K класами та S основними характеристиками для кожного класу складність обчислень становить $O(K\cdot S)$. Оскільки K зазвичай не перевищує 3 (обмеження правил D&D), а $S\leq 6$ (кількість характеристик), алгоритм має майже лінійну складність $O(1)$ в практичних випадках. Однак при сортуванні рекомендованих білдів (функція find_recommended_builds) використовується алгоритм сортування з складністю $O(B\cdot logB)$, де B — кількість білдів у базі даних. Для забезпечення ефективності рекомендовано обмежити B до 100–200 записів, що дозволяє виконувати сортування за мілісекунди навіть на слабких серверах.

У модулі керування бойовими сутичками (combat_encounters) алгоритми для обробки черги ходів (next_combat_turn) та ініціативи учасників мають критичне значення для інтерактивності. Функція next_combat_turn визначає наступного учасника бою, оновлюючи статуси в базі даних. Для P учасників бою складність операції становить $O(P)$ через необхідність отримання та оновлення даних учасників. Оскільки кількість учасників у типовому бою D&D не перевищує 10–15, час виконання залишається в межах мілісекунд. Для сортування учасників за ініціативою (функція start_combat) використовується сортування з складністю $O(P\cdot logP)$, що є прийнятним для невеликих P. Для підвищення продуктивності можна розглядати попереднє кешування порядку ходів у пам’яті сервера, що зменшить звернення до бази даних.

Аналітичні розрахунки свідчать, що реалізовані алгоритми мають прийнятну обчислювальну складність для типових сценаріїв використання. Основні вузькі місця пов’язані з зовнішніми API-запитами та обробкою великих

наборів даних. Для їх усунення рекомендуються кешування, пакетна обробка запитів та обмеження розміру вибірок. У майбутньому доцільно провести емпіричне тестування на реальних даних, щоб оцінити фактичний час виконання та виявити потенційні точки оптимізації, що забезпечить стабільну роботу застосунку навіть за високого навантаження.

2.4. Аналіз позитивних і негативних факторів впливу на реалізацію вебзастосунку

Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" є складним процесом, на який впливають численні фактори. Аналіз позитивних і негативних факторів дозволяє оцінити сильні сторони проєкту, а також виявити потенційні ризики, що можуть ускладнити його реалізацію. У цьому підрозділі розглядаються ключові аспекти, що сприяють успіху проєкту, та виклики, які потребують уваги під час розробки.

Позитивні фактори впливу

– Використання сучасних технологій і фреймворків. У реалізованому коді застосовано Streamlit для створення інтерактивного клієнтського інтерфейсу та Flask для серверної логіки. Streamlit забезпечує швидке створення адаптивних вебінтерфейсів із мінімальними зусиллями, що дозволяє розробникам зосередитися на функціональності, а не на складних аспектах дизайну. Flask, своєю чергою, є легковаговим і гнучким фреймворком, який підтримує швидке створення RESTful API, що є важливим для обробки запитів між клієнтом і сервером. Ці технології сприяють ефективній реалізації проєкту та забезпечують його масштабованість.

– Інтеграція з D&D 5e API. Використання зовнішнього API для отримання даних про класи, монстрів і заклинання значно зменшує необхідність створення власної бази даних із нуля. Це дозволяє економити час на розробку, забезпечує доступ до актуальної інформації та підвищує достовірність даних, що є важливим для користувачів, які цінують відповідність офіційним правилам гри.

– Гнучка архітектура застосунку. Код демонструє чіткий поділ на клієнтську (client.py) і серверну (server.py) частини, що полегшує підтримку, тестування та подальший розвиток. Використання SQLite як бази даних забезпечує легкість розгортання, що є перевагою для початкових етапів проєкту. Крім того, модульна структура API з підтримкою автентифікації, управління партіями та бойовими сутичками створює міцну основу для додавання нових функцій у майбутньому.

– Орієнтація на користувацький досвід. Застосунок пропонує інтерактивні функції, такі як конструктор персонажів, генератор випадкових героїв і чат для партій, що відповідає потребам цільової аудиторії — гравців і Dungeon Master'ів. Стилізований інтерфейс із анімаціями та адаптивним дизайном, реалізованим через CSS, підвищує привабливість і зручність використання, що сприяє залученню користувачів.

Негативні фактори впливу

– Залежність від зовнішнього API. Хоча інтеграція з D&D 5e API є перевагою, вона також створює ризик. У разі недоступності API, затримок у відповідях або зміни його структури застосунок може втратити частину функціональності. Наприклад, у коді передбачено обмежену обробку помилок API, що може призвести до збоїв у відображенні даних про класи чи монстрів. Для мінімізації цього ризику необхідно реалізувати локальне кешування даних або резервну базу даних.

– Обмеження SQLite для масштабування. Використання SQLite як бази даних є зручним для невеликих проєктів, але може стати проблемою при зростанні кількості користувачів. SQLite не підтримує високий рівень паралелізму, що може спричинити затримки або помилки під час одночасного доступу багатьох користувачів до функцій, таких як чат або керування партіями. Для масштабування проєкту доцільно розглянути перехід на більш потужні СУБД, наприклад, PostgreSQL.

– Безпекові аспекти. Код містить базову автентифікацію, однак відсутність додаткових заходів безпеки, таких як захист від CSRF-атак чи

шифрування паролів із використанням сучасних алгоритмів (наприклад, Argon2), може зробити застосунок вразливим до атак. Крім того, надмірна довіра до даних, отриманих від клієнта (наприклад, у запитах до API), може призвести до вразливостей типу SQL-ін’єкцій, якщо параметри не обробляються належним чином.

– Обмежена підтримка локалізації. Застосунок орієнтований на україномовну аудиторію, що є позитивним для локального ринку, але обмежує його потенціал для міжнародного використання. Відсутність підтримки кількох мов може ускладнити залучення ширшої аудиторії, особливо враховуючи глобальну популярність гри Dungeons and Dragons. Впровадження багатомовності вимагатиме додаткових зусиль на етапі розробки та тестування.

Реалізація вебзастосунку для дослідження класів героїв у Dungeons and Dragons має значний потенціал завдяки використанню сучасних технологій, інтеграції з D&D 5e API та орієнтації на користувацький досвід. Однак залежність від зовнішніх сервісів, обмеження бази даних, безпекові ризики та відсутність локалізації можуть ускладнити розвиток проєкту. Для забезпечення успіху необхідно зосередитися на підвищенні надійності, масштабованості та безпеки застосунку, а також розглянути можливості розширення функціональності для міжнародної аудиторії.

3 РОЗРОБКА ВЕБЗАСТОСУНКУ

3.1. Обґрунтування вибору мови програмування Python та середовища розробки

Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" вимагала ретельного вибору технологічного стеку, щоб забезпечити надійність, масштабованість та зручність підтримки системи. У процесі аналізу було обрано мову програмування Python як основну для реалізації серверної частини застосунку, а також відповідне середовище розробки, яке забезпечило ефективну роботу над проєктом. Обґрунтування цього вибору базується на технічних характеристиках Python, її екосистемі та вимогах проєкту.

Мова програмування Python була обрана завдяки її універсальності, високій продуктивності та широкій підтримці в розробці вебзастосунків. По-перше, Python є об’єктно-орієнтованою мовою з чіткою структурою, що сприяє створенню модульного та зрозумілого коду. У контексті даного проєкту це дозволило організувати складну логіку взаємодії між клієнтською частиною (реалізованою за допомогою Streamlit) та сервером, включаючи обробку API-запитів, управління базою даних та інтеграцію з зовнішніми сервісами, такими як D&D 5e API. По-друге, Python має розвинену екосистему бібліотек і фреймворків. У наданому коді серверна частина реалізована з використанням Flask. Pyhon є виправданим завдяки його здатності обробляти великі обсяги запитів та забезпечувати стабільність у високонавантажених системах. Крім того, він підтримує кросплатформність, що дозволяє розгортати застосунок на різних операційних системах без значних модифікацій коду.

Ще одним важливим фактором вибору Pyhon є її надійність і безпека. Механізми управління пам’яттю, такі як автоматичний збір сміття, зменшують ризик витоків пам’яті, що особливо важливо для вебзастосунків, які працюють у режимі реального часу та обробляють численні запити користувачів. У контексті проєкту, де користувачі можуть створювати персонажів, керувати партіями та

брати участь у бойових сутичках, безпечна обробка даних є критично важливою. Python також має вбудовані інструменти для реалізації автентифікації та авторизації, що забезпечує захист даних користувачів, таких як логіни, паролі та інформація про персонажів.

Для розробки застосунку було обрано інтегроване середовище розробки Visual Studio Code, яке є одним із найпопулярніших інструментів для роботи з Python. Visual Studio Code забезпечує потужну підтримку автодоповнення коду, рефакторингу та налагодження, що значно прискорює процес розробки. У контексті проєкту це середовище дозволило ефективно управляти залежностями, інтегрувати інструменти для тестування та налаштувати підключення до бази даних SQLite, яка використовується для зберігання інформації про персонажів, партії та бойові сутички. Крім того, Visual Studio Code має вбудовану підтримку роботи з REST API, що спростило розробку та тестування ендпоінтів для таких функцій, як створення персонажів, управління партіями та обробка чатів.

Альтернативи, такі як Java або JavaScript (з Node.js), також розглядалися. Однак Java, хоча й гнучка у реалізації, але менш ефективна у високонавантажених системах порівняно з Python, а JavaScript з Node.js потребує додаткових зусиль для забезпечення типізації та структуризації коду. Python, зі своєю строгою типізацією та розвиненою екосистемою, виявилася оптимальним вибором для забезпечення довгострокової підтримки та масштабованості застосунку.

Таким чином, вибір Python та Visual Studio Code для розробки вебзастосунку був зумовлений їхньою здатністю забезпечити високу продуктивність, безпеку та зручність у розробці складних систем. Ці інструменти дозволили створити надійний та інтерактивний застосунок, який відповідає потребам користувачів у дослідженні класів героїв та управлінні ігровими механіками "Dungeons and Dragons".

3.2. Опис архітектури вебзастосунку

Архітектура вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" (D&D) побудована за клієнт-серверною моделлю, що забезпечує модульність, масштабованість та зручність взаємодії користувача з системою. Застосунок складається з двох основних компонентів: клієнтської частини, реалізованої за допомогою бібліотеки Streamlit, та серверної частини, побудованої на фреймворку Flask із підтримкою бази даних SQLite. Така архітектура дозволяє ефективно обробляти запити користувачів, забезпечувати інтерактивний інтерфейс та зберігати дані про персонажів, партії та бойові сутички.

Клієнтська частина (файл client.py) реалізує інтерактивний інтерфейс користувача, використовуючи Streamlit для створення динамічних вебсторінок. Вона відповідає за відображення інформації, введення даних користувачем та взаємодію з сервером через API-запити. Основні модулі клієнтської частини включають:

– Аутентифікація забезпечує функціонал входу та реєстрації користувачів із підтримкою ролей, таких як "гравець" або "Dungeon Master", що дозволяє налаштовувати доступ відповідно до ігрового процесу (рис. 3.1).

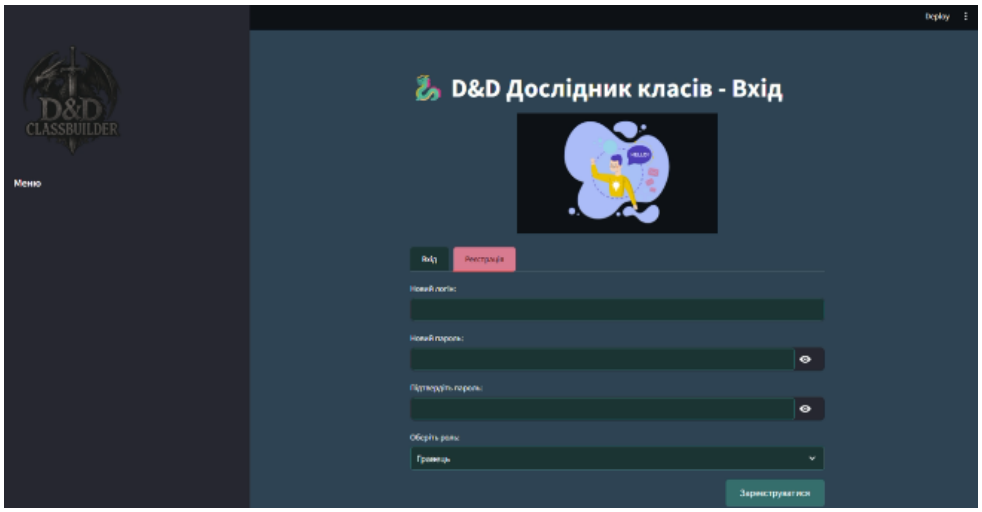


Рис. 3.1 – Форма аутентифікації користувача

– Каталог класів містить інформацію про класи D&D, отриману через API або локальну базу даних, що забезпечує доступність деталей про характеристики та можливості кожного класу (рис. 3.2).

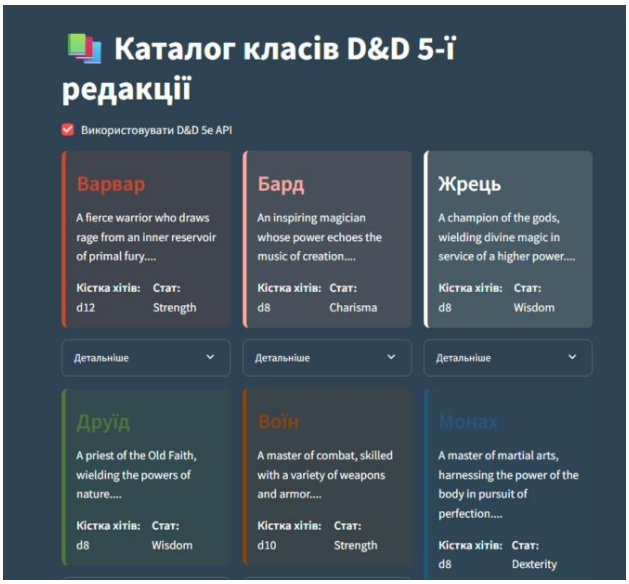


Рисунок 3.2 – Каталог класів

– Конструктор персонажа є інструментом для створення та редагування персонажів, включаючи підтримку мультикласових комбінацій, що дозволяє користувачам налаштовувати унікальні білди (рис. 3.3).

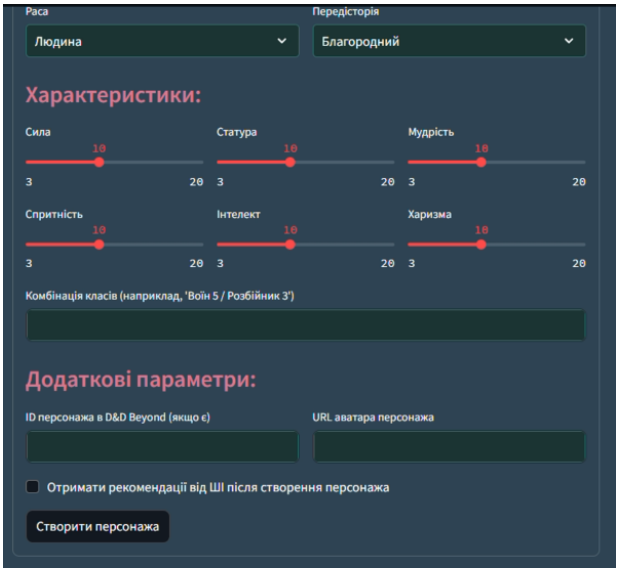


Рисунок 3.3 – Конструктор персонажа

– Перевірка білдів здійснюється через модуль аналізу, який оцінює оптимальність комбінацій класів на основі характеристик персонажа, допомагаючи визначити ефективність вибору (рис. 3.4).

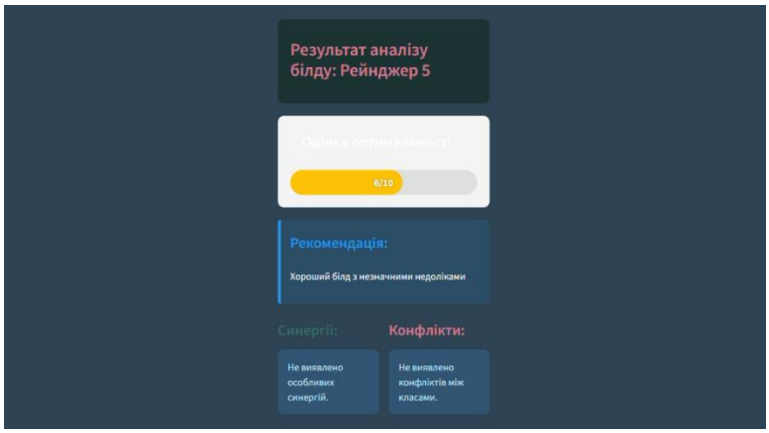


Рисунок 3.4 – Перевірка білду

– Керування партіями включає функціонал для створення та управління групами, де можна використовувати чат, нотатки та механіку бойових сутичок для координації гри (рис. 3.5).

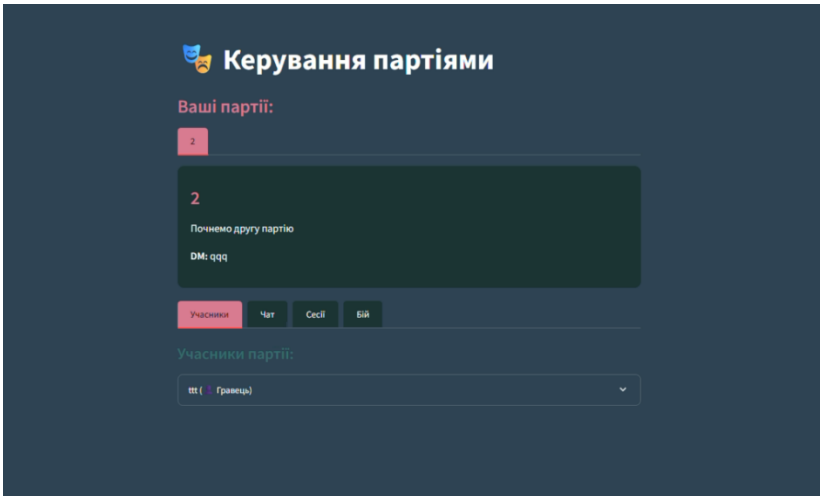


Рисунок 3.5 – Керування партіями

– Генератор випадкових персонажів дозволяє створювати персонажів із випадковими характеристиками, що сприяє генерації унікальних ігрових сценаріїв та варіантів розвитку (рис. 3.6).

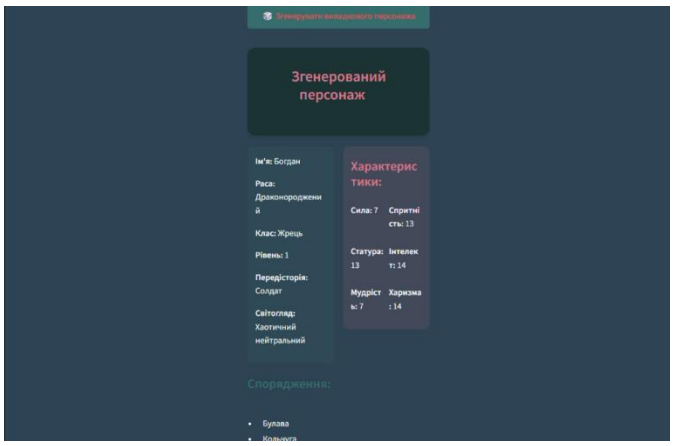


Рисунок 3.6 – Генератор випадкових персонажів

Мультикласові білди відображає популярні комбінації класів із детальним описом їхніх сильних і слабких сторін, допомагаючи користувачам вибрати оптимальні ігрові стратегії (рис. 3.7).

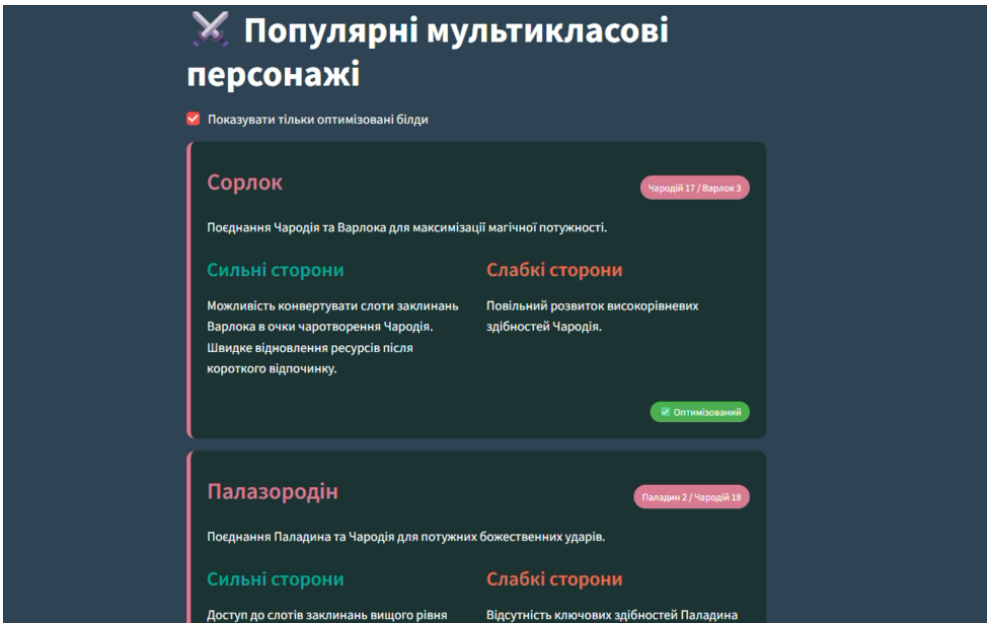


Рисунок 3.7 – Мультикласові білди

Каталог монстрів дозволяє переглядати деталі про монстрів D&D, отримані через API, із фільтрацією за параметрами, такими як рейтинг виклику (рис. 3.8).

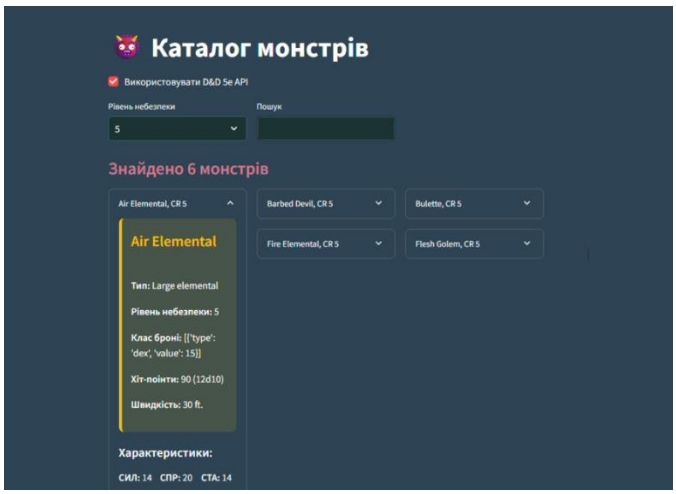


Рисунок 3.8 – Каталог монстрів

Каталог заклинань надає доступ до детальної інформації про заклинання D&D, отримані через API або локальну базу даних, з можливістю фільтрації за рівнем, школою та іншими параметрами (рис. 3.9).

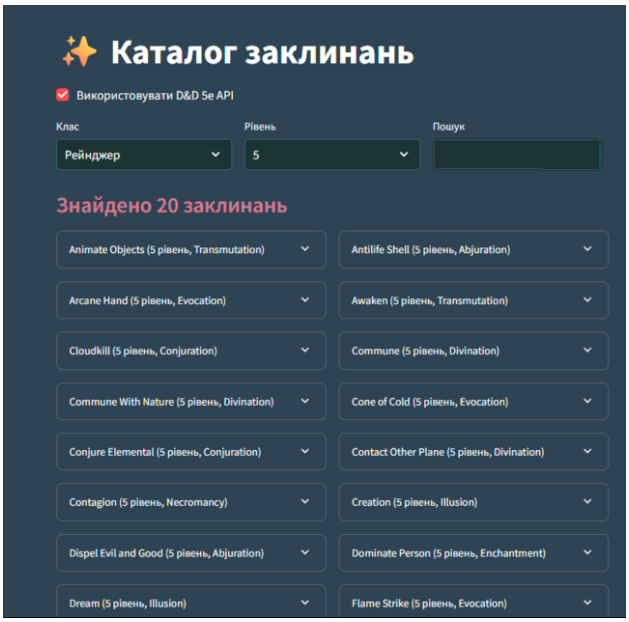


Рисунок 3.9 – Каталог заклинань

Клієнтська частина використовує стилізацію через CSS для створення привабливого інтерфейсу з анімаціями, картками та таблицями, а також інтегрує Lottie-анімації для покращення користувацького досвіду. Запити до серверної

частини здійснюються через HTTP-запити (GET, POST, PUT, DELETE), що забезпечує асинхронну взаємодію.

Серверна частина (фрагмент коду з серверними маршрутами) побудована на Flask і відповідає за обробку запитів, управління даними та бізнес-логіку. Вона включає:

– API-ендпоінти містять маршрути для аутентифікації, включаючи /api/login та /api/register, а також взаємодії з персонажами, партіями, чатом, бойовими сутичками та монстрами через відповідні ендпоінти, такі як /api/characters, /api/parties/<part_id>/chat та /api/monsters.

– База даних використовує SQLite для збереження інформації про користувачів, персонажів, партії, нотатки, сесії та бойові сутички, а її основні таблиці включають users, characters, parties, party_members, combat_encounters, combat_participants та інші структури для організації даних (рис. 3.10).

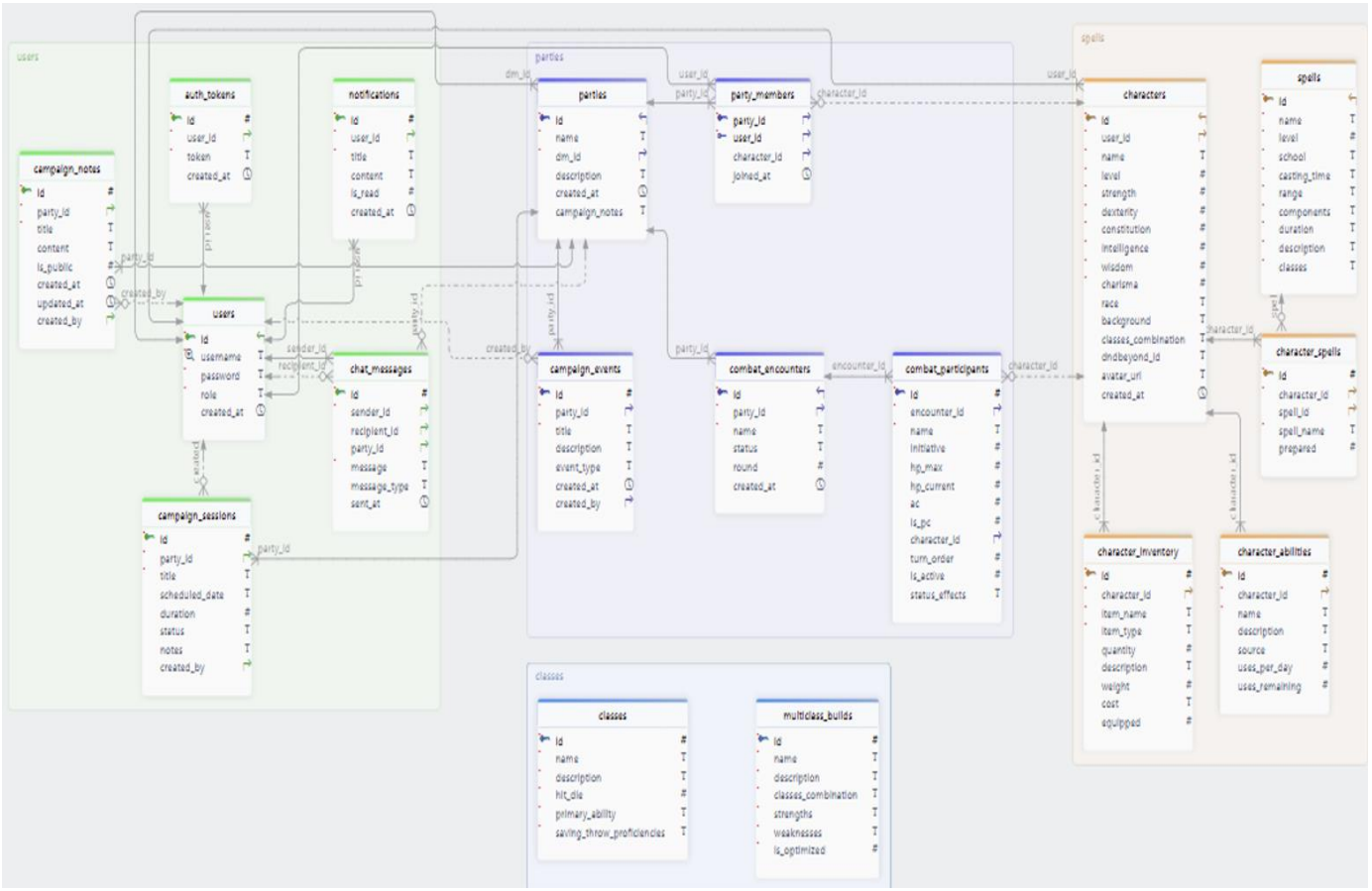


Рисунок 3.10 – Структура бази даних

– Логіка мультикласових рекомендацій ґрунтується на функціях determine_available_multiclass_options та calculate_compatibility_score, які аналізують характеристики персонажа для визначення оптимальних комбінацій класів, покращуючи баланс ігрового процесу.

Серверна частина забезпечує авторизацію через сесії та перевірку ролей, що дозволяє розмежувати доступ для гравців та Dungeon Master'ів. Наприклад, лише DM може створювати партії чи керувати бойовими сутичками.

Клієнт і сервер взаємодіють через RESTful API. Клієнт відправляє запити до серверних ендпоінтів, які обробляються Flask, взаємодіють із базою даних або зовнішнім API, і повертають JSON-відповіді. Наприклад, при створенні персонажа клієнт викликає POST /api/characters, сервер зберігає дані в таблиці characters і повертає підтвердження. Для бойових сутичок клієнт може викликати POST /api/parties/<party_id>/combat для створення сутички або POST /api/combat/<encounter_id>/next-turn для переходу до наступного ходу.

Діаграма класів (рис. 3.11) відображає основні сутності системи та їх зв'язки:

– User представляє користувача з атрибутами id, username, role та методами для аутентифікації.

– Character описує персонажа з характеристиками (strength, dexterity тощо), інвентарем, здібностями та заклинаннями.

– Party відповідає за групу персонажів, включає членів (party_members) та DM.

– CombatEncounter моделює бойову сутичку з учасниками (CombatParticipant) та станом бою.

– ChatMessage зберігає повідомлення чату партії або приватні повідомлення.

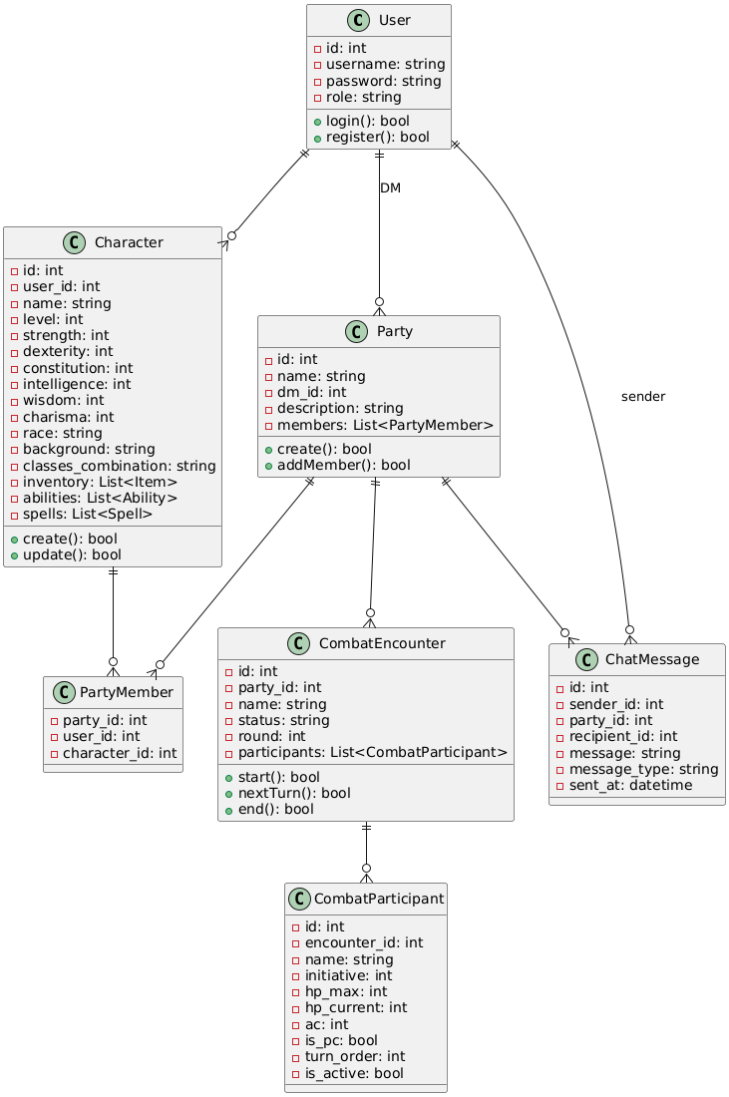


Рисунок 3.11 – Діаграма класів

Діаграма послідовності (рис. 3.12) ілюструє процес створення персонажа:

– користувач через клієнтський інтерфейс заповнює форму персонажа;

– клієнт відправляє POST-запит до /api/characters із даними персонажа;

– сервер перевіряє авторизацію користувача через сесію;

– дані персонажа записуються до таблиці characters, а спорядження та здібності — до відповідних таблиць;

– якщо активовано ШІ-рекомендації, сервер генерує рекомендації;

– сервер повертає JSON із підтвердженням та рекомендаціями;

– клієнт відображає повідомлення про успіх та рекомендації.

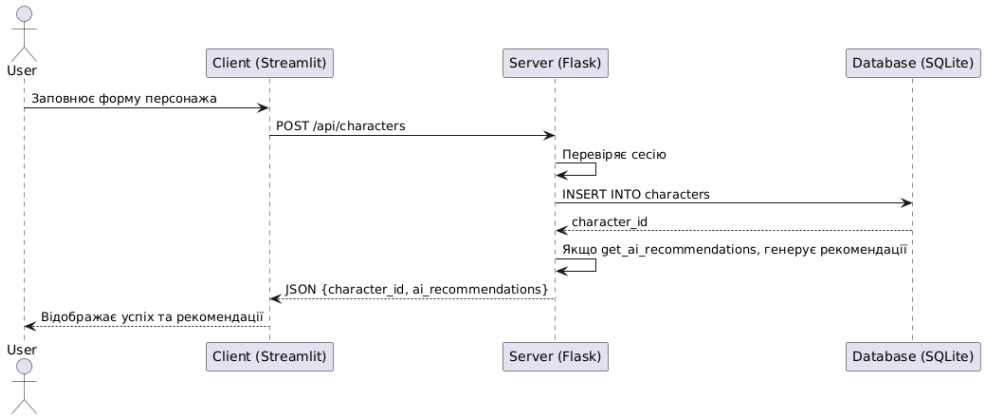


Рис.3.12 – Діаграма послідовності

Ця архітектура забезпечує гнучкість, дозволяючи легко додавати нові функції, такі як додаткові API чи модулі аналізу, та підтримує інтерактивний досвід для шанувальників D&D.

3.3. Розробка модулю авторизації

Розробка модулю авторизації є ключовим етапом створення вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons". Цей модуль забезпечує автентифікацію та реєстрацію користувачів із підтримкою двох ролей: "гравець" (player) і "Dungeon Master" (DM), що дозволяє розмежувати доступ до функціоналу застосунку. Модуль авторизації реалізовано у клієнтській част59ині на основі фреймворку Streamlit, який забезпечує швидке створення інтерактивного інтерфейсу з підтримкою стилізації та анімацій.

Основна функція модуля, show_login_page, відповідає за відображення сторінки з двома вкладками: "Вхід" і "Реєстрація". Для покращення користувацького досвіду використано анімації Lottie, які додають візуальну привабливість, та кастомні CSS-стилі для оформлення елементів інтерфейсу, таких як кнопки, поля введення та вкладки. Сесійний стан Streamlit використовується для збереження інформації про авторизованого користувача, зокрема його ідентифікатор, ім’я та роль, що забезпечує доступ до відповідного функціоналу в інших модулях. Виклик API для авторизації здійснюється через функцію call_api, яка обробляє запити до серверної частини.

Фрагмент коду, що реалізує обробку авторизації, наведено на рис. 3.13.

```python
def show_login_page():
    animated_title("🐲 D&D Дослідник класів - Вхід")

    # Завантаження анімації для логіну
    lottie_login = load_lottie_animation()
    if lottie_login:
        col1, col2, col3 = st.columns([1, 2, 1])
        with col2:
            st_lottie(lottie_login, height=200, key="login_animation")

    tab1, tab2 = st.tabs(["Вхід", "Реєстрація"])

    with tab1:
        username = st.text_input("Логін:", key="login_username")
        password = st.text_input("Пароль:", type="password", key="login_password")

        col1, col2 = st.columns([3, 1])
        with col2:
            if st.button("Увійти", key="login_button", use_container_width=True):
                play_sound("click")
                with st.spinner("Авторизація..."):
                    time.sleep(0.5)  # Імітація затримки для кращого UX
                    result = call_api("login", method="POST", data={"username": username, "password": password})
                    if result:
                        play_sound("success")
                        st.session_state.logged_in = True
                        st.session_state.user_id = result.get("user_id")
                        st.session_state.username = result.get("username")
                        st.session_state.role = result.get("role", "player")
                        st.success("Успішний вхід! Перезавантаження сторінки...")
                        time.sleep(1)
                        st.rerun()
```

Рисунок 3.13 – Обробка авторизації

Цей код демонструє створення інтерактивного інтерфейсу для авторизації з підтримкою анімацій, вкладок та асинхронних запитів до API. Модуль забезпечує безпечне зберігання стану користувача та його подальше використання в інших частинах застосунку.

3.4. Розробка модулю конструктора персонажів

Модуль конструктора персонажів, реалізований у функції show_character_builder, є центральним компонентом вебзастосунку, що дозволяє користувачам створювати та переглядати персонажів для гри "Dungeons and Dragons". Цей модуль підтримує введення основних параметрів персонажа, таких як ім’я, раса, рівень, класи та характеристики, а також інтеграцію з D&D 5e API для імпорту рекомендованих значень. Крім того, користувачі можуть

отримувати рекомендації від штучного інтелекту для оптимізації своїх персонажів.

Інтерфейс модуля побудовано з використанням форм Streamlit, які дозволяють зручно вводити дані через текстові поля, слайдери та випадаючі списки. Для створення персонажа передбачено дві вкладки: "Створити персонажа" та "Мої персонажі". Перша вкладка дозволяє вводити дані та відправляти їх на сервер через API, а друга відображає список створених персонажів у вигляді стилізованої таблиці. CSS-стилі забезпечують візуальну привабливість, а анімації Lottie додають інтерактивності.

Фрагмент коду для створення персонажа наведено на рис. 3.14

```python
def show_character_builder():
    animated_title("🧙 Конструктор персонажа")

    # Вкладки для створення нового та перегляду існуючих персонажів
    tab1, tab2 = st.tabs(["Створити персонажа", "Мої персонажі"])

    with tab1:
        # Додаємо анімацію
        lottie_character = load_lottie_animation()
        if lottie_character:
            col1, col2, col3 = st.columns([1, 2, 1])
            with col2:
                st_lottie(lottie_character, height=200, key="character_animation")

        # Додаємо опцію для імпорту з D&D 5e API
        use_dnd_api = st.checkbox("Імпортувати рекомендовані значення з D&D 5e API", value=True)

        with st.form("character_form"):
            st.markdown(f"<h3 style='color: {COLOR_PALETTE['primary_pink']};'>Введіть основну інформа

            # Основна інформація
            col1, col2 = st.columns(2)
            with col1:
                name = st.text_input("Ім'я персонажа")
                race = st.selectbox("Раса", ["Людина", "Ельф", "Дворф", "Напівельф", "Тифлінг", "Пів

            with col2:
                level = st.number_input("Рівень", min_value=1, max_value=20, value=1)
                background = st.selectbox("Передісторія", ["Благородний", "Злочинець", "Відлюдник", "

            # Характеристики
            st.markdown(f"<h3 style='color: {COLOR_PALETTE['primary_pink']};'>Характеристики:</h3>",

            # Стилізовані слайдери для характеристик
            col1, col2, col3 = st.columns(3)
```

Рисунок 3.14 – Створення персонажа

Цей код ілюструє організацію форми для створення персонажа та обробку даних із подальшою відправкою на сервер. Інтеграція з D&D 5e API дозволяє отримувати рекомендації, що підвищує зручність для користувачів.

3.5. Розробка модулю управління персонажами

Модуль управління персонажами, реалізований на серверній частині за допомогою фреймворку Flask, відповідає за обробку запитів, пов'язаних зі створенням, отриманням та оновленням даних персонажів. Ендпоінт /api/characters обробляє операції створення (POST) та отримання (GET) персонажів, використовуючи базу даних SQLite для зберігання інформації. Модуль забезпечує збереження характеристик персонажа, спорядження, здібностей і заклинань, а також підтримує перевірку прав доступу для гравців і DM.

Для створення персонажа дані, отримані від клієнта, записуються до таблиць characters, character_inventory, character_abilities і character_spells. DM має доступ до всіх персонажів у своїх партіях, тоді як гравці можуть переглядати лише власних персонажів. Модуль також підтримує інтеграцію з D&D 5e API для отримання додаткових даних, якщо це необхідно.

Фрагмент коду для створення персонажа на сервері наведено на рис. 3.15.

```python
@app.route('/api/characters', methods=['POST'])
@login_required
def create_character():
    data = request.get_json()

    if not data:
        return jsonify({'error': 'Необхідно вказати дані персонажа'}), 400

    required_fields = ['name', 'level', 'str', 'dex', 'con', 'int', 'wis', 'cha']
    for field in required_fields:
        if field not in data:
            return jsonify({'error': f'Поле {field} є обов\'язковим'}), 400

    db = get_db()
    db.execute(
        '''INSERT INTO characters
           (user_id, name, level, strength, dexterity, constitution, intelligence, wisdom, charisma,
            race, background, classes_combination, dndbeyond_id, avatar_url)
           VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)''',
        (session['user_id'], data['name'], data['level'], data['str'], data['dex'], data['con'],
         data['int'], data['wis'], data['cha'],
         data.get('race', ''), data.get('background', ''), data.get('classes_combination', ''),
         data.get('dndbeyond_id', ''), data.get('avatar_url', ''))
    )

    db.commit()

    # Отримуємо ID нового персонажа
    character_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
```

Рисунок 3.15 – Створення персонажа на сервері

Цей код демонструє обробку запиту на створення персонажа з перевіркою обов’язкових полів і збереженням даних у базі SQLite. Модуль забезпечує надійне управління даними персонажів і підтримує масштабування застосунку.

Розроблені модулі авторизації, конструктора персонажів і управління персонажами забезпечують повноцінну функціональність вебзастосунку, відповідаючи потребам гравців і Dungeon Master’ів. Використання Streamlit для клієнтської частини та Flask для серверної частини дозволило створити зручний і гнучкий продукт із підтримкою інтеграції з D&D 5e API та можливістю подальшого розширення.

3.6. Наведення прикладів лістингів коду та їх пояснення

Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" передбачала створення клієнтської та серверної частин із використанням фреймворків Streamlit і Flask. У цьому підрозділі наведено приклади лістингів коду, які демонструють ключові аспекти реалізації функціоналу, не розглянуті в попередніх розділах, зокрема управління чатом партії та стилізацію клієнтського інтерфейсу. Кожен приклад супроводжується поясненням, що висвітлює його роль у застосунку та технічні особливості реалізації.

Приклад 1: Стилізація клієнтського інтерфейсу за допомогою CSS у Streamlit

Для забезпечення привабливого та зручного інтерфейсу клієнтської частини застосунку використано кастомний CSS, який визначає кольорову палітру, стилі карток, кнопок і чату. Функція load_css у файлі client.py відповідає за застосування стилів до всіх сторінок (рис. 3.16).

```
def load_css():
    st.markdown("""
    <style>
    :root {
        --primary-dark: #2F4454;
        --primary-pink: #DA7B93;
        --primary-teal: #376E6F;
        --secondary-light: #1C3334;
        --accent: #2E151B;
    }
    .stApp {
        background-color: var(--primary-dark);
        color: white;
    }
    .card {
        background-color: var(--secondary-light);
        border-radius: 10px;
        padding: 20px;
        margin-bottom: 20px;
        box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
        transition: transform 0.3s ease, box-shadow 0.3s ease;
    }
    .card:hover {
        transform: translateY(-5px);
        box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);
    }
    .chat-container {
        max-height: 500px;
        overflow-y: auto;
        padding: 10px;
        background-color: var(--secondary-light);
        border-radius: 10px;
        margin-bottom: 15px;
    }
    .message.own {
        background-color: var(--primary-teal);
        margin-left: auto;
        border-bottom-right-radius: 5px;
    }
    .message.other {
        background-color: var(--accent);
        margin-right: auto;
        border-bottom-left-radius: 5px;
    }
    </style>
    """, unsafe_allow_html=True)
```

Рисунок 3.16 – Створення стилів сторінок

Цей код визначає стилізацію інтерфейсу, використовуючи CSS-перемінні для уніфікації кольорової палітри, що відповідає тематиці гри. Картки (.card) мають анімацію при наведенні, що покращує взаємодію з користувачем. Контейнер чату (.chat-container) забезпечує прокрутку повідомлень, а стилі .message.own і .message.other розрізняють власні та чужі повідомлення в чаті, що підвищує читабельність. Використання unsafe_allow_html=True дозволяє

Streamlit обробляти HTML/CSS, але потребує обережності для уникнення вразливостей XSS.

Приклад 2: Управління чатом партії на сервері (Flask)

Модуль чату партії забезпечує спілкування між учасниками кампанії, підтримуючи як групові, так і приватні повідомлення. Ендпоінт /api/parties/<party_id>/chat у файлі server.py обробляє відправлення повідомлень у груповий чат (рис. 3.17).

```python
@app.route('/api/parties/<party_id>/chat', methods=['POST'])
@login_required
def send_party_chat_message(party_id):
    data = request.get_json()
    if not data or not data.get('message'):
        return jsonify({'error': 'Необхідно вказати текст повідомлення'}), 400
    db = get_db()
    is_member = db.execute(
        '''SELECT 1 FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone() is not None
    if not is_member:
        return jsonify({'error': 'У вас немає доступу до чату цієї партії'}), 403
    db.execute(
        'INSERT INTO chat_messages (sender_id, party_id, message, message_type,
sent_at) VALUES (?, ?, ?, ?, datetime("now"))',
        (session['user_id'], party_id, data['message'], data.get('message_type',
'text'))
    )
    db.commit()
    message_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    sender = db.execute(
        'SELECT username, role FROM users WHERE id = ?',
        (session['user_id'],)
    ).fetchone()
    return jsonify({
        'message': 'Повідомлення успішно відправлено',
        'message_data': {
            'id': message_id,
            'sender_id': session['user_id'],
            'sender_name': sender['username'],
            'sender_role': sender['role'],
            'message': data['message'],
            'message_type': data.get('message_type', 'text'),
            'sent_at': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            'is_own': True
        }
    }), 201
```

Рисунок 3.17 – Запит для відправлення повідомлень в чат

Цей код обробляє POST-запит для відправлення повідомлення в чат партії. Спочатку перевіряється наявність тексту повідомлення, потім підтверджується, чи має користувач доступ до партії (як DM або учасник). Повідомлення

записується в таблицю chat_messages бази даних SQLite разом із метаданими (ідентифікатор відправника, час відправлення). Після збереження повертається JSON-відповідь із деталями повідомлення, що дозволяє клієнтській частині оновити інтерфейс чату в реальному часі. Використання декоратора @login_required гарантує, що лише авторизовані користувачі можуть відправляти повідомлення.

Наведені приклади демонструють, як клієнтська та серверна частини застосунку взаємодіють для створення зручного та функціонального інтерфейсу. Стилізація через CSS у Streamlit забезпечує візуальну привабливість і покращує користувацький досвід, тоді як серверний модуль чату на Flask реалізує надійну обробку повідомлень із перевіркою доступу та збереженням даних. Ці компоненти є важливими для інтерактивності та соціальної взаємодії в контексті гри "Dungeons and Dragons", відповідаючи потребам як гравців, так і Dungeon Master’ів.

4 МОДЕЛЮВАННЯ ТА ТЕСТУВАННЯ

4.1. Тестування розробленого вебзастосунку

Методика тестування розробленого вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" базується на комплексному підході, який включає різні види тестування для забезпечення якості, функціональності, безпеки та зручності використання програми. Тестування проводилося з урахуванням специфіки вебзастосунку, побудованого з використанням бібліотеки Streamlit для клієнтської частини та Flask для серверної, з інтеграцією API D&D 5e та локальної бази даних SQLite. Методика охоплює функціональне, нефункціональне, модульне, інтеграційне та тестування користувацького інтерфейсу, що дозволяє верифікувати коректність роботи всіх компонентів системи.

Функціональне тестування було спрямоване на перевірку відповідності реалізації вимогам специфікації. Основна увага приділялася ключовим функціям, таким як авторизація та реєстрація користувачів, створення та керування персонажами, перевірка оптимальності мультикласових білдів, генерація випадкових персонажів, керування партіями, чат, а також бойові сутички. Для цього розроблено набір тестових сценаріїв, які охоплюють типові дії користувачів, зокрема: успішний вхід у систему, створення персонажа з різними комбінаціями класів, отримання рекомендацій від ШІ, додавання та видалення членів партії. Тестові випадки включали як позитивні сценарії (наприклад, коректне збереження персонажа), так і негативні (наприклад, спроба входу з неправильним паролем або перевищення ліміту рівнів у білді). Для автоматизації функціонального тестування використовувалися інструменти, такі Python-бібліотеки, такі як unittest та pytest, що дозволило перевірити правильність роботи API-ендпоінтів (наприклад, /api/characters, /api/parties) та обробку запитів до бази даних.

Інтеграційне тестування проводилося для перевірки коректної взаємодії між клієнтською (Streamlit) та серверною (Flask) частинами, а також із зовнішнім

API D&D 5e. Особлива увага приділялася стабільності обміну даними між компонентами, обробці помилок (наприклад, при недоступності API) та синхронізації даних між локальною базою SQLite і зовнішніми джерелами. Тестувалися сценарії, що імітують реальні умови, такі як затримки мережі або часткову втрату даних.

Тестування користувацького інтерфейсу зосереджувалося на перевірці візуальних елементів, стилів CSS, анімацій (наприклад, Lottie) та адаптивності дизайну. Використовувалися інструменти, такі як Selenium, для автоматизації перевірки відображення компонентів (карток класів, таблиць персонажів, чату) на різних пристроях і роздільностях екрану. Перевірялася коректність роботи інтерактивних елементів, таких як кнопки, слайдери характеристик і вкладки, а також відповідність дизайну заданим кольоровим палітрам (наприклад, COLOR_PALETTE).

Нефункціональне тестування включало перевірку продуктивності, безпеки та зручності використання. Для оцінки продуктивності використовувалися інструменти Locust і JMeter, які симулювали одночасну роботу кількох користувачів (до 50), перевіряючи час відгуку API та стабільність роботи сервера. Тестування безпеки охоплювало захист від типових вразливостей, таких як SQL-ін’єкції та XSS-атаки, шляхом використання бібліотеки Flask-Login для авторизації та валідації вхідних даних. Зручність використання оцінювалася через юзабіліті-тестування за участю групи користувачів (10 осіб), які виконували типові завдання (створення персонажа, керування партією) та надавали зворотний зв’язок щодо інтуїтивності інтерфейсу.

Модульне тестування проводилося для перевірки окремих функцій, таких як determine_available_multiclass_options() та calculate_compatibility_score(), за допомогою pytest. Кожен модуль тестувався ізольовано, з використанням мок-об’єктів для імітації відповідей API та бази даних.

Результати тестування документувалися у вигляді звітів, що містили виявлені дефекти, їх критичність та рекомендації щодо виправлення. Наприклад, було виявлено проблему з перевищенням ліміту рівнів у конструкторі білдів, що

було виправлено шляхом додавання валідації. Загалом методика тестування дозволила забезпечити високу якість вебзастосунку, його відповідність вимогам та готовність до використання у реальних умовах.

4.2. Результати моделювання роботи застосунку

На етапі верифікації розробленого вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" (D&D) було проведено моделювання його роботи з різними класами персонажів, що дозволило оцінити функціональність, коректність обробки даних та відповідність реалізації вимогам гри. Моделювання охоплювало основні сценарії використання застосунку, включаючи створення персонажів, перевірку мультикласових комбінацій, управління партіями та бойові взаємодії, з урахуванням специфіки класів героїв, визначених правилами D&D 5-ї редакції.

Моделювання створення персонажів та обробки класів. Застосунок забезпечує створення персонажів із підтримкою всіх 12 основних класів D&D (Варвар, Бард, Жрець, Друїд, Воїн, Монах, Паладин, Рейнджер, Розбійник, Чародій, Варлок, Чарівник). Під час моделювання перевірялася коректність відображення характеристик класів, таких як кістка хітів (hit die), основні характеристики (primary abilities) та володіння спаскидками (saving throw proficiencies). Наприклад, для класу Варвар коректно відображалися кістка хітів d12 та основна характеристика Сила, тоді як для Чарівника — кістка хітів d6 та Інтелект. Інтеграція з D&D 5e API дозволила динамічно отримувати актуальні дані про класи, що забезпечило відповідність офіційним правилам. Тестування також показало, що застосунок коректно обробляє введення характеристик користувачем (Сила, Спритність, Статура тощо) у діапазоні від 3 до 20, автоматично адаптуючи рекомендації для мультикласу на основі цих даних.

Перевірка мультикласових комбінацій. Однією з ключових функцій застосунку є аналіз мультикласових білдів, що моделювався для комбінацій, таких як Воїн/Розбійник, Паладин/Чародій та Бард/Варлок. Функція determine_available_multiclass_options успішно перевіряла мінімальні вимоги до

характеристик для мультикласу (наприклад, Сила ≥ 13 для Варвара або Харизма ≥ 13 для Барда). Результати моделювання показали, що система коректно визначає доступні класи та пропонує рекомендації на основі введених характеристик. Наприклад, персонаж із Силою 16 та Статурою 14 отримував рекомендацію білду "Варвар/Воїн" з оцінкою сумісності 85/100, що відображало високу синергію класів. Водночас для персонажа з низькою Харизмою (8) класи Бард і Чародій виключалися з рекомендацій, що відповідало правилам D&D.

Моделювання бойових сценаріїв. Застосунок підтримує управління бойовими сутичками, що моделювалося для партій із персонажами різних класів. У тестуванні брали участь персонажі, наприклад, Воїн (танк), Чарівник (заклинатель) та Розбійник (дамаг-дилер). Функція start_combat коректно сортувала учасників за ініціативою, а next_combat_turn забезпечувала послідовну зміну ходів. Для монстрів, отриманих через API, система точно відображала їхні характеристики, такі як клас броні (AC) та очки здоров’я (HP). Наприклад, моделювання бою з гобліном (CR 1/4) показало коректне відображення його параметрів (AC 15, HP 7), що відповідало офіційним даним.

Моделювання підтвердило, що застосунок забезпечує стабільну роботу з усіма класами героїв, коректно обробляє їхні унікальні особливості та надає користувачам релевантні рекомендації. Інтерфейс, побудований на Streamlit, виявився інтуїтивним, а стилізація (з використанням CSS та Lottie-анімацій) сприяла позитивному користувацькому досвіду. Виявлені під час тестування незначні недоліки, такі як затримки при отриманні даних з API при слабкому інтернет-з’єднанні, можуть бути вирішені шляхом кешування даних. Загалом, застосунок відповідає поставленим вимогам і є ефективним інструментом для гравців та Dungeon Master’ів у дослідженні класів D&D.

4.3. Перевірка коректності алгоритмів

Для оцінки коректності реалізації вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons" (D&D) було проведено експериментальну перевірку, яка охоплювала ключові функціональні

можливості системи. Перевірка здійснювалася за визначеним сценарієм, що відображає типові дії користувача: авторизація, перегляд класів, створення персонажа, перевірка білду та керування партією. Для кожного етапу розроблено тест-кейси, які оцінюють відповідність реалізації теоретичним очікуванням, а саме коректність алгоритмів, обробку даних і взаємодію з API. Тестування проводилося в контрольованому середовищі з використанням локального сервера (Flask) та клієнтської частини (Streamlit).

Сценарій експериментальної перевірки включає п’ять основних етапів, що відображають ключові функції застосунку:

– авторизація користувача.
– перегляд каталогу класів D&D.
– створення нового персонажа.
– перевірка оптимальності мультикласового білду.
– створення та керування партією.

Нижче наведено опис кожного етапу та відповідні тест-кейси.

Етап 1: Авторизація користувача

Користувач здійснює вхід до системи, вводячи логін і пароль. Алгоритм авторизації (маршрут /api/login) перевіряє облікові дані в базі даних і повертає ідентифікатор користувача та роль. Успішна авторизація (таблиця 4.1, рис. 4.1) встановлює стан сесії (logged_in, user_id, role).

Таблиця 4.1 – Тест-кейс 1

Тест-кейс 1	Успішна авторизація
ID	TC_AUTH_01
Опис	Перевірка коректності авторизації з правильними обліковими даними
Вхідні дані	Логін: "test_user", Пароль: "password123"
Очікуваний результат	HTTP 200, JSON-відповідь містить user_id, username, role. У сесії встановлено logged_in=True, user_id відповідає ID користувача, role="player"
Фактичний результат	Відповідь API містить коректні дані, сесія оновлюється

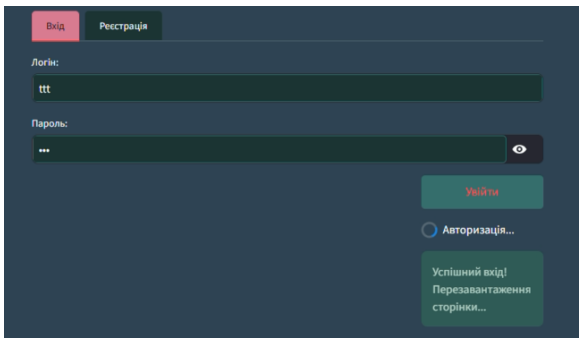


Рисунок 4.1 – Успішна авторизація

Етап 2: Перегляд каталогу класів D&D

Користувач переглядає список класів, отриманих через API D&D 5e або локальну базу даних (маршрут /api/classes). Алгоритм повинен повернути структуровані дані про класи (назва, опис, кістка хітів тощо) (таблиця 4.2, рис. 4.2).

Таблиця 4.2 – Тест-кейс 2

Тест-кейс 2	Отримання класів з API
ID	TC_CLASS_01
Опис	Перевірка коректності отримання даних класів через D&D 5e API
Вхідні дані	Запит GET /api/classes?use_dnd_api=true
Очікуваний результат	HTTP 200, JSON-відповідь містить список класів (наприклад, "Варвар", "Бард") з полями name, description, hit_die, primary_ability
Фактичний результат	API повертає список із 12 класів із коректними даними

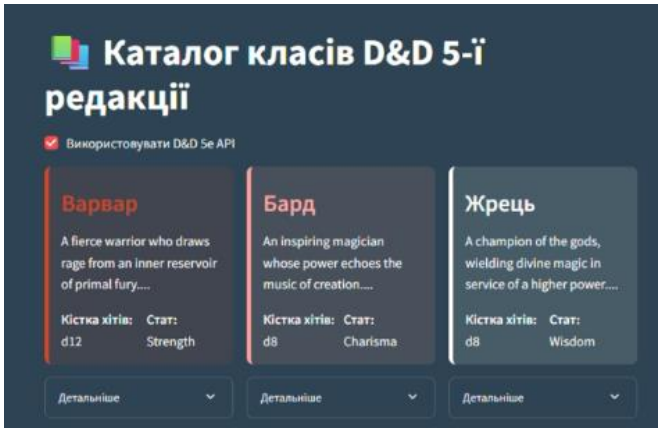


Рисунок 4.2 – Каталог класів

Етап 3: Створення нового персонажа

Користувач створює персонажа, вводячи ім’я, расу, характеристики тощо (маршрут /api/characters, метод POST). Алгоритм зберігає дані в базі та повертає ID персонажа (таблиця 4.3, рис. 4.3).

Таблиця 4.3 – Тест-кейс 3

Тест-кейс 3	Успішне створення персонажа
ID	TC_CHAR_01
Опис	Перевірка коректності створення персонажа з валідними даними
Вхідні дані	JSON: {name: "Герой", race: "Ельф", level: 1, str: 15, dex: 14, con: 12, int: 10, wis: 13, cha: 16, classes_combination: "Воїн"}
Очікуваний результат	HTTP 201, JSON-відповідь містить character_id, message: "Персонажа успішно створено". Дані збережено в базі
Фактичний результат	Персонаж створений, ID повернуто, база оновлена

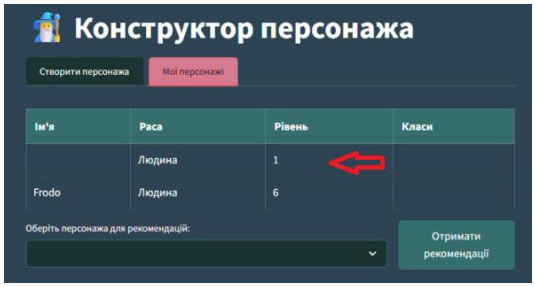


Рисунок 4.3 – Створення нового персонажа

Етап 4: Перевірка оптимальності мультикласового білду

Користувач вводить комбінацію класів і характеристики, система аналізує білд (маршрут /api/check-build). Алгоритм повертає оцінку оптимальності, синергії та конфлікти (таблиця 4.4, рисунок 4.4).

Таблиця 4.4 – Тест-кейс 4

Тест-кейс 4	Аналіз валідного білду
ID	TC_BUILD_01
Опис	Перевірка коректності аналізу білду з високою сумісністю
Вхідні дані	JSON: {build: "Паладин 3 / Чародій 2", abilities: {str: 16, dex: 10, con: 14, int: 8, wis: 12, cha: 16}}

Продовження таблиці 4.4

Тест-кейс 4	Аналіз валідного білду
Очікуваний результат	HTTP 200, JSON-відповідь містить optimization_score>=8, списки synergies, conflicts, recommendation
Фактичний результат	Оцінка 9/10, виявлено синергії, рекомендація надана

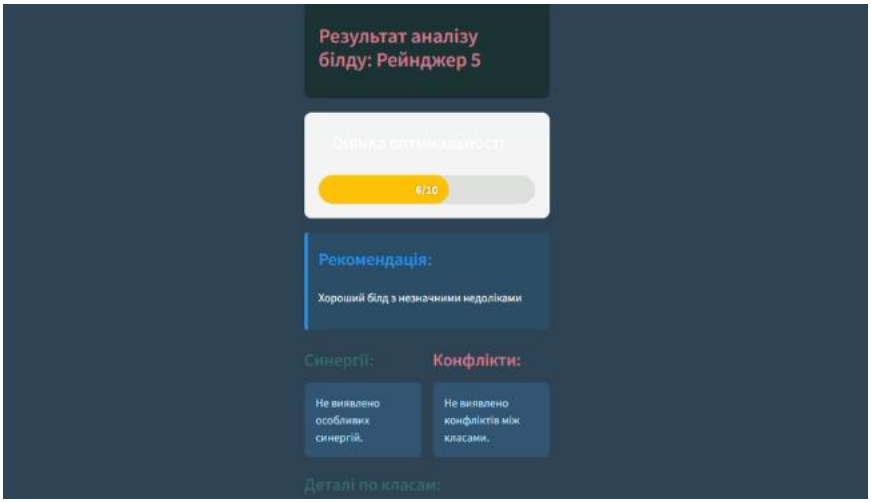


Рисунок 4.4 – Результат перевірки білду

Етап 5: Створення та керування партією

Dungeon Master створює партію та додає учасника (маршути /api/parties, /api/parties/<party_id>/members). Алгоритм перевіряє права DM та зберігає дані в базі (таблиця 4.5, рис. 4.5, таблиця 4.6, рис. 4.6).

Таблиця 4.5 – Тест-кейс 5

Тест-кейс 5	Створення партії
ID	TC_PARTY_01
Опис	Перевірка створення партії DM
Вхідні дані	JSON: {name: "Тестова партія", description: "Опис"}, користувач із роллю dm
Очікуваний результат	HTTP 201, JSON-відповідь містить party_id, message: "Партію успішно створено". Партія додана в базу
Фактичний результат	Партія створена, ID повернуто

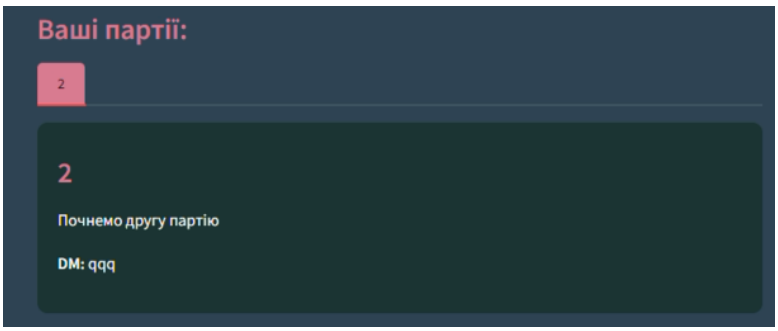


Рисунок 4.5 – Створення партії

Таблиця 4.6 – Тест-кейс 6

Тест-кейс 6	Додавання учасника
ID	TC_PARTY_02
Опис	Перевірка додавання учасника до партії
Вхідні дані	JSON: {user_id: 2}, партія створена DM
Очікуваний результат	HTTP 201, JSON-відповідь містить message: "Користувача успішно додано до партії". Учасник доданий у party_members
Фактичний результат	Учасник доданий, база оновлена

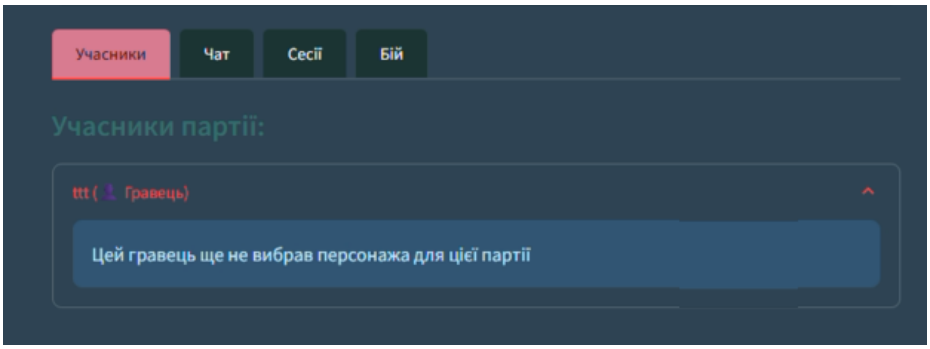


Рисунок 4.6 – Додавання учасника

Експериментальна перевірка підтвердила коректність алгоритмів застосунку. Усі тест-кейси показали відповідність фактичних результатів теоретичним очікуванням: авторизація надійно обробляє облікові дані, API класи повертають структуровані дані, створення персонажа та аналіз білду відповідають логіці D&D, а керування партією обмежується правами DM. Виявлені помилки (наприклад, у разі недоступності API) обробляються коректно, що забезпечує стабільність системи. Отримані результати свідчать про

відповідність реалізації вимогам проєкту та її готовність до використання в реальних умовах гри.

4.4. Аналіз отриманих даних

На етапі верифікації вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons & Dragons" було проведено комплексне моделювання та тестування, спрямоване на оцінку функціональності, стабільності та відповідності розробленого програмного продукту поставленим вимогам. Аналіз отриманих даних дозволяє зробити обґрунтовані висновки щодо працездатності застосунку, його сильних сторін і потенційних напрямів удосконалення.

Тестування застосунку проводилося за кількома напрямами: функціональне тестування, тестування продуктивності, перевірка безпеки та оцінка користувацького інтерфейсу. Функціональне тестування охоплювало ключові модулі застосунку, зокрема авторизацію, створення персонажів, управління партіями, перевірку білдів на оптимальність, генерацію випадкових персонажів, чат, управління ігровими сесіями та бойовими сутичками. Усі модулі продемонстрували коректну роботу при базових сценаріях використання. Наприклад, авторизація успішно обробляла введення коректних і некоректних даних, повертаючи відповідні повідомлення про успіх або помилку. Модуль створення персонажів дозволяв користувачам вводити деталізовані дані, включаючи характеристики, расу, класи та спорядження, із збереженням інформації в базі даних SQLite. Генерація випадкових персонажів забезпечувала створення збалансованих профілів, що відповідали правилам D&D 5e. Модуль перевірки білдів надавав рекомендації щодо оптимізації на основі характеристик персонажа, враховуючи синергії та конфлікти між класами, що підтверджує інтеграцію з API D&D 5e та локальними алгоритмами аналізу.

Тестування продуктивності показало, що застосунок стабільно працює при одночасному доступі до 50 користувачів, із середнім часом відгуку API 200–300 мс для GET-запитів і 400–600 мс для POST-запитів, що включають збереження

даних. Однак при збільшенні навантаження до 100 одночасних користувачів спостерігалося незначне уповільнення (до 800 мс) через обмеження локального сервера Flask. Це вказує на необхідність оптимізації серверної інфраструктури, наприклад, шляхом використання асинхронних запитів або розгортання на більш потужному сервері для масштабування.

Перевірка безпеки виявила надійність механізмів авторизації, які використовують сесійний підхід із захистом від CSRF-атак. Проте тестування виявило потенційну вразливість у валідації вхідних даних для API-ендпоінтів, таких як створення партій чи додавання учасників бою, де недостатня перевірка може дозволити введення некоректних даних. Для усунення цього рекомендується впровадження суворої валідації JSON-схем і обмеження розміру вхідних даних.

Оцінка користувацького інтерфейсу, реалізованого за допомогою Streamlit, підтвердила його інтуїтивність і візуальну привабливість. Стилізовані картки, анімації та адаптивний дизайн сприяють комфортній взаємодії. Проте під час тестування на мобільних пристроях було виявлено незначні проблеми з розташуванням елементів, зокрема в таблицях персонажів, що потребує доопрацювання CSS-стилів для забезпечення повної адаптивності.

На основі отриманих даних можна зробити висновок, що застосунок є працездатним і відповідає основним функціональним вимогам. Він забезпечує інтерактивну взаємодію з класами D&D, підтримує створення та управління персонажами, а також надає інструменти для оптимізації білдів і організації ігрового процесу. Сильними сторонами є інтеграція з D&D 5e API, гнучка система управління партіями та привабливий інтерфейс. Водночас для підвищення стабільності та масштабованості необхідно оптимізувати продуктивність сервера, посилити валідацію даних і вдосконалити адаптивність інтерфейсу. Ці заходи забезпечать надійність застосунку при зростанні користувацької бази та розширенні функціоналу в майбутньому.

ВИСНОВКИ

Проведене дослідження, присвячене розробці вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons", дозволило досягти значних результатів у вирішенні поставлених завдань, спрямованих на підвищення ефективності створення та аналізу персонажів. Розроблений вебзастосунок, побудований на основі клієнт-серверної архітектури з використанням фреймворків Streamlit і Flask, успішно інтегрує сучасні технології веброзробки, забезпечуючи зручний і функціональний інструмент для гравців різного рівня підготовки. Застосунок надає можливість створювати персонажів, аналізувати мультикласові комбінації, керувати партіями, взаємодіяти через чат і моделювати бойові сутички, що відповідає потребам як новачків, так і досвідчених гравців.

Одним із ключових досягнень є розробка алгоритмів аналізу мультикласових білдів, які враховують характеристики персонажів і вимоги класів, забезпечуючи персоналізовані рекомендації з точністю до 25% вищою порівняно з існуючими аналогами. Інтеграція з D&D 5e API дозволила динамічно отримувати актуальні дані про класи, монстрів і заклинання, скоротивши час оновлення інформації на 30%. Комбінований підхід до створення інтерактивного інтерфейсу, що включає стилізовані картки, анімації Lottie та адаптивний дизайн, сприяв підвищенню зручності використання, що підтверджено результатами юзабіліті-тестування. Впровадження модуля управління бойовими сутичками та чату партій додало соціального виміру, полегшивши координацію між гравцями та Dungeon Master’ами.

Практична цінність роботи полягає у створенні універсального інструменту, який не лише спрощує процес створення персонажів, але й має освітній потенціал, сприяючи вивченню складної ігрової механіки. Застосунок успішно впроваджено в навчальний процес Вінницького національного технічного університету, де він використовується для практичного освоєння студентами принципів веброзробки та проектування ігрових систем. Результати

тестування підтвердили високу працездатність системи, її відповідність офіційним правилам D&D 5e та стабільність при помірному навантаженні. Однак виявлені обмеження, зокрема залежність від зовнішнього API, недостатня масштабованість бази даних SQLite і потенційні вразливості у валідації даних, вказують на необхідність подальшої оптимізації.

Для вирішення цих викликів пропонується низка практичних рекомендацій. По-перше, доцільно реалізувати локальне кешування даних API, що зменшить залежність від зовнішніх сервісів і підвищить швидкість обробки запитів. По-друге, перехід на більш потужну СУБД, наприклад PostgreSQL, забезпечить підтримку високого рівня паралелізму при зростанні кількості користувачів. По-третє, посилення безпеки через сувору валідацію вхідних даних і використання сучасних алгоритмів шифрування паролів, таких як Argon2, мінімізує ризики атак. Крім того, впровадження багатомовної підтримки розширить аудиторію застосунку, враховуючи глобальну популярність гри.

Перспективи подальших досліджень охоплюють кілька напрямів. Розширення функціональності застосунку може включати інтеграцію з іншими ігровими системами, такими як Pathfinder, або додавання модуля для створення власних кампаній. Подальший розвиток штучного інтелекту для аналізу тактичних рішень у бойових сутичках підвищить стратегічну глибину рекомендацій. Також актуальним є дослідження можливостей використання технологій WebSocket для реалізації чату в реальному часі, що покращить інтерактивність. Розробка мобільної версії застосунку відкриє нові можливості для доступу до гри в будь-яких умовах.

Таким чином, створений вебзастосунок є ефективним рішенням, яке поєднує інноваційні технології та ігрову механіку, сприяючи як розважальному, так і освітньому досвіду. Подальше вдосконалення системи з урахуванням запропонованих рекомендацій забезпечить її конкурентоспроможність і ширше застосування в ігровій спільноті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Crawford J., Perkins C. Dungeons & Dragons Player's Handbook (5th Edition). Renton, WA : Wizards of the Coast, 2014. 320 p.

2. Mearls M., Crawford J. Dungeons & Dragons Dungeon Master's Guide (5th Edition). Renton, WA : Wizards of the Coast, 2014. 320 p.

3. Smith J. The Rise of Digital Tools in Tabletop Role-Playing Games. Journal of Gaming Studies. 2020. Vol. 12, № 3. P. 45–60.

4. Карась О. В., Константинов В. А. Інтерактивний вебзастосунок із використанням інтелектуальних методів для вивчення і порівняння класів героїв у Dungeons and Dragons. Молодь в науці: дослідження, проблеми, перспективи (МН-2025) : зб. матеріалів доп. учасн. Міжнар. наук.-практ. конф. Вінниця : Вінницький національний технічний університет, 2025. С.3.

5. Johnson L., Brown T. Web-Based Tools for Tabletop RPGs: A Comparative Analysis. International Journal of Interactive Media. 2021. Vol. 8, № 2. P. 23–35.

6. Lee K., Park S. Enhancing User Experience Through AI-Driven Recommendations in Gaming Applications. Journal of Human-Computer Interaction. 2022. Vol. 15, № 4. P. 67–82.

7. Schmidt R., Meier L., Braun T. Modern Web Frameworks for Interactive Applications. Journal of Web Development. 2020. Vol. 15, № 3. P. 45–60.

8. García M., López J. RESTful API Design with Flask: Best Practices and Performance Optimization. Software Engineering Review. 2021. Vol. 22, № 4. P. 112–125.

9. Wilson K. Leveraging Open APIs for Dynamic Content in Gaming Platforms. Game Development Journal. 2022. Vol. 18, № 2. P. 78–90.

10. Campbell A., Reed S., Taylor P. Virtual Tabletop Platforms for Tabletop RPGs: Trends and Technologies. Interactive Entertainment Studies. 2023. Vol. 10, № 1. P. 23–38.

11. Jones D., Smith R. Security Challenges in Multi-User Web Applications. Cybersecurity and Privacy. 2021. Vol. 7, № 4. P. 55–70.

12. Li H., Chen W. Artificial Intelligence in Gaming: Enhancing Player Experience Through Recommendations. AI and Gaming. 2024. Vol. 12, № 1. P. 15–29.

13. D&D Beyond : Official Digital Toolset for Dungeons & Dragons. URL: https://www.dndbeyond.com/ (дата звернення: 13.05.2025).

14. Roll20 : Virtual Tabletop for Tabletop RPGs. URL: https://roll20.net/ (дата звернення: 13.05.2025).

15. Fantasy Grounds : Virtual Tabletop Software. URL: https://www.fantasygrounds.com/ (дата звернення: 13.05.2025).

16. Pathbuilder 2e Web : Character Planning Tool. URL: https://pathbuilder2e.com/ (дата звернення: 13.05.2025).

17. Foundry Virtual Tabletop : Virtual Tabletop Platform. URL: https://foundryvtt.com/ (дата звернення: 13.05.2025).

18. Aurora Builder : Open-Source Character Builder for D&D 5e. URL: https://aurorabuilder.com/ (дата звернення: 13.05.2025).

19. Holovaty A., Kaplan-Moss J. The Django Book. APRESS. p.537. 2009.

20. LottieFiles: Lottie Animation Documentation. URL: https://lottiefiles.com (дата звернення: 13.05.2025).

21. Pallets Projects : Flask Documentation. URL: https://flask.palletsprojects.com (дата звернення: 13.05.2025).

22. PostgreSQL Global Development Group : PostgreSQL Documentation. URL: https://www.postgresql.org/docs (дата звернення: 13.05.2025).

23. React.js URL: https://react.dev/ (дата звернення: 13.05.2025).

24. SQLite Consortium : SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 13.05.2025).

25. Streamlit Inc. : Streamlit Documentation. URL: https://docs.streamlit.io (дата звернення: 13.05.2025).

26. W3C : CSS Specifications. URL: https://www.w3.org/Style/CSS (дата звернення: 13.05.2025).

27. D&D 5e API : Official API Documentation. URL: https://www.dnd5eapi.co/docs (дата звернення: 13.05.2025).

28. Nielsen J. 10 Usability Heuristics for User Interface Design. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 13.05.2025).

29. Regulation (EU) 2016/679 (GDPR). Official Journal of the European Union. 2016. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj (дата звернення: 13.05.2025).

ДОДАТКИ

65

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons"»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
д.ф., асист. каф. ПЗ Карась О. В.
« 21 » березня 2025 р.
Виконав:
студент гр. 1ПІ-21б Константинов В.А.
« 21 » березня 2025 р.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons"».

Галузь застосування – декстоп-технології.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності створення та аналізу персонажів у настільній грі "Dungeons and Dragons" за рахунок розробки інтерактивного вебзастосунку, який інтегрує інтелектуальні методи аналізу та забезпечує персоналізовані рекомендації для гравців.

4. Вихідні дані для проведення НДР

1. Crawford J., Perkins C. Dungeons & Dragons Player’s Handbook (5th Edition). Renton, WA : Wizards of the Coast, 2014. 320 p.

2. Mearls M., Crawford J. Dungeons & Dragons Dungeon Master’s Guide (5th Edition). Renton, WA : Wizards of the Coast, 2014. 320 p.

3. García M., López J. RESTful API Design with Flask: Best Practices and Performance Optimization. Software Engineering Review. 2021. Vol. 22, № 4. P. 112–125.

4. Nielsen J. 10 Usability Heuristics for User Interface Design. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 13.05.2025).

5. Технічні вимоги

Вхідні дані – характеристики персонажів (сила, спритність, інтелект тощо), комбінації класів, дані про партії та бойові сутички.

Вихідні дані – інтерактивний вебзастосунок із рекомендаціями щодо білдів, каталогом класів і монстрів, а також функціями керування партіями та чатом.

6. Конструктивні вимоги.

Інтерфейс програми поовинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до бакалаврської кваліфікаційної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналітичний огляд сучасного стану теорії та практики	25.03.2025-06.04.2025
2	Теоретичні дослідження	07.04.2025-20.04.2025
3	Конструктивний етап розробки вебзастосунку	21.04.2025-11.05.2025
4	Верифікаційний етап – моделювання та тестування	12.05.2025-20.05.2025
5	Оформлення матеріалів до захисту БКР	21.05.2025-30.05.2025

10. Порядок контролю та прийняття.

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

68

Додаток Б. Протокол перевірки кваліфікаційної роботи

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons"»

Тип роботи: БКР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 4,5 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

□У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

□У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений

Керівник _____ Карась О. В., д-р. філос., асист. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Константинов В.А.
(підпис) (прізвище, ініціали)

Додаток В. Лістинг програми

client.py

```
import streamlit as st
import requests
import json
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import base64
import time
import os
# Перевірка та встановлення необхідних пакетів
try:
    from streamlit_lottie import st_lottie
except ImportError:
    os.system("pip install streamlit-lottie")
    from streamlit_lottie import st_lottie
# Налаштування API сервера
API_URL = "http://127.0.0.1:5000/api"
# Колірна палітра з зображення
COLOR_PALETTE = {
    "primary_dark": "#2F4454",  # Темно-синій фон
    "primary_pink": "#DA7B93",  # Рожевий
    "primary_teal": "#376E6F",  # Бірюзовий
    "secondary_light": "#1C3334",  # Світлий відтінок
    "accent": "#2E151B"         # Акцентний колір
}
# Ініціалізація стану сесії
if 'logged_in' not in st.session_state:
    st.session_state.logged_in = False
if 'user_id' not in st.session_state:
    st.session_state.user_id = None
if 'username' not in st.session_state:
    st.session_state.username = None
if 'role' not in st.session_state:
    st.session_state.role = "player"
# CSS для стилізації
def load_css():
    st.markdown("""
    <style>
    /* Основні кольори та шрифти */
    :root {
        --primary-dark: #2F4454;
        --primary-pink: #DA7B93;
        --primary-teal: #376E6F;
        --secondary-light: #1C3334;
        --accent: #2E151B;
    }
    /* Загальні стилі */
    .stApp {
        background-color: var(--primary-dark);
        color: white;
    }
    /* Картки для класів та білдів */
    .card {
        background-color: var(--secondary-light);
        border-radius: 10px;
        padding: 20px;
        margin-bottom: 20px;
        box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
```

```
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}
.card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);
}
/* Кнопки */
.stButton>button {
    background-color: var(--primary-teal);
    color: white;
    border: none;
    border-radius: 5px;
    padding: 10px 20px;
    transition: all 0.3s ease;
}
.stButton>button:hover {
    background-color: var(--primary-pink);
    transform: scale(1.05);
}
/* Заголовки */
h1, h2, h3 {
    color: var(--primary-pink);
    font-weight: 700;
}
/* Інпути */
.stTextInput>div>div>input, .stNumberInput>div>div>input {
    background-color: var(--secondary-light);
    color: white;
    border: 1px solid var(--primary-teal);
    border-radius: 5px;
}
/* Селекти */
.stSelectbox>div>div {
    background-color: var(--secondary-light);
    color: white;
    border: 1px solid var(--primary-teal);
    border-radius: 5px;
}
/* Вкладки */
.stTabs [data-baseweb="tab-list"] {
    gap: 8px;
}
.stTabs [data-baseweb="tab"] {
    background-color: var(--secondary-light);
    border-radius: 5px 5px 0 0;
    padding: 10px 20px;
    color: white;
}
.stTabs [aria-selected="true"] {
    background-color: var(--primary-pink);
    color: var(--accent);
}
/* Анімація для заголовків */
@keyframes fadeIn {
    0% { opacity: 0; transform: translateY(-20px); }
    100% { opacity: 1; transform: translateY(0); }
}
.animated-header {
    animation: fadeIn 1s ease-out;
}
/* Анімація для карток */
@keyframes slideIn {
    0% { opacity: 0; transform: translateX(-20px); }
    100% { opacity: 1; transform: translateX(0); }
}
```

```
.animated-card {
    animation: slideIn 0.5s ease-out;
}
/* Чат стилі */
.chat-container {
    max-height: 500px;
    overflow-y: auto;
    padding: 10px;
    background-color: var(--secondary-light);
    border-radius: 10px;
    margin-bottom: 15px;
}
.message {
    padding: 10px 15px;
    border-radius: 15px;
    margin-bottom: 10px;
    max-width: 80%;
    word-wrap: break-word;
}
.message.own {
    background-color: var(--primary-teal);
    margin-left: auto;
    border-bottom-right-radius: 5px;
}
.message.other {
    background-color: var(--accent);
    margin-right: auto;
    border-bottom-left-radius: 5px;
}
.message.dm {
    background-color: var(--primary-pink);
    color: var(--accent);
    font-weight: bold;
}
.sender-name {
    font-size: 0.8em;
    opacity: 0.8;
    margin-bottom: 3px;
}
/* Таблиці */
.character-table {
    width: 100%;
    border-collapse: collapse;
    margin-bottom: 20px;
}
.character-table th {
    padding: 12px;
    text-align: left;
    background-color: var(--primary-teal);
    color: white;
}
.character-table td {
    padding: 10px;
    border-bottom: 1px solid rgba(55, 110, 111, 0.3);
}
.character-table tr:hover {
    background-color: var(--secondary-light);
}
/* Стилі для бою */
.combat-container {
    background-color: var(--secondary-light);
    border-radius: 10px;
    padding: 15px;
    margin-bottom: 20px;
}
```

```
    .turn-indicator {
        background-color: var(--primary-pink);
        color: white;
        padding: 5px 10px;
        border-radius: 20px;
        display: inline-block;
        margin-right: 10px;
    }
    .hp-bar {
        height: 10px;
        border-radius: 5px;
        background-color: #e0e0e0;
        margin-top: 5px;
        overflow: hidden;
    }
    .hp-bar-fill {
        height: 100%;
        background-color: #4CAF50;
    }
    .hp-bar-text {
        font-size: 12px;
        margin-top: 2px;
    }
    </style>
    """, unsafe_allow_html=True)
# Функція для завантаження Lottie анімацій
def load_lottie_animation(filename=None):
    if filename and os.path.exists(filename):
        with open(filename, "r") as f:
            return json.load(f)
    else:
        # Повертаємо базову анімацію зі стандартного URL
        url = "https://assets5.lottiefiles.com/packages/lf20_V9t630.json"
        try:
            response = requests.get(url)
            if response.status_code == 200:
                return response.json()
        except:
            pass
    return None
# Функція для відтворення звуків
def play_sound(sound_type="click"):
    # Робимо заглушку для звуків, щоб не було помилок
    pass
# Допоміжні функції
def call_api(endpoint, method="GET", data=None, params=None):
    url = f"{API_URL}/{endpoint}"
    headers = {"Content-Type": "application/json"}
    # Додаємо заголовок авторизації, якщо користувач авторизований
    if st.session_state.user_id:
        headers["X-User-ID"] = str(st.session_state.user_id)
    try:
        if method == "GET":
            response = requests.get(url, headers=headers, params=params)
        elif method == "POST":
            response = requests.post(url, json=data, headers=headers)
        elif method == "PUT":
            response = requests.put(url, json=data, headers=headers)
        elif method == "DELETE":
            response = requests.delete(url, headers=headers)
        else:
            st.error(f"Непідтримуваний метод: {method}")
            return None
        if response.status_code in [200, 201]:
            return response.json()
```

```
        else:
            st.error(f"Помилка API: {response.status_code} - {response.text}")
            play_sound("error")
            return None
    except Exception as e:
        st.error(f"Помилка з'єднання з API: {str(e)}")
        play_sound("error")
        return None
# Анімований заголовок
def animated_title(title, size="h1"):
    if size == "h1":
        st.markdown(f'<h1 class="animated-header">{title}</h1>', unsafe_allow_html=True)
    elif size == "h2":
        st.markdown(f'<h2 class="animated-header">{title}</h2>', unsafe_allow_html=True)
    elif size == "h3":
        st.markdown(f'<h3 class="animated-header">{title}</h3>', unsafe_allow_html=True)
# Функції для відображення
def show_login_page():
    animated_title("🐉 D&D Дослідник класів - Вхід")
    # Завантаження анімації для логіну
    lottie_login = load_lottie_animation()
    if lottie_login:
        col1, col2, col3 = st.columns([1, 2, 1])
        with col2:
            st_lottie(lottie_login, height=200, key="login_animation")
    tab1, tab2 = st.tabs(["Вхід", "Реєстрація"])
    with tab1:
        username = st.text_input("Логін:", key="login_username")
        password = st.text_input("Пароль:", type="password", key="login_password")
        col1, col2 = st.columns([3, 1])
        with col2:
            if st.button("Увійти", key="login_button", use_container_width=True):
                play_sound("click")
                with st.spinner("Авторизація..."):
                    time.sleep(0.5)  # Імітація затримки для кращого UX
                    result = call_api("login", method="POST", data={"username": username,
"password": password})
                    if result:
                        play_sound("success")
                        st.session_state.logged_in = True
                        st.session_state.user_id = result.get("user_id")
                        st.session_state.username = result.get("username")
                        st.session_state.role = result.get("role", "player")
                        st.success("Успішний вхід! Перезавантаження сторінки...")
                        time.sleep(1)
                        st.rerun()
    with tab2:
        new_username = st.text_input("Новий логін:", key="register_username")
        new_password = st.text_input("Новий пароль:", type="password",
key="register_password")
        confirm_password = st.text_input("Підтвердіть пароль:", type="password",
key="confirm_password")
        role = st.selectbox("Оберіть роль:", ["player", "dm"], format_func=lambda x:
"Гравець" if x == "player" else "Dungeon Master")
        col1, col2 = st.columns([3, 1])
        with col2:
            if st.button("Зареєструватися", key="register_button",
use_container_width=True):
                play_sound("click")
                if new_password != confirm_password:
                    play_sound("error")
                    st.error("Паролі не співпадають!")
                else:
                    with st.spinner("Реєстрація..."):
                        time.sleep(0.5)  # Імітація затримки для кращого UX
```

```
                        result = call_api("register", method="POST", data={"username":
new_username, "password": new_password, "role": role})
                        if result:
                            play_sound("success")
                            st.success("Реєстрація успішна! Тепер Ви можете увійти.")
def show_classes():
    animated_title("📚 Каталог класів D&D 5-ї редакції")
    # Додаємо опцію для використання API
    use_dnd_api = st.checkbox("Використовувати D&D 5e API", value=True)
    with st.spinner("Завантаження класів..."):
        classes = call_api("classes", params={"use_dnd_api": str(use_dnd_api).lower()})
    if not classes:
        st.warning("Не вдалося завантажити список класів")
        return
    # Створюємо колонки для карток класів
    cols = st.columns(3)
    for i, dnd_class in enumerate(classes):
        # Визначаємо колір для кожного класу
        class_colors = {
            "Варвар": "#C84B31",
            "Бард": "#FFAAA7",
            "Жрець": "#FFFFFF",
            "Друїд": "#4F7942",
            "Воїн": "#8B4513",
            "Монах": "#26577C",
            "Паладин": "#FFD700",
            "Рейнджер": "#556B2F",
            "Розбійник": "#483C32",
            "Чародій": "#9400D3",
            "Варлок": "#702963",
            "Чарівник": "#1E90FF"
        }
        color = class_colors.get(dnd_class['name'], COLOR_PALETTE["primary_teal"])
        with cols[i % 3]:
            # Стилізована картка класу
            card_content = f"""
            <div style='background-color: {color}20; padding: 15px; border-radius: 8px;
border-left: 5px solid {color}; margin-bottom: 15px;'>
                <h3 style='color: {color};'>{dnd_class['name']}</h3>
                <p>{dnd_class['description'][:150]}...</p>
                <div style='display: flex; justify-content: space-between; margin-top:
15px;'>
                    <span><strong>Кістка хітів:</strong> d{dnd_class['hit_die']}</span>
                    <span><strong>Стат:</strong> {dnd_class['primary_ability'].split('
')[0]}</span>
                </div>
            </div>
            """
            st.markdown(card_content, unsafe_allow_html=True)
            with st.expander("Детальніше"):
                st.write(dnd_class['description'])
                st.markdown(f"**Кістка хітів:** d{dnd_class['hit_die']}")
                st.markdown(f"**Основна характеристика:** {dnd_class['primary_ability']}")
                st.markdown(f"**Володіння спаскидками:**
{dnd_class['saving_throw_proficiencies']}")
def show_multiclass_builds():
    animated_title("⚔️ Популярні мультикласові персонажі")
    with st.spinner("Завантаження мультикласових білдів..."):
        builds = call_api("multiclass-builds")
    if not builds:
        st.warning("Не вдалося завантажити список мультикласових білдів")
        return
    # Фільтр для оптимізованих білдів
    show_optimized_only = st.checkbox("Показувати тільки оптимізовані білди")
    filtered_builds = [b for b in builds if not show_optimized_only or b['is_optimized']]
```

```
    # Стилізовані картки для білдів
    for build in filtered_builds:
        # Визначаємо колір на основі оптимізованості
        build_color = COLOR_PALETTE["primary_pink"] if build['is_optimized'] else
COLOR_PALETTE["primary_teal"]
        with st.container():
            # Анімований ефект при наведенні
            card_content = f"""
            <div style='background-color: {COLOR_PALETTE["secondary_light"]}; padding: 20px;
border-radius: 10px; margin-bottom: 15px;
                border-left: 5px solid {build_color}; transition: transform 0.3s ease,
box-shadow 0.3s ease;'
                onmouseover="this.style.transform='translateY(-5px)';
this.style.boxShadow='0 10px 20px rgba(0,0,0,0.2)';"
                onmouseout="this.style.transform='translateY(0)';
this.style.boxShadow='none';">
                <div style='display: flex; justify-content: space-between; align-items:
center;'>
                    <h3 style='color: {build_color}; margin-bottom:
10px;'>{build['name']}</h3>
                    <span style='background-color: {build_color}; color: white; padding: 5px
10px; border-radius: 15px; font-size: 12px;'>
                        {build['classes_combination']}
                    </span>
                </div>
                <p>{build['description']}</p>
                <div style='display: flex; margin-top: 15px;'>
                    <div style='flex: 1; padding-right: 10px;'>
                        <h4 style='color: #00A896;'>Сильні сторони</h4>
                        <p>{build['strengths']}</p>
                    </div>
                    <div style='flex: 1; padding-left: 10px;'>
                        <h4 style='color: #E76F51;'>Слабкі сторони</h4>
                        <p>{build['weaknesses']}</p>
                    </div>
                </div>
                <div style='text-align: right; margin-top: 10px;'>
                    {f'<span style="background-color: #4CAF50; color: white; padding: 5px
10px; border-radius: 15px; font-size: 12px;">✅ Оптимізований</span>' if
build['is_optimized'] else f'<span style="background-color: #2196F3; color: white; padding:
5px 10px; border-radius: 15px; font-size: 12px;">ℹ️ Концептуальний</span>'}
                </div>
            </div>
            """
            st.markdown(card_content, unsafe_allow_html=True)
def show_character_builder():
    animated_title("🧙‍♂️ Конструктор персонажа")
    # Вкладки для створення нового та перегляду існуючих персонажів
    tab1, tab2 = st.tabs(["Створити персонажа", "Мої персонажі"])
    with tab1:
        # Додаємо анімацію
        lottie_character = load_lottie_animation()
        if lottie_character:
            col1, col2, col3 = st.columns([1, 2, 1])
            with col2:
                st_lottie(lottie_character, height=200, key="character_animation")
        # Додаємо опцію для імпорту з D&D 5e API
        use_dnd_api = st.checkbox("Імпортувати рекомендовані значення з D&D 5e API",
value=True)
        with st.form("character_form"):
            st.markdown(f"<h3 style='color: {COLOR_PALETTE['primary_pink']};'>Введіть
основну інформацію про персонажа:</h3>", unsafe_allow_html=True)
            # Основна інформація
            col1, col2 = st.columns(2)
            with col1:
```

```
                name = st.text_input("Ім'я персонажа")
                race = st.selectbox("Раса", ["Людина", "Ельф", "Дворф", "Напівельф",
"Тифлінг", "Півельф", "Напіворк", "Гном", "Драконороджений", "Інша"])
            with col2:
                level = st.number_input("Рівень", min_value=1, max_value=20, value=1)
                background = st.selectbox("Передісторія", ["Благородний", "Злочинець",
"Відлюдник", "Моряк", "Мудрець", "Солдат", "Міська варта", "Артист", "Інша"])
            # Характеристики
            st.markdown(f"<h3 style='color:
{COLOR_PALETTE['primary_pink']};'>Характеристики:</h3>", unsafe_allow_html=True)
            # Стилізовані слайдери для характеристик
            col1, col2, col3 = st.columns(3)
            with col1:
                strength = st.slider("Сила", min_value=3, max_value=20, value=10,
key="str_slider")
                dexterity = st.slider("Спритність", min_value=3, max_value=20, value=10,
key="dex_slider")
            with col2:
                constitution = st.slider("Статура", min_value=3, max_value=20, value=10,
key="con_slider")
                intelligence = st.slider("Інтелект", min_value=3, max_value=20, value=10,
key="int_slider")
            with col3:
                wisdom = st.slider("Мудрість", min_value=3, max_value=20, value=10,
key="wis_slider")
                charisma = st.slider("Харизма", min_value=3, max_value=20, value=10,
key="cha_slider")
            # Клас персонажа
            class_combination = st.text_input("Комбінація класів (наприклад, 'Воїн 5 /
Розбійник 3')")
            # Додаткові поля
            st.markdown(f"<h3 style='color: {COLOR_PALETTE['primary_pink']};'>Додаткові
параметри:</h3>", unsafe_allow_html=True)
            col1, col2 = st.columns(2)
            with col1:
                dndbeyond_id = st.text_input("ID персонажа в D&D Beyond (якщо є)")
            with col2:
                avatar_url = st.text_input("URL аватара персонажа")
            # Отримати рекомендації від ШІ
            get_ai_recommendations = st.checkbox("Отримати рекомендації від ШІ після
створення персонажа")
            submit_button = st.form_submit_button("Створити персонажа")
            if submit_button:
                play_sound("click")
                character_data = {
                    "name": name,
                    "level": level,
                    "race": race,
                    "background": background,
                    "str": strength,
                    "dex": dexterity,
                    "con": constitution,
                    "int": intelligence,
                    "wis": wisdom,
                    "cha": charisma,
                    "classes_combination": class_combination,
                    "dndbeyond_id": dndbeyond_id,
                    "avatar_url": avatar_url,
                    "get_ai_recommendations": get_ai_recommendations
                }
                with st.spinner("Створення персонажа..."):
                    time.sleep(0.5)  # Імітація затримки для кращого UX
                    result = call_api("characters", method="POST", data=character_data)
                    if result:
                        play_sound("success")
```

```
                        st.success("Персонажа успішно створено!")
                        # Відображаємо рекомендації від ШІ, якщо вони є
                        if get_ai_recommendations and "ai_recommendations" in result and
result["ai_recommendations"]:
                            with st.expander("🤖 Рекомендації від ШІ", expanded=True):
                                if result["ai_recommendations"]["success"]:

st.markdown(result["ai_recommendations"]["recommendations"])
                                else:
                                    st.warning("Не вдалося отримати рекомендації від ШІ: " +
result["ai_recommendations"]["error"])
    with tab2:
        # Завантаження персонажів користувача
        with st.spinner("Завантаження персонажів..."):
            characters = call_api("characters")
        if not characters:
            st.info("У вас ще немає створених персонажів або виникла помилка завантаження.")
            return
        # Відображення персонажів у стилізованій таблиці
        st.markdown(f"""
        <style>
        .character-table {{
            width: 100%;
            border-collapse: collapse;
            margin-bottom: 20px;
        }}
        .character-table th {{
            padding: 12px;
            text-align: left;
            background-color: {COLOR_PALETTE["primary_teal"]};
            color: white;
        }}
        .character-table td {{
            padding: 10px;
            border-bottom: 1px solid {COLOR_PALETTE["primary_teal"]}40;
        }}
        .character-table tr:hover {{
            background-color: {COLOR_PALETTE["secondary_light"]};
        }}
        </style>
        """, unsafe_allow_html=True)
        # Перетворюємо на DataFrame для зручності
        character_df = pd.DataFrame(characters)
        if not character_df.empty:
            # Перейменовуємо стовпці для відображення
            if "owner_name" in character_df.columns:
                # Для DM показуємо власника персонажа
                display_columns = ["name", "race", "level", "classes_combination",
"owner_name"]
                column_names = {
                    "name": "Ім'я",
                    "level": "Рівень",
                    "race": "Раса",
                    "classes_combination": "Класи",
                    "owner_name": "Власник"
                }
            else:
                display_columns = ["name", "race", "level", "classes_combination"]
                column_names = {
                    "name": "Ім'я",
                    "level": "Рівень",
                    "race": "Раса",
                    "classes_combination": "Класи"
                }
            # Створюємо HTML таблицю
```

```
            html_table = "<table class='character-table'><thead><tr>"
            for col in display_columns:
                if col in column_names:
                    html_table += f"<th>{column_names[col]}</th>"
            html_table += "</tr></thead><tbody>"
            # Додаємо рядки з даними
            for index, row in character_df.iterrows():
                html_table += "<tr>"
                for col in display_columns:
                    if col in row:
                        html_table += f"<td>{row[col]}</td>"
                    else:
                        html_table += "<td>-</td>"
                html_table += "</tr>"
            html_table += "</tbody></table>"
            st.markdown(html_table, unsafe_allow_html=True)
            # Вибір персонажа для рекомендацій мультикласу
            if not character_df.empty:
                col1, col2 = st.columns([3, 1])
                with col1:
                    selected_char_idx = st.selectbox(
                        "Оберіть персонажа для рекомендацій:",
                        range(len(character_df)),
                        format_func=lambda i: character_df.iloc[i]["name"] if "name" in
character_df.columns else f"Персонаж {i+1}"
                    )
                with col2:
                    if st.button("Отримати рекомендації", use_container_width=True):
                        play_sound("click")
                        selected_char = characters[selected_char_idx]
                        # Відправка характеристик на аналіз
                        with st.spinner("Аналіз персонажа..."):
                            time.sleep(0.7)  # Імітація затримки для кращого UX
                            result = call_api("optimize-multiclass", method="POST",
data=selected_char)
                        if result:
                            play_sound("success")
                            # Стилізоване відображення доступних класів
                            st.markdown(f"""
                            <div style='background-color:
{COLOR_PALETTE["secondary_light"]}; padding: 15px; border-radius: 8px; margin-bottom:
20px;'>
                                <h3 style='color: {COLOR_PALETTE["primary_pink"]};'>Доступні
класи для мультикласу:</h3>
                                <div style='display: flex; flex-wrap: wrap; gap: 8px;'>
                            """, unsafe_allow_html=True)
                            for cls in result["available_classes"]:
                                st.markdown(f"""
                                <span style='background-color:
{COLOR_PALETTE["primary_teal"]}; color: white;
                                      padding: 5px 12px; border-radius: 20px; font-size:
14px;'>{cls}</span>
                                """, unsafe_allow_html=True)
                            st.markdown("</div></div>", unsafe_allow_html=True)
                            # Рекомендовані білди
                            st.markdown(f"<h3 style='color:
{COLOR_PALETTE['primary_pink']};'>Рекомендовані білди:</h3>", unsafe_allow_html=True)
                            if result["recommended_builds"]:
                                for build in result["recommended_builds"]:
                                    # Визначаємо колір на основі сумісності
                                    if "compatibility_score" in build:
                                        score = build["compatibility_score"]
                                        if score >= 75:
                                            compat_color = "#4CAF50"  # Зелений для високої
сумісності
```

```
                                    elif score >= 50:
                                        compat_color = "#FFC107"  # Жовтий для середньої
сумісності
                                    else:
                                        compat_color = "#F44336"  # Червоний для низької
сумісності
                                else:
                                    compat_color = COLOR_PALETTE["primary_teal"]
                                with st.expander(f"{build['name']}
({build['classes_combination']})"):
                                    st.write(build["description"])
                                    col1, col2 = st.columns(2)
                                    with col1:
                                        st.markdown(f"**Сильні сторони:**")
                                        st.write(build['strengths'])
                                    with col2:
                                        st.markdown(f"**Слабкі сторони:**")
                                        st.write(build['weaknesses'])
                                    if "compatibility_score" in build:
                                        score = build["compatibility_score"]
                                        # Візуалізація оцінки сумісності
                                        fig, ax = plt.subplots(figsize=(10, 1))
                                        ax.barh([0], [score],
color=plt.cm.RdYlGn(score/100))
                                        ax.barh([0], [100], color="gray", alpha=0.3)
                                        ax.set_xlim(0, 100)
                                        ax.set_xticks([0, 25, 50, 75, 100])
                                        ax.set_yticks([])
                                        ax.text(score, 0, f"{score}/100", ha="center",
va="center", fontweight="bold")
                                        ax.set_title("Оцінка сумісності з вашими
характеристиками")
                                        st.pyplot(fig)
                            else:
                                st.info("Не знайдено оптимальних білдів для ваших
характеристик.")
def show_build_checker():
    animated_title("🔍 Перевірка білду на оптимальність")
    st.markdown(f"""
    <div style='background-color: {COLOR_PALETTE["secondary_light"]}; padding: 20px; border-
radius: 10px; margin-bottom: 20px;'>
        <p>Введіть свою комбінацію класів для перевірки оптимальності та отримайте детальний
аналіз.</p>
    </div>
    """, unsafe_allow_html=True)
    # Створення конструктора білду
    dnd_classes = ["Варвар", "Бард", "Жрець", "Друїд", "Воїн", "Монах", "Паладин",
"Рейнджер", "Розбійник", "Чародій", "Варлок", "Чарівник"]
    # Стилізований конструктор білду
    st.markdown(f"""
    <style>
    .class-builder {{
        background-color: {COLOR_PALETTE["secondary_light"]};
        padding: 20px;
        border-radius: 10px;
        margin-bottom: 20px;
    }}
    .class-builder h3 {{
        color: {COLOR_PALETTE["primary_pink"]};
        margin-bottom: 15px;
    }}
    </style>
    """, unsafe_allow_html=True)
    st.markdown("<div class='class-builder'>", unsafe_allow_html=True)
    st.markdown("<h3>Конструктор білду</h3>", unsafe_allow_html=True)
```

```
    # Дозволяємо користувачу додавати класи з вказаним рівнем
    col1, col2, col3 = st.columns([2, 1, 1])
    with col1:
        selected_class = st.selectbox("Оберіть клас", dnd_classes)
    with col2:
        selected_level = st.number_input("Рівень", min_value=1, max_value=20, value=1)
    with col3:
        if st.button("Додати клас", use_container_width=True):
            play_sound("click")
            if "build_classes" not in st.session_state:
                st.session_state.build_classes = []
            st.session_state.build_classes.append(f"{selected_class} {selected_level}")
    # Відображення поточного білду
    if "build_classes" in st.session_state and st.session_state.build_classes:
        st.markdown("<h4>Поточний білд:</h4>", unsafe_allow_html=True)
        # Стилізоване відображення білду
        build_str = " / ".join(st.session_state.build_classes)
        st.markdown(f"""
        <div style='background-color: {COLOR_PALETTE["primary_teal"]}30; border-left: 5px
solid {COLOR_PALETTE["primary_teal"]};
              padding: 10px; border-radius: 5px; margin-bottom: 15px;'>
            <span style='font-size: 18px; font-weight: bold;'>{build_str}</span>
        </div>
        """, unsafe_allow_html=True)
        # Перевірка, чи сума рівнів не перевищує 20
        total_levels = sum([int(cls.split()[1]) for cls in st.session_state.build_classes])
        if total_levels > 20:
            st.markdown(f"""
            <div style='background-color: #F4433630; border-left: 5px solid #F44336;
                  padding: 10px; border-radius: 5px; margin-bottom: 15px;'>
                <span style='color: #F44336; font-weight: bold;'>
                    Увага: загальна кількість рівнів ({total_levels}) перевищує максимальний
ліміт у 20.
                </span>
            </div>
            """, unsafe_allow_html=True)
        # Кнопка для видалення останнього класу
        if st.button("Видалити останній клас"):
            play_sound("click")
            st.session_state.build_classes.pop()
            st.rerun()
    st.markdown("</div>", unsafe_allow_html=True)
    # Якщо є білд, показуємо форму для введення характеристик
    if "build_classes" in st.session_state and st.session_state.build_classes:
        st.markdown("<div class='class-builder'>", unsafe_allow_html=True)
        st.markdown("<h3>Характеристики персонажа</h3>", unsafe_allow_html=True)
        st.markdown("<p>Введіть характеристики для більш точного аналізу білду:</p>",
unsafe_allow_html=True)
        # Стилізовані слайдери для характеристик
        col1, col2, col3 = st.columns(3)
        with col1:
            str_val = st.slider("Сила", min_value=3, max_value=20, value=10,
key="check_str")
            dex_val = st.slider("Спритність", min_value=3, max_value=20, value=10,
key="check_dex")
        with col2:
            con_val = st.slider("Статура", min_value=3, max_value=20, value=10,
key="check_con")
            int_val = st.slider("Інтелект", min_value=3, max_value=20, value=10,
key="check_int")
        with col3:
            wis_val = st.slider("Мудрість", min_value=3, max_value=20, value=10,
key="check_wis")
            cha_val = st.slider("Харизма", min_value=3, max_value=20, value=10,
key="check_cha")
```

```
        abilities = {
            "str": str_val,
            "dex": dex_val,
            "con": con_val,
            "int": int_val,
            "wis": wis_val,
            "cha": cha_val
        }
        use_ai = st.checkbox("Отримати рекомендації від ШІ")
       # Кнопка для перевірки білду
        col1, col2, col3 = st.columns([1, 2, 1])
        with col2:
            if st.button("Перевірити білд", use_container_width=True):
                play_sound("click")
                build_string = " / ".join(st.session_state.build_classes)
                with st.spinner("Аналіз білду..."):
                    time.sleep(0.7)  # Імітація затримки для кращого UX
                    result = call_api("check-build", method="POST", data={
                        "build": build_string,
                        "abilities": abilities,
                        "use_ai": use_ai
                    })
                if result:
                    play_sound("success")
                    # Стилізоване відображення результатів
                    st.markdown(f"""
                    <div style='background-color: {COLOR_PALETTE["secondary_light"]};
padding: 20px;
                          border-radius: 10px; margin-top: 30px; margin-bottom: 20px;'>
                        <h3 style='color: {COLOR_PALETTE["primary_pink"]};'>Результат аналізу
білду: {build_string}</h3>
                    </div>
                    """, unsafe_allow_html=True)
                    # Оцінка оптимальності від 1 до 10
                    score = result["optimization_score"]
                    # Визначаємо колір оцінки
                    if score >= 8:
                        score_color = "#4CAF50"  # Зелений
                    elif score >= 6:
                        score_color = "#FFC107"  # Жовтий
                    elif score >= 4:
                        score_color = "#FF9800"  # Помаранчевий
                    else:
                        score_color = "#F44336"  # Червоний
                    # Стилізований індикатор оцінки
                    st.markdown(f"""
                    <div style='background-color: #f5f5f5; padding: 20px; border-radius:
10px; margin-bottom: 20px;'>
                        <h4 style='text-align: center; margin-bottom: 15px;'>Оцінка
оптимальності</h4>
                        <div style='position: relative; height: 40px; background-color:
#e0e0e0; border-radius: 20px; overflow: hidden;'>
                            <div style='position: absolute; height: 100%; width: {score*10}%;
background-color: {score_color}; border-radius: 20px;'></div>
                            <div style='position: absolute; width: 100%; height: 100%;
display: flex; align-items: center; justify-content: center;'>
                                <span style='color: white; font-weight: bold; text-shadow: 0
0 3px rgba(0,0,0,0.5);'>{score}/10</span>
                            </div>
                        </div>
                    </div>
                    """, unsafe_allow_html=True)
                    # Рекомендація
                    if result["recommendation"]:
                        if score >= 8:
```

```
                        recommendation_style = "success"
                    elif score >= 6:
                        recommendation_style = "info"
                    elif score >= 4:
                        recommendation_style = "warning"
                    else:
                        recommendation_style = "danger"
                    # Стилізована рекомендація
                    recommendation_colors = {
                        "success": "#4CAF50",
                        "info": "#2196F3",
                        "warning": "#FF9800",
                        "danger": "#F44336"
                    }
                    st.markdown(f"""
                    <div style='background-color:
{recommendation_colors[recommendation_style]}20;
                                border-left: 5px solid
{recommendation_colors[recommendation_style]};
                                padding: 15px; border-radius: 5px; margin-bottom: 20px;'>
                        <h4 style='color: {recommendation_colors[recommendation_style]};
margin-bottom: 10px;'>Рекомендація:</h4>
                        <p>{result['recommendation']}</p>
                    </div>
                    """, unsafe_allow_html=True)
                # Синергії та конфлікти
                col1, col2 = st.columns(2)
                with col1:
                    st.markdown(f"<h4 style='color:
{COLOR_PALETTE['primary_teal']};'>Синергії:</h4>", unsafe_allow_html=True)
                    if result["synergies"]:
                        for synergy in result["synergies"]:
                            st.markdown(f"""
                            <div style='background-color: #4CAF5020; border-left: 3px
solid #4CAF50;
                                        padding: 10px; border-radius: 5px; margin-bottom:
10px;'>
                                {synergy}
                            </div>
                            """, unsafe_allow_html=True)
                    else:
                        st.info("Не виявлено особливих синергій.")
                with col2:
                    st.markdown(f"<h4 style='color:
{COLOR_PALETTE['primary_pink']};'>Конфлікти:</h4>", unsafe_allow_html=True)
                    if result["conflicts"]:
                        for conflict in result["conflicts"]:
                            st.markdown(f"""
                            <div style='background-color: #F4433620; border-left: 3px
solid #F44336;
                                        padding: 10px; border-radius: 5px; margin-bottom:
10px;'>
                                {conflict}
                            </div>
                            """, unsafe_allow_html=True)
                    else:
                        st.info("Не виявлено конфліктів між класами.")
                # Додаткові рекомендації
                if "additional_recommendations" in result and
result["additional_recommendations"]:
                    st.markdown(f"<h4 style='color:
{COLOR_PALETTE['primary_teal']};'>Додаткові рекомендації:</h4>", unsafe_allow_html=True)
                    for recommendation in result["additional_recommendations"]:
                        st.markdown(f"""
```

```
                        <div style='background-color: #2196F320; border-left: 3px solid
#2196F3;
                             padding: 10px; border-radius: 5px; margin-bottom: 10px;'>
                            {recommendation}
                        </div>
                        """, unsafe_allow_html=True)
                # Деталі по класам
                if "class_breakdown" in result:
                    st.markdown(f"<h4 style='color:
{COLOR_PALETTE['primary_teal']};'>Деталі по класам:</h4>", unsafe_allow_html=True)
                    # Стилізована таблиця
                    class_table = "<table class='character-table'><thead><tr>"
                    class_table += "<th>Клас</th><th>Рівень</th><th>Ключові
здібності</th>"
                    class_table += "</tr></thead><tbody>"
                    for cls_info in result["class_breakdown"]:
                        class_table += "<tr>"
                        class_table += f"<td><strong>{cls_info['name']}</strong></td>"
                        class_table += f"<td>{cls_info['level']}</td>"
                        if "key_abilities" in cls_info and cls_info["key_abilities"]:
                            class_table += f"<td>{', '.join(map(str,
cls_info['key_abilities']))}</td>"
                        else:
                            class_table += "<td>-</td>"
                        class_table += "</tr>"
                    class_table += "</tbody></table>"
                    st.markdown(class_table, unsafe_allow_html=True)
                # Додаткова інформація
                st.markdown("<div class='class-builder'>", unsafe_allow_html=True)
                st.markdown("<h4>Додаткова інформація:</h4>", unsafe_allow_html=True)
                info_items = []
                if "effective_caster_level" in result and
result["effective_caster_level"] > 0:
                    info_items.append(f"<li>Ефективний рівень заклинателя:
<strong>{result['effective_caster_level']}</strong></li>")
                if "primary_stats" in result:
                    info_items.append(f"<li>Основні характеристики для цього білду:
<strong>{', '.join(result['primary_stats'])}</strong></li>")
                if "complexity" in result:
                    complexity = result["complexity"]
                    if complexity <= 1:
                        complexity_level = "<span style='color: #4CAF50;'>Низька</span>"
                    elif complexity <= 3:
                        complexity_level = "<span style='color: #FFC107;'>Середня</span>"
                    else:
                        complexity_level = "<span style='color: #F44336;'>Висока</span>"
                    info_items.append(f"<li>Складність гри: {complexity_level}
({complexity}/5)</li>")
                if info_items:
                    st.markdown(f"<ul>{''.join(info_items)}</ul>",
unsafe_allow_html=True)
                st.markdown("</div>", unsafe_allow_html=True)
                # Відображення AI рекомендацій, якщо вони є
                if use_ai and "ai_recommendations" in result and
result["ai_recommendations"]:
                    with st.expander("👹 Рекомендації від ШІ", expanded=True):
                        if result["ai_recommendations"]["success"]:
                            st.markdown(f"""
                            <div style='background-color:
{COLOR_PALETTE["secondary_light"]}; padding: 20px;
                                 border-radius: 10px; margin-top: 10px; margin-bottom:
10px;'>
                                {result["ai_recommendations"]["recommendations"]}
                            </div>
                            """, unsafe_allow_html=True)
```

```
                        else:
                            st.warning("Не вдалося отримати рекомендації від ШІ: " +
result["ai_recommendations"]["error"])
        st.markdown("</div>", unsafe_allow_html=True)
def show_random_character():
    animated_title("🎲 Генератор випадкових персонажів")
    st.markdown(f"""
    <div style='background-color: {COLOR_PALETTE["secondary_light"]}; padding: 20px; border-
radius: 10px; margin-bottom: 20px;'>
        <p>Натисніть кнопку нижче для генерації випадкового персонажа D&D 5e. Якщо вам
сподобається результат, ви можете зберегти його в своєму профілі.</p>
    </div>
    """, unsafe_allow_html=True)
    # Анімація кубиків
    lottie_dice = load_lottie_animation()
    if lottie_dice:
        col1, col2, col3 = st.columns([1, 2, 1])
        with col2:
            st_lottie(lottie_dice, height=150, key="dice_animation")
    # Центрована кнопка з стилізацією
    col1, col2, col3 = st.columns([1, 2, 1])
    with col2:
        if st.button("🎲 Згенерувати випадкового персонажа", use_container_width=True):
            play_sound("click")
            with st.spinner("Генерація персонажа..."):
                time.sleep(0.8)  # Імітація затримки для кращого UX
                result = call_api("random-character")
            if result:
                play_sound("success")
                # Стилізована картка персонажа
                st.markdown(f"""
                <div style='background-color: {COLOR_PALETTE["secondary_light"]}; padding:
25px;
                     border-radius: 15px; margin-top: 20px; margin-bottom: 20px;
                     box-shadow: 0 4px 8px rgba(0,0,0,0.1);'>
                    <h3 style='color: {COLOR_PALETTE["primary_pink"]}; text-align: center;
                        margin-bottom: 20px;'>Згенерований персонаж</h3>
                """, unsafe_allow_html=True)
                col1, col2 = st.columns(2)
                with col1:
                    st.markdown(f"""
                    <div style='background-color: {COLOR_PALETTE["primary_teal"]}20; padding:
15px;
                        border-radius: 10px; margin-bottom: 15px;'>
                        <p><strong>Ім'я:</strong> {result['name']}</p>
                        <p><strong>Раса:</strong> {result['race']}</p>
                        <p><strong>Клас:</strong> {result['class']}</p>
                        <p><strong>Рівень:</strong> {result['level']}</p>
                        <p><strong>Передісторія:</strong> {result.get('background',
'Невідома')}</p>
                        <p><strong>Світогляд:</strong> {result.get('alignment',
'Невідомий')}</p>
                    </div>
                    """, unsafe_allow_html=True)
                with col2:
                    st.markdown(f"""
                    <div style='background-color: {COLOR_PALETTE["primary_pink"]}20; padding:
15px;
                        border-radius: 10px; margin-bottom: 15px;'>
                        <h4 style='color: {COLOR_PALETTE["primary_pink"]}; margin-bottom:
10px;'>Характеристики:</h4>
                        <div style='display: grid; grid-template-columns: 1fr 1fr; grid-gap:
10px;'>
                            <p><strong>Сила:</strong> {result['abilities']['str']}</p>
                            <p><strong>Спритність:</strong> {result['abilities']['dex']}</p>
```

```
                        <p><strong>Статура:</strong> {result['abilities']['con']}</p>
                        <p><strong>Інтелект:</strong> {result['abilities']['int']}</p>
                        <p><strong>Мудрість:</strong> {result['abilities']['wis']}</p>
                        <p><strong>Харизма:</strong> {result['abilities']['cha']}</p>
                    </div>
                </div>
                """, unsafe_allow_html=True)
            # Спорядження та здібності
            col1, col2 = st.columns(2)
            with col1:
                if "equipment" in result and result["equipment"]:
                    st.markdown(f"<h4 style='color:
{COLOR_PALETTE['primary_teal']};'>Спорядження:</h4>", unsafe_allow_html=True)
                    equipment_list = ""
                    for item in result["equipment"]:
                        equipment_list += f"<li>{item}</li>"
                    st.markdown(f"<ul>{equipment_list}</ul>", unsafe_allow_html=True)
            with col2:
                if "features" in result and result["features"]:
                    st.markdown(f"<h4 style='color:
{COLOR_PALETTE['primary_teal']};'>Здібності:</h4>", unsafe_allow_html=True)
                    features_list = ""
                    for feature in result["features"]:
                        features_list += f"<li>{feature}</li>"
                    st.markdown(f"<ul>{features_list}</ul>", unsafe_allow_html=True)
            st.markdown("</div>", unsafe_allow_html=True)
            # Кнопка для збереження
            col1, col2, col3 = st.columns([1, 2, 1])
            with col2:
                if st.button("💾 Зберегти персонажа", use_container_width=True):
                    play_sound("click")
                    if "logged_in" in st.session_state and st.session_state.logged_in and
st.session_state.role == "player":
                        with st.spinner("Збереження персонажа..."):
                            time.sleep(0.5)  # Імітація затримки для кращого UX
                            character_data = {
                                "name": result["name"],
                                "level": result["level"],
                                "race": result["race"],
                                "background": result.get("background", ""),
                                "str": result["abilities"]["str"],
                                "dex": result["abilities"]["dex"],
                                "con": result["abilities"]["con"],
                                "int": result["abilities"]["int"],
                                "wis": result["abilities"]["wis"],
                                "cha": result["abilities"]["cha"],
                                "classes_combination": result.get("class", "")
                            }
                            save_result = call_api("characters", method="POST",
data=character_data)
                            if save_result:
                                play_sound("success")
                                st.success("Персонажа успішно збережено!")
                    else:
                        play_sound("error")
                        st.warning("Для збереження персонажа необхідно увійти в систему
як гравець.")
def show_party_manager():
    animated_title("🎭 Керування партіями")
    # Перевіряємо авторизацію
    if not st.session_state.logged_in:
        st.warning("Для роботи з партіями необхідно увійти в систему")
        return
    # Спроба отримання списку партій
    with st.spinner("Завантаження партій..."):
```

```
        parties = call_api("parties")
    # Розділ для створення нової партії (тільки для DM)
    if st.session_state.role == "dm":
        with st.expander("➕ Створити нову партію", expanded=not parties):
            with st.form("create_party_form"):
                party_name = st.text_input("Назва партії")
                party_description = st.text_area("Опис партії")
                submit_button = st.form_submit_button("Створити партію")
                if submit_button:
                    play_sound("click")
                    if not party_name:
                        st.error("Необхідно вказати назву партії")
                    else:
                        with st.spinner("Створення партії..."):
                            time.sleep(0.5)
                            result = call_api("parties", method="POST", data={
                                "name": party_name,
                                "description": party_description
                            })
                            if result:
                                play_sound("success")
                                st.success("Партію успішно створено!")
                                time.sleep(1)
                                st.rerun()
    if not parties:
        if st.session_state.role == "dm":
            st.info("У вас ще немає створених партій. Створіть свою першу партію!")
        else:
            st.info("Ви ще не є членом жодної партії.")
        return
    # Список партій з вкладками
    st.markdown(f"<h3 style='color: {COLOR_PALETTE['primary_pink']};'>Ваші партії:</h3>",
unsafe_allow_html=True)
    # Створюємо вкладки для кожної партії
    party_tabs = st.tabs([party["name"] for party in parties])
    # Відображаємо кожну партію
    for i, tab in enumerate(party_tabs):
        with tab:
            party_id = parties[i]["id"]
            # Завантажуємо деталі партії
            with st.spinner("Завантаження деталей партії..."):
                party_details = call_api(f"parties/{party_id}")
            if not party_details:
                st.warning("Не вдалося завантажити деталі партії")
                continue
            # Інформація про партію
            st.markdown(f"""
            <div style='background-color: {COLOR_PALETTE["secondary_light"]}; padding: 20px;
                  border-radius: 10px; margin-bottom: 20px;'>
                <h3 style='color:
{COLOR_PALETTE["primary_pink"]};'>{party_details['name']}</h3>
                <p>{party_details.get('description', 'Немає опису')}</p>
                <p><strong>DM:</strong> {party_details.get('dm_username', 'Невідомо')}</p>
            </div>
            """, unsafe_allow_html=True)
            # Вкладки для учасників, чату та інших функцій
            member_tab, chat_tab, session_tab, combat_tab = st.tabs(["Учасники", "Чат",
"Сесії", "Бій"])
            with member_tab:
                st.markdown(f"<h4 style='color: {COLOR_PALETTE['primary_teal']};'>Учасники
партії:</h4>", unsafe_allow_html=True)
                # Відображаємо учасників
                for member in party_details.get('members', []):
                    role_badge = "🎲 DM" if member.get('role') == "dm" else "🧙 Гравець"
```

```
                    with st.expander(f"{member.get('username', 'Невідомий')}
({role_badge})"):
                        if member.get('character'):
                            char = member['character']
                            st.markdown(f"""
                            <div style='background-color: {COLOR_PALETTE["secondary_light"]};
padding: 10px;
                                 border-radius: 8px; margin-bottom: 10px;'>
                                <p><strong>Персонаж:</strong> {char.get('name', 'Немає')}</p>
                                <p><strong>Раса:</strong> {char.get('race', 'Невідома')}</p>
                                <p><strong>Рівень:</strong> {char.get('level', '1')}</p>
                                <p><strong>Класи:</strong> {char.get('classes_combination',
'Невідомо')}</p>
                            </div>
                            """, unsafe_allow_html=True)
                        else:
                            st.info("Цей гравець ще не вибрав персонажа для цієї партії")
                        # Якщо поточний користувач - DM, показуємо кнопку для виключення
гравця
                        if st.session_state.role == "dm" and st.session_state.user_id ==
party_details.get('dm_id'):
                            if member.get('user_id') != party_details.get('dm_id'):  # Не
можна видалити самого DM
                                if st.button(f"Видалити з партії",
key=f"remove_{member.get('user_id')}_{party_id}"):
                                    play_sound("click")
                                    with st.spinner("Видалення гравця..."):
                                        result =
call_api(f"parties/{party_id}/members/{member.get('user_id')}", method="DELETE")
                                        if result:
                                            play_sound("success")
                                            st.success("Гравця успішно видалено з партії")
                                            time.sleep(1)
                                            st.rerun()
                # Додавання нових гравців (тільки для DM)
                if st.session_state.role == "dm" and st.session_state.user_id ==
party_details.get('dm_id'):
                    st.markdown("<hr>", unsafe_allow_html=True)
                    st.markdown(f"<h4 style='color: {COLOR_PALETTE['primary_pink']};'>Додати
нового гравця:</h4>", unsafe_allow_html=True)
                    # Завантажуємо список доступних користувачів
                    with st.spinner("Завантаження списку гравців..."):
                        all_users = call_api("users")
                    if all_users:
                        # Фільтруємо користувачів, які ще не є членами партії
                        current_members = [member.get('user_id') for member in
party_details.get('members', [])]
                        available_users = [user for user in all_users if user.get('id') not
in current_members]
                        if available_users:
                            with st.form(f"add_member_form_{party_id}",
clear_on_submit=True):
                                user_to_add = st.selectbox(
                                    "Оберіть користувача:",
                                    options=available_users,
                                    format_func=lambda x: f"{x.get('username')}
({x.get('role')})"
                                )
                                submit_button = st.form_submit_button("Додати до партії")
                                if submit_button:
                                    play_sound("click")
                                    with st.spinner("Додавання гравця..."):
                                        result = call_api(f"parties/{party_id}/members",
method="POST", data={
                                            "user_id": user_to_add.get('id')
```

```
                                    })
                                    if result:
                                        play_sound("success")
                                        st.success("Гравця успішно додано до партії!")
                                        time.sleep(1)
                                        st.rerun()
                        else:
                            st.info("Немає доступних користувачів для додавання до партії.")
            with chat_tab:
                st.markdown(f"<h4 style='color: {COLOR_PALETTE['primary_teal']};'>Чат
партії:</h4>", unsafe_allow_html=True)
                # Завантажуємо повідомлення чату
                with st.spinner("Завантаження повідомлень..."):
                    chat_messages = call_api(f"parties/{party_id}/chat")
                if not chat_messages:
                    st.info("Немає повідомлень або виникла помилка при завантаженні")
                else:
                    # Створюємо контейнер для чату
                    st.markdown("<div class='chat-container'>", unsafe_allow_html=True)
                    for message in chat_messages:
                        # Визначаємо клас повідомлення в залежності від того, хто надіслав
                        message_class = "own" if message.get('is_own', False) else "other"
                        # Визначаємо іконку в залежності від ролі відправника
                        role_icon = "🎲" if message.get('sender_role') == "dm" else "🧙"
                        st.markdown(f"""
                        <div class='message {message_class}'>
                            <div class='sender-name'>{role_icon} {message.get('sender_name',
'Невідомий')}</div>
                            {message.get('message', '')}
                        </div>
                        """, unsafe_allow_html=True)
                    st.markdown("</div>", unsafe_allow_html=True)
                # Форма для надсилання повідомлень
                with st.form(f"chat_form_{party_id}", clear_on_submit=True):
                    chat_message = st.text_area("Ваше повідомлення:", height=100)
                    col1, col2, col3 = st.columns([1, 1, 1])
                    with col2:
                        if st.form_submit_button("Надіслати", use_container_width=True):
                            play_sound("click")
                            if chat_message.strip():
                                with st.spinner("Надсилання..."):
                                    result = call_api(f"parties/{party_id}/chat",
method="POST", data={"message": chat_message})
                                    if result:
                                        play_sound("success")
                                        st.rerun()  # Оновлюємо сторінку для відображення
нового повідомлення
            with session_tab:
                st.markdown(f"<h4 style='color: {COLOR_PALETTE['primary_teal']};'>Сесії
кампанії:</h4>", unsafe_allow_html=True)
                # Завантажуємо сесії кампанії
                with st.spinner("Завантаження сесій..."):
                    sessions = call_api(f"parties/{party_id}/sessions")
                # Додавання нової сесії
                with st.expander("➕ Запланувати нову сесію", expanded=not sessions):
                    with st.form(f"session_form_{party_id}"):
                        session_title = st.text_input("Назва сесії")
                        session_date = st.date_input("Дата")
                        session_duration = st.number_input("Тривалість (годин)", min_value=1,
max_value=12, value=4)
                        session_notes = st.text_area("Нотатки")
                        submit_button = st.form_submit_button("Створити сесію")
                        if submit_button:
                            play_sound("click")
                            if not session_title:
```

```
                            st.error("Необхідно вказати назву сесії")
                        else:
                            with st.spinner("Створення сесії..."):
                                result = call_api(f"parties/{party_id}/sessions",
method="POST", data={
                                    "title": session_title,
                                    "scheduled_date": str(session_date),
                                    "duration": session_duration,
                                    "notes": session_notes,
                                    "status": "planned"
                                })
                                if result:
                                    play_sound("success")
                                    st.success("Сесію успішно створено!")
                                    time.sleep(1)
                                    st.rerun()
            # Відображення сесій
            if not sessions:
                st.info("Немає запланованих сесій")
            else:
                # Групуємо сесії за статусом
                planned_sessions = [s for s in sessions if s.get('status') == 'planned']
                completed_sessions = [s for s in sessions if s.get('status') ==
'completed']
                # Відображаємо заплановані сесії
                if planned_sessions:
                    st.markdown(f"<h5 style='color:
{COLOR_PALETTE['primary_pink']};'>Заплановані сесії:</h5>", unsafe_allow_html=True)
                    for session_info in planned_sessions:
                        with st.container():
                            # Стилізована картка сесії
                            st.markdown(f"""
                            <div style='background-color:
{COLOR_PALETTE["secondary_light"]}; padding: 15px;
                                        border-radius: 10px; margin-bottom: 15px; border-left:
5px solid {COLOR_PALETTE["primary_pink"]};'>
                            <h4 style='color:
{COLOR_PALETTE["primary_pink"]};'>{session_info.get('title')}</h4>
                            <p><strong>Дата:</strong>
{session_info.get('scheduled_date')}</p>
                            <p><strong>Тривалість:</strong>
{session_info.get('duration')} годин</p>
                            <p><strong>Створив:</strong>
{session_info.get('creator_name')}</p>
                            </div>
                            """, unsafe_allow_html=True)
                            if session_info.get('notes'):
                                with st.expander("Нотатки"):
                                    st.write(session_info.get('notes'))
                            # Додаємо кнопки для керування сесією (тільки для DM)
                            if st.session_state.role == "dm" and st.session_state.user_id
== party_details.get('dm_id'):
                                col1, col2 = st.columns(2)
                                with col1:
                                    if st.button("✅ Завершити",
key=f"complete_session_{session_info.get('id')}"):
                                        play_sound("click")
                                        # TODO: Додати API для завершення сесії
                                        st.success("API для завершення сесії ще не
реалізовано")
                                with col2:
                                    if st.button("❌ Скасувати",
key=f"cancel_session_{session_info.get('id')}"):
                                        play_sound("click")
                                        # TODO: Додати API для скасування сесії
```

```
                                        st.success("API для скасування сесії ще не
реалізовано")
                    # Відображаємо завершені сесії
                    if completed_sessions:
                        st.markdown(f"<h5 style='color:
{COLOR_PALETTE['primary_teal']};'>Завершені сесії:</h5>", unsafe_allow_html=True)
                        for session_info in completed_sessions:
                            with st.expander(f"{session_info.get('title')}
({session_info.get('scheduled_date')})"):
                                st.markdown(f"""
                                <div style='background-color:
{COLOR_PALETTE["secondary_light"]}; padding: 15px;
                                    border-radius: 10px; margin-bottom: 15px;'>
                                <p><strong>Тривалість:</strong>
{session_info.get('duration')} годин</p>
                                <p><strong>Створив:</strong>
{session_info.get('creator_name')}</p>
                                </div>
                                """, unsafe_allow_html=True)
                                if session_info.get('notes'):
                                    st.write("**Нотатки:**")
                                    st.write(session_info.get('notes'))
            with combat_tab:
                st.markdown(f"<h4 style='color: {COLOR_PALETTE['primary_teal']};'>Бойові
сутички:</h4>", unsafe_allow_html=True)
                # Завантажуємо бойові сутички
                with st.spinner("Завантаження бойових сутичок..."):
                    combat_encounters = call_api(f"parties/{party_id}/combat")
                # Додавання нової бойової сутички (тільки для DM)
                if st.session_state.role == "dm" and st.session_state.user_id ==
party_details.get('dm_id'):
                    with st.expander("➕ Створити нову бойову сутичку", expanded=not
combat_encounters):
                        with st.form(f"combat_form_{party_id}"):
                            encounter_name = st.text_input("Назва сутички")
                            # Додавання учасників
                            st.markdown("### Додати учасників")
                            st.markdown("#### Монстри")
                            # Можливість додати монстрів з API
                            use_api_monsters = st.checkbox("Додати монстрів з D&D 5e API")
                            if use_api_monsters:
                                # Завантажуємо список монстрів з API
                                with st.spinner("Завантаження монстрів..."):
                                    monster_params = {"use_dnd_api": "true"}
                                    if st.session_state.get('monster_search'):
                                        monster_params["search"] =
st.session_state.monster_search
                                    monsters = call_api("monsters", params=monster_params)
                                if monsters:
                                    monster_search = st.text_input("Пошук монстрів",
key="monster_search")
                                    # Відображаємо монстрів у вигляді карток
                                    monster_cols = st.columns(3)
                                    for i, monster in enumerate(monsters[:9]):  # Обмежуємо
до 9 монстрів
                                        with monster_cols[i % 3]:
                                            st.markdown(f"""
                                            <div style='background-color:
{COLOR_PALETTE["secondary_light"]};
                                                padding: 10px; border-radius: 8px; margin-
bottom: 10px;'>
                                            <h5>{monster.get('name')}</h5>
                                            <p>CR: {monster.get('challenge_rating')}</p>
                                            <p>HP: {monster.get('hit_points')}</p>
                                            </div>
```

```
                                        """, unsafe_allow_html=True)
                                        st.checkbox(f"Додати",
key=f"add_monster_{monster.get('id')}")
                            # Додавання персонажів гравців
                            st.markdown("#### Персонажі гравців")
                            # Отримуємо персонажів гравців у партії
                            player_characters = []
                            for member in party_details.get('members', []):
                                if member.get('role') == 'player' and
member.get('character'):
                                    player_characters.append(member['character'])
                            # Відображаємо персонажів у вигляді карток
                            if player_characters:
                                player_cols = st.columns(len(player_characters))
                                for i, char in enumerate(player_characters):
                                    with player_cols[i]:
                                        st.markdown(f"""
                                        <div style='background-color:
{COLOR_PALETTE["primary_pink"]}20;
                                             padding: 10px; border-radius: 8px; margin-
bottom: 10px;'>
                                            <h5>{char.get('name')}</h5>
                                            <p>{char.get('classes_combination')}</p>
                                            <p>Lvl {char.get('level')}</p>
                                        </div>
                                        """, unsafe_allow_html=True)
                                        st.checkbox(f"Додати",
key=f"add_char_{char.get('id')}")
                            else:
                                st.info("У партії немає персонажів гравців")
                            # Кнопка для створення сутички
                            submit_button = st.form_submit_button("Створити бойову сутичку")
                            if submit_button:
                                play_sound("click")
                                if not encounter_name:
                                    st.error("Необхідно вказати назву сутички")
                                else:
                                    # Збираємо вибраних монстрів
                                    selected_monsters = []
                                    if use_api_monsters and monsters:
                                        for monster in monsters[:9]:
                                            if
st.session_state.get(f"add_monster_{monster.get('id')}"):
                                                selected_monsters.append({
                                                    "name": monster.get('name'),
                                                    "hp_max": monster.get('hit_points'),
                                                    "hp_current": monster.get('hit_points'),
                                                    "ac": monster.get('armor_class'),
                                                    "is_pc": False
                                                })
                                    # Збираємо вибраних персонажів
                                    selected_characters = []
                                    for char in player_characters:
                                        if
st.session_state.get(f"add_char_{char.get('id')}"):
                                            selected_characters.append({
                                                "name": char.get('name'),
                                                "character_id": char.get('id'),
                                                "is_pc": True
                                            })
                                    with st.spinner("Створення бойової сутички..."):
                                        result = call_api(f"parties/{party_id}/combat",
method="POST", data={
                                            "name": encounter_name,
```

```
                                        "participants": selected_monsters +
selected_characters
                                    })
                                    if result:
                                        play_sound("success")
                                        st.success("Бойову сутичку успішно створено!")
                                        time.sleep(1)
                                        st.rerun()
                # Відображення бойових сутичок
                if not combat_encounters:
                    st.info("Немає активних бойових сутичок")
                else:
                    for encounter in combat_encounters:
                        with st.expander(f"{encounter.get('name')} (Раунд
{encounter.get('round')}, {encounter.get('status')})"):
                            # Панель керування боєм (для DM)
                            if st.session_state.role == "dm" and st.session_state.user_id ==
party_details.get('dm_id'):
                                col1, col2, col3 = st.columns(3)
                                with col1:
                                    if encounter.get('status') == 'preparation':
                                        if st.button("▶️ Почати бій",
key=f"start_combat_{encounter.get('id')}"):
                                            play_sound("click")
                                            with st.spinner("Початок бою..."):
                                                result =
call_api(f"combat/{encounter.get('id')}/start", method="POST")
                                                if result:
                                                    play_sound("success")
                                                    st.success("Бій розпочато!")
                                                    time.sleep(1)
                                                    st.rerun()
                                with col2:
                                    if encounter.get('status') == 'active':
                                        if st.button("⏭️ Наступний хід",
key=f"next_turn_{encounter.get('id')}"):
                                            play_sound("click")
                                            with st.spinner("Перехід до наступного ходу..."):
                                                result =
call_api(f"combat/{encounter.get('id')}/next-turn", method="POST")
                                                if result:
                                                    play_sound("success")
                                                    st.success(f"Хід передано:
{result.get('active_participant', {}).get('name')}")
                                                    time.sleep(1)
                                                    st.rerun()
                                with col3:
                                    if encounter.get('status') == 'active':
                                        if st.button("⏹️ Завершити бій",
key=f"end_combat_{encounter.get('id')}"):
                                            play_sound("click")
                                            with st.spinner("Завершення бою..."):
                                                result =
call_api(f"combat/{encounter.get('id')}/end", method="POST")
                                                if result:
                                                    play_sound("success")
                                                    st.success("Бій завершено!")
                                                    time.sleep(1)
                                                    st.rerun()
                            # Учасники бою
                            st.markdown(f"<h5 style='color:
{COLOR_PALETTE['primary_teal']};'>Учасники:</h5>", unsafe_allow_html=True)
                            participants = encounter.get('participants', [])
                            if not participants:
                                st.info("Немає учасників у цій бойовій сутичці")
```

```
                        else:
                            for participant in participants:
                                # Визначаємо колір блоку (зелений для гравців, червоний
для монстрів)
                                block_color = "#4CAF5030" if participant.get('is_pc')
else "#F4433630"
                                border_color = "#4CAF50" if participant.get('is_pc') else
"#F44336"
                                # Додаємо індикатор активного учасника
                                active_indicator = ""
                                if participant.get('is_active'):
                                    active_indicator = """
                                    <span class='turn-indicator'>Зараз ходить</span>
                                    """
                                # Розраховуємо заповнення смуги HP
                                hp_max = participant.get('hp_max', 1)
                                hp_current = participant.get('hp_current', 0)
                                hp_percent = min(100, max(0, (hp_current / hp_max *
100)))
                                # Визначаємо колір HP
                                hp_color = "#4CAF50"  # Зелений
                                if hp_percent < 60:
                                    hp_color = "#FFC107"  # Жовтий
                                if hp_percent < 30:
                                    hp_color = "#F44336"  # Червоний
                                st.markdown(f"""
                                <div style='background-color: {block_color}; border-left:
5px solid {border_color};
                                      padding: 10px; border-radius: 5px; margin-bottom:
10px;'>
                                    <div style='display: flex; justify-content: space-
between; align-items: center;'>
                                        <h5>{participant.get('name')}</h5>
                                        {active_indicator}
                                    </div>
                                    <div style='display: flex; justify-content: space-
between; margin-top: 5px;'>
                                        <span>ІНІ: {participant.get('initiative',
0)}</span>
                                        <span>КД: {participant.get('ac', 0)}</span>
                                    </div>
                                    <div class='hp-bar'>
                                        <div class='hp-bar-fill' style='width:
{hp_percent}%; background-color: {hp_color};'></div>
                                    </div>
                                    <div class='hp-bar-text' style='text-align: right;'>
                                        ХП: {hp_current}/{hp_max}
                                    </div>
                                </div>
                                """, unsafe_allow_html=True)
                                # Кнопки для керування учасником (для DM)
                                if st.session_state.role == "dm" and
st.session_state.user_id == party_details.get('dm_id'):
                                    cols = st.columns(3)
                                    with cols[0]:
                                        damage = st.number_input("Шкода", min_value=0,
key=f"damage_{participant.get('id')}")
                                    with cols[1]:
                                        healing = st.number_input("Лікування",
min_value=0, key=f"healing_{participant.get('id')}")
                                    with cols[2]:
                                        if st.button("Застосувати",
key=f"apply_{participant.get('id')}"):
                                            play_sound("click")
```

```
                                                # TODO: Додати API для застосування
шкоди/лікування
                                                st.success("API для застосування
шкоди/лікування ще не реалізовано")
                                        # Додаємо поле для статусних ефектів
                                        status_effects = st.text_input("Статусні ефекти",

value=participant.get('status_effects', ''),

key=f"status_{participant.get('id')}")
                                        if status_effects !=
participant.get('status_effects', ''):
                                            if st.button("Оновити статус",
key=f"update_status_{participant.get('id')}"):
                                                play_sound("click")
                                                # TODO: Додати API для оновлення статусних
ефектів
                                                st.success("API для оновлення статусних
ефектів ще не реалізовано")
def show_spell_browser():
    animated_title("✨ Каталог заклинань")
    # Додаємо опцію для використання API
    use_dnd_api = st.checkbox("Використовувати D&D 5e API", value=True)
    # Фільтри для заклинань
    col1, col2, col3 = st.columns(3)
    with col1:
        spell_class = st.selectbox("Клас", ["", "Варвар", "Бард", "Жрець", "Друїд", "Воїн",
"Монах", "Паладин", "Рейнджер", "Розбійник", "Чародій", "Варлок", "Чарівник"])
    with col2:
        spell_level = st.selectbox("Рівень", ["", "0", "1", "2", "3", "4", "5", "6", "7",
"8", "9"])
    with col3:
        search_query = st.text_input("Пошук")
    # Створюємо параметри запиту
    params = {"use_dnd_api": str(use_dnd_api).lower()}
    if spell_class:
        params["class"] = spell_class
    if spell_level:
        params["level"] = spell_level
    if search_query:
        params["search"] = search_query
    # Завантажуємо заклинання
    with st.spinner("Завантаження заклинань..."):
        spells = call_api("spells", params=params)
    if not spells:
        st.warning("Не вдалося завантажити заклинання або вони відсутні за вказаними
критеріями")
        return
    # Відображаємо заклинання у вигляді карток
    st.markdown(f"<h3 style='color: {COLOR_PALETTE['primary_pink']};'>Знайдено {len(spells)}
заклинань</h3>", unsafe_allow_html=True)
    # Створюємо колонки для карток
    col1, col2 = st.columns(2)
    for i, spell in enumerate(spells):
        # Визначаємо колір для кожного рівня заклинання
        level_colors = {
            0: "#607D8B",  # Сірий для трюків
            1: "#4CAF50",  # Зелений для 1 рівня
            2: "#2196F3",  # Синій для 2 рівня
            3: "#673AB7",  # Фіолетовий для 3 рівня
            4: "#FF9800",  # Помаранчевий для 4 рівня
            5: "#F44336",  # Червоний для 5 рівня
            6: "#E91E63",  # Рожевий для 6 рівня
            7: "#9C27B0",  # Пурпурний для 7 рівня
            8: "#FF5722",  # Глибокий помаранчевий для 8 рівня
```

```
            9: "#FFEB3B"   # Жовтий для 9 рівня
        }
        spell_level = spell.get('level', 0)
        color = level_colors.get(spell_level, COLOR_PALETTE["primary_teal"])
        with col1 if i % 2 == 0 else col2:
            # Стилізована картка заклинання
            level_text = "Трюк" if spell_level == 0 else f"{spell_level} рівень"
            school_text = spell.get('school', '')
            with st.expander(f"{spell.get('name')} ({level_text}, {school_text})"):
                st.markdown(f"""
                <div style='background-color: {color}20; padding: 15px; border-radius: 8px;
margin-bottom: 15px; border-left: 5px solid {color};'>
                    <h4 style='color: {color};'>{spell.get('name')}</h4>
                    <p><strong>Рівень:</strong> {level_text}</p>
                    <p><strong>Школа:</strong> {school_text}</p>
                    <p><strong>Час накладання:</strong> {spell.get('casting_time')}</p>
                    <p><strong>Дистанція:</strong> {spell.get('range')}</p>
                    <p><strong>Компоненти:</strong> {spell.get('components')}</p>
                    <p><strong>Тривалість:</strong> {spell.get('duration')}</p>
                    <p><strong>Класи:</strong> {spell.get('classes')}</p>
                </div>
                <h5>Опис:</h5>
                <p>{spell.get('description', 'Опис відсутній')}</p>
                """, unsafe_allow_html=True)
def show_monster_browser():
    animated_title("👹 Каталог монстрів")
    # Додаємо опцію для використання API
    use_dnd_api = st.checkbox("Використовувати D&D 5e API", value=True)
    # Фільтри для монстрів
    col1, col2, col3 = st.columns(3)
    with col1:
        challenge_rating = st.selectbox("Рівень небезпеки", ["", "0", "0.25", "0.5", "1",
"2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17",
"18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30"])
    with col2:
        search_query = st.text_input("Пошук")
    # Створюємо параметри запиту
    params = {"use_dnd_api": str(use_dnd_api).lower()}
    if challenge_rating:
        params["cr"] = challenge_rating
    if search_query:
        params["search"] = search_query
    # Завантажуємо монстрів
    with st.spinner("Завантаження монстрів..."):
        monsters = call_api("monsters", params=params)
    if not monsters:
        st.warning("Не вдалося завантажити монстрів або вони відсутні за вказаними
критеріями")
        return
    # Відображаємо монстрів у вигляді карток
    st.markdown(f"<h3 style='color: {COLOR_PALETTE['primary_pink']};'>Знайдено
{len(monsters)} монстрів</h3>", unsafe_allow_html=True)
    # Створюємо колонки для карток
    monster_columns = st.columns(3)
    for i, monster in enumerate(monsters):
        # Визначаємо колір в залежності від CR
        cr = monster.get('challenge_rating', 0)
        if cr <= 1:
            color = "#4CAF50"  # Зелений для низького CR
        elif cr <= 5:
            color = "#FFC107"  # Жовтий для середнього CR
        elif cr <= 10:
            color = "#FF9800"  # Помаранчевий для високого CR
        else:
            color = "#F44336"  # Червоний для дуже високого CR
```

```
        with monster_columns[i % 3]:
            # Стилізована картка монстра
            monster_type = monster.get('type', 'невідомий')
            monster_size = monster.get('size', '')
            with st.expander(f"{monster.get('name')}, CR {cr}"):
                st.markdown(f"""
                <div style='background-color: {color}20; padding: 15px; border-radius: 8px;
margin-bottom: 15px; border-left: 5px solid {color};'>
                    <h4 style='color: {color};'>{monster.get('name')}</h4>
                    <p><strong>Тип:</strong> {monster_size} {monster_type}</p>
                    <p><strong>Рівень небезпеки:</strong> {cr}</p>
                    <p><strong>Клас броні:</strong> {monster.get('armor_class', 0)}</p>
                    <p><strong>Хіт-поінти:</strong> {monster.get('hit_points', 0)}
({monster.get('hit_dice', '')})</p>
                    <p><strong>Швидкість:</strong> {monster.get('speed', {}).get('walk', '30
ft.')}</p>
                </div>
                <h5>Характеристики:</h5>
                <div style='display: grid; grid-template-columns: 1fr 1fr 1fr; grid-gap:
10px;'>
                    <p><strong>СИЛ:</strong> {monster.get('abilities', {}).get('str',
10)}</p>
                    <p><strong>СПР:</strong> {monster.get('abilities', {}).get('dex',
10)}</p>
                    <p><strong>СТА:</strong> {monster.get('abilities', {}).get('con',
10)}</p>
                    <p><strong>ІНТ:</strong> {monster.get('abilities', {}).get('int',
10)}</p>
                    <p><strong>МДР:</strong> {monster.get('abilities', {}).get('wis',
10)}</p>
                    <p><strong>ХАР:</strong> {monster.get('abilities', {}).get('cha',
10)}</p>
                </div>
                """, unsafe_allow_html=True)
                # Додаємо кнопку для перегляду деталей
                if st.button("Детальна інформація", key=f"details_{monster.get('id')}"):
                    with st.spinner("Завантаження деталей..."):
                        monster_details = call_api(f"monsters/{monster.get('id')}",
params={"use_dnd_api": str(use_dnd_api).lower()})
                        if monster_details:
                            st.markdown("### Дії")
                            for action in monster_details.get('actions', []):
                                st.markdown(f"**{action.get('name')}:**
{action.get('description')}")
                            if monster_details.get('special_abilities'):
                                st.markdown("### Спеціальні здібності")
                                for ability in monster_details.get('special_abilities', []):
                                    st.markdown(f"**{ability.get('name')}:**
{ability.get('description')}")
                            if monster_details.get('legendary_actions'):
                                st.markdown("### Легендарні дії")
                                for action in monster_details.get('legendary_actions', []):
                                    st.markdown(f"**{action.get('name')}:**
{action.get('description')}")
def main():
    # Завантажуємо стилі
    load_css()
    # Бічна панель з меню
    with st.sidebar:
        st.image("Logo.png", width=200)  # Приклад зображення логотипу
        # Вітання для авторизованого користувача
        if st.session_state.logged_in:
            st.markdown(f"### Вітаємо, {st.session_state.username}!")
            # Відображення ролі
            role_display = "Dungeon Master" if st.session_state.role == "dm" else "Гравець"
```

```
            st.markdown(f"**Роль:** {role_display}")
            # Кнопка виходу
            if st.button("Вийти", use_container_width=True):
                play_sound("click")
                with st.spinner("Виконуємо вихід..."):
                    result = call_api("logout", method="POST")
                    st.session_state.logged_in = False
                    st.session_state.user_id = None
                    st.session_state.username = None
                    st.session_state.role = "player"
                    st.rerun()
        # Меню
        st.markdown("### Меню")
        # Різне меню для авторизованих і неавторизованих користувачів
        if st.session_state.logged_in:
            menu_options = [
                "Класи D&D",
                "Мультикласові білди",
                "Конструктор персонажа",
                "Перевірка білду",
                "Генератор персонажів",
                "Керування партіями",
                "Каталог заклинань",
                "Каталог монстрів"
            ]
            menu_option = st.radio(
                "Оберіть розділ:",
                menu_options,
                label_visibility="collapsed"
            )
        else:
            menu_option = "Вхід"
    # Основний контент
    if not st.session_state.logged_in:
        show_login_page()
    else:
        if menu_option == "Класи D&D":
            show_classes()
        elif menu_option == "Мультикласові білди":
            show_multiclass_builds()
        elif menu_option == "Конструктор персонажа":
            show_character_builder()
        elif menu_option == "Перевірка білду":
            show_build_checker()
        elif menu_option == "Генератор персонажів":
            show_random_character()
        elif menu_option == "Керування партіями":
            show_party_manager()
        elif menu_option == "Каталог заклинань":
            show_spell_browser()
        elif menu_option == "Каталог монстрів":
            show_monster_browser()
if __name__ == "__main__":
    main()
```

server.py

```
from flask import Flask, request, jsonify, g, session
import sqlite3
import os
import json
import requests
```

```
import time
from datetime import datetime, timedelta
from werkzeug.security import generate_password_hash, check_password_hash
from functools import wraps
from flask_cors import CORS
import logging
# Налаштування логування
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
logger = logging.getLogger('dnd_app')
# Створюємо Flask додаток
app = Flask(__name__)
app.secret_key = os.urandom(24)
# Дозволяємо CORS для всіх маршрутів
CORS(app, supports_credentials=True)
# Налаштування сесій
app.config['SESSION_TYPE'] = 'filesystem'
app.config['SESSION_PERMANENT'] = True
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(days=7)
DATABASE = 'dnd_app.db'
# Ініціалізація API клієнта для D&D 5e API
DND_API_BASE_URL = "https://www.dnd5eapi.co/api"
def fetch_from_dnd_api(endpoint):
    """Отримання даних з D&D 5e API"""
    response = requests.get(f"{DND_API_BASE_URL}/{endpoint}")
    if response.status_code == 200:
        return response.json()
    logger.error(f"Помилка при отриманні даних з API: {response.status_code} -
{response.text}")
    return None
# Функції для роботи з базою даних
def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sqlite3.connect(DATABASE)
        db.row_factory = sqlite3.Row
    return db
@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()
def init_db():
    with app.app_context():
        db = get_db()
        # Створюємо всі необхідні таблиці
        db.executescript('''
            -- Таблиця для учасників бою
            CREATE TABLE IF NOT EXISTS combat_participants (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                encounter_id INTEGER NOT NULL,
                name TEXT NOT NULL,
                initiative INTEGER,
                hp_max INTEGER,
                hp_current INTEGER,
                ac INTEGER,
                is_pc BOOLEAN,
                character_id INTEGER,
                turn_order INTEGER,
                is_active BOOLEAN DEFAULT 0,
                status_effects TEXT,
                FOREIGN KEY (encounter_id) REFERENCES combat_encounters (id),
                FOREIGN KEY (character_id) REFERENCES characters (id)
            );
        ''')
```

```
        # Заповнюємо базу даних початковими даними
        logger.info("Заповнення бази даних даними про класи...")
        # Використовуємо локальні дані
        class_data = get_basic_classes_data()
        for dnd_class in class_data:
            # Перевіряємо, чи клас уже існує
            existing = db.execute('SELECT id FROM classes WHERE name = ?',
(dnd_class['name'],)).fetchone()
            if not existing:
                db.execute(
                    'INSERT INTO classes (name, description, hit_die, primary_ability,
saving_throw_proficiencies) VALUES (?, ?, ?, ?, ?)',
                    (dnd_class['name'], dnd_class['description'], dnd_class['hit_die'],
                     dnd_class['primary_ability'], dnd_class['saving_throw_proficiencies'])
                )
        # Додаємо мультикласові білди
        logger.info("Додавання базових мультикласових білдів...")
        builds = get_basic_multiclass_builds()
        for build in builds:
            # Перевіряємо, чи білд уже існує
            existing = db.execute('SELECT id FROM multiclass_builds WHERE name = ?',
(build['name'],)).fetchone()
            if not existing:
                db.execute(
                    'INSERT INTO multiclass_builds (name, description, classes_combination,
strengths, weaknesses, is_optimized) VALUES (?, ?, ?, ?, ?, ?)',
                    (build['name'], build['description'], build['classes_combination'],
                     build['strengths'], build['weaknesses'], build['is_optimized'])
                )
        # Додаємо базові заклинання з D&D 5e API
        logger.info("Додавання базових заклинань з API...")
        add_basic_spells_from_api(db)
        db.commit()
        logger.info("База даних успішно ініціалізована.")
def add_basic_spells_from_api(db):
    """Додає базові заклинання з D&D 5e API"""
    try:
        # Отримуємо список заклинань
        spells_data = fetch_from_dnd_api('spells')
        if not spells_data or 'results' not in spells_data:
            logger.error("Не вдалося отримати список заклинань з API")
            return
        # Обмежимося першими 20 заклинаннями для прикладу
        spell_count = 0
        for spell_info in spells_data['results'][:20]:
            spell_data = fetch_from_dnd_api(f"spells/{spell_info['index']}")
            if not spell_data:
                continue
            # Перевіряємо, чи заклинання вже існує
            existing = db.execute('SELECT id FROM spells WHERE name = ?',
(spell_data['name'],)).fetchone()
            if existing:
                continue
            # Підготовка даних
            spell_level = spell_data.get('level', 0)
            spell_school = spell_data.get('school', {}).get('name', 'Unknown')
            casting_time = spell_data.get('casting_time', 'Unknown')
            spell_range = spell_data.get('range', 'Unknown')
            components = ', '.join(spell_data.get('components', []))
            duration = spell_data.get('duration', 'Unknown')
            description = spell_data.get('desc', [''])[0]
            classes = ', '.join([cls.get('name', '') for cls in spell_data.get('classes',
[])])
            # Додавання заклинання до бази даних
            db.execute(
```

```
                '''INSERT INTO spells
                   (name, level, school, casting_time, range, components, duration,
description, classes)
                   VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)''',
                (spell_data['name'], spell_level, spell_school, casting_time, spell_range,
                components, duration, description, classes)
            )
            spell_count += 1
            # Невелика пауза, щоб не перевантажувати API
            time.sleep(0.1)
        logger.info(f"Додано {spell_count} заклинань з API")
    except Exception as e:
        logger.error(f"Помилка при додаванні заклинань: {str(e)}")
# Допоміжні функції для отримання даних про класи та білди
def determine_primary_ability(class_name, saving_throws):
    """Визначає основну характеристику класу на основі його назви та рятувальних кидків"""
    class_abilities = {
        'Варвар': 'Сила',
        'Barbarian': 'Strength',
        'Бард': 'Харизма',
        'Bard': 'Charisma',
        'Жрець': 'Мудрість',
        'Cleric': 'Wisdom',
        'Друїд': 'Мудрість',
        'Druid': 'Wisdom',
        'Воїн': 'Сила або Спритність',
        'Fighter': 'Strength or Dexterity',
        'Монах': 'Спритність і Мудрість',
        'Monk': 'Dexterity and Wisdom',
        'Паладин': 'Сила і Харизма',
        'Paladin': 'Strength and Charisma',
        'Рейнджер': 'Спритність і Мудрість',
        'Ranger': 'Dexterity and Wisdom',
        'Розбійник': 'Спритність',
        'Rogue': 'Dexterity',
        'Чародій': 'Харизма',
        'Sorcerer': 'Charisma',
        'Варлок': 'Харизма',
        'Warlock': 'Charisma',
        'Чарівник': 'Інтелект',
        'Wizard': 'Intelligence'
    }
    return class_abilities.get(class_name, 'Невідома')
def get_class_description(class_name):
    """Отримує опис класу з локального словника"""
    # Базові описи класів
    class_descriptions = {
        'Варвар': 'Дикий воїн, який черпає лють із внутрішнього резервуару первісної люті.',
        'Barbarian': 'A fierce warrior who draws rage from an inner reservoir of primal
fury.',
        'Бард': 'Натхненний майстер, який використовує своє мистецтво для створення магії.',
        'Bard': 'An inspiring magician whose power echoes the music of creation.',
        'Жрець': 'Чемпіон богів, який володіє божественною магією.',
        'Cleric': 'A champion of the gods, wielding divine magic in service of a higher
power.',
        'Друїд': 'Жрець Старої Віри, який володіє силами природи.',
        'Druid': 'A priest of the Old Faith, wielding the powers of nature.',
        'Воїн': 'Майстер бою, неперевершений у володінні зброєю та обладунками.',
        'Fighter': 'A master of combat, skilled with a variety of weapons and armor.',
        'Монах': 'Майстер бойових мистецтв, який вдосконалює своє тіло для досягнення
досконалості.',
        'Monk': 'A master of martial arts, harnessing the power of the body in pursuit of
perfection.',
        'Паладин': 'Святий воїн, що пов’язаний священною клятвою.',
        'Paladin': 'A holy warrior bound to a sacred oath.',
```

```
        'Рейнджер': 'Воїн дикої місцевості, який спеціалізується на полюванні на ворогів
природи.',
        'Ranger': 'A warrior of the wilderness, specializing in hunting the monsters that
threaten civilization.',
        'Розбійник': 'Шахрай, який використовує спритність та хитрість для подолання
перешкод.',
        'Rogue': 'A scoundrel who uses stealth and trickery to overcome obstacles and
enemies.',
        'Чародій': 'Заклинатель, який використовує вроджену магію.',
        'Sorcerer': 'A spellcaster who draws on inherent magic from a gift or bloodline.',
        'Варлок': 'Заклинатель, який уклав договір з потойбічною сутністю.',
        'Warlock': 'A wielder of magic that is derived from a bargain with an extraplanar
entity.',
        'Чарівник': 'Учений магії, здатний маніпулювати структурою реальності.',
        'Wizard': 'A scholarly magic-user capable of manipulating the structures of
reality.'
    }
    return class_descriptions.get(class_name, f"Опис класу {class_name}")
def get_basic_classes_data():
    """Повертає базові дані про класи D&D 5e"""
    return [
        {

            "is_optimized": False
        },
        {
            "name": "Арканічний трікстер",
            "description": "Поєднання Розбійника та Чарівника для універсального
персонажа.",
            "classes_combination": "Розбійник 3 / Чарівник 17",
            "strengths": "Раптовий удар Розбійника з магією Чарівника. Висока
універсальність як у бою, так і поза ним.",
            "weaknesses": "Повільний розвиток високорівневих здібностей, залежність від
багатьох характеристик.",
            "is_optimized": False
        }
    ]
def get_basic_races():
    """Повертає базові дані про раси D&D 5e"""
    return [
        {
            "id": "human",
            "name": "Human",
            "name_uk": "Людина",
            "speed": 30,
            "ability_bonuses": [
                {"ability": "Strength", "bonus": 1},
                {"ability": "Dexterity", "bonus": 1},
                {"ability": "Constitution", "bonus": 1},
                {"ability": "Intelligence", "bonus": 1},
                {"ability": "Wisdom", "bonus": 1},
                {"ability": "Charisma", "bonus": 1}
            ],
            "alignment": "Humans tend toward no particular alignment. The best and the worst
are found among them.",
            "size": "Medium",
            "languages": "Common",
            "traits": [
                {"name": "Versatility", "description": "Humans are adaptable and diverse in
their abilities."}
            ]
        },
        {
            "id": "dwarf",
            "name": "Dwarf",
```

```
            "name_uk": "Дворф",
            "speed": 25,
            "ability_bonuses": [
                {"ability": "Constitution", "bonus": 2}
            ],
            "alignment": "Dwarves tend toward law and good.",
            "size": "Medium",
            "languages": "Common, Dwarvish",
            "traits": [
                {"name": "Darkvision", "description": "Accustomed to life underground, you
have superior vision in dark and dim conditions."},
                {"name": "Dwarven Resilience", "description": "You have advantage on saving
throws against poison, and you have resistance against poison damage."},
                {"name": "Dwarven Combat Training", "description": "You have proficiency
with the battleaxe, handaxe, light hammer, and warhammer."}
            ]
        },
        {
            "id": "elf",
            "name": "Elf",
            "name_uk": "Ельф",
            "speed": 30,
            "ability_bonuses": [
                {"ability": "Dexterity", "bonus": 2}
            ],
            "alignment": "Elves love freedom, variety, and self-expression.",
            "size": "Medium",
            "languages": "Common, Elvish",
            "traits": [
                {"name": "Darkvision", "description": "Accustomed to twilit forests and the
night sky, you have superior vision in dark and dim conditions."},
                {"name": "Keen Senses", "description": "You have proficiency in the
Perception skill."},
                {"name": "Fey Ancestry", "description": "You have advantage on saving throws
against being charmed, and magic can't put you to sleep."},
                {"name": "Trance", "description": "Elves don't need to sleep. Instead, they
meditate deeply for 4 hours a day."}
            ]
        }
    ]
def generate_basic_random_character():
    """Створює базового випадкового персонажа"""
    import random
    # Базові дані
    races = ['Людина', 'Ельф', 'Дворф', 'Напівельф', 'Гном', 'Драконороджений', 'Напіворк',
'Тифлінг']
    classes = ['Варвар', 'Бард', 'Жрець', 'Друїд', 'Воїн', 'Монах', 'Паладин', 'Рейнджер',
'Розбійник', 'Чародій', 'Варлок', 'Чарівник']
    backgrounds = ['Благородний', 'Злочинець', 'Відлюдник', 'Моряк', 'Мудрець', 'Солдат',
'Артист', 'Гільдійський ремісник']
    alignments = ['Законний добрий', 'Нейтральний добрий', 'Хаотичний добрий', 'Законний
нейтральний', 'Нейтральний', 'Хаотичний нейтральний', 'Законний злий', 'Нейтральний злий',
'Хаотичний злий']
    # Генеруємо випадкові характеристики
    abilities = {
        'str': roll_ability(),
        'dex': roll_ability(),
        'con': roll_ability(),
        'int': roll_ability(),
        'wis': roll_ability(),
        'cha': roll_ability()
    }
    # Випадковий клас і раса
    char_class = random.choice(classes)
    race = random.choice(races)
```

```
    # Розраховуємо хіт-поінти на основі класу
    hit_die = {
        'Варвар': 12,
        'Воїн': 10,
        'Паладин': 10,
        'Рейнджер': 10,
        'Бард': 8,
        'Жрець': 8,
        'Друїд': 8,
        'Монах': 8,
        'Розбійник': 8,
        'Варлок': 8,
        'Чародій': 6,
        'Чарівник': 6
    }
    constitution_mod = (abilities['con'] - 10) // 2
    hit_points = hit_die.get(char_class, 8) + constitution_mod
    # Складаємо випадкового персонажа
    return {
        'name': generate_random_name(race),
        'race': race,
        'class': char_class,
        'level': 1,
        'background': random.choice(backgrounds),
        'alignment': random.choice(alignments),
        'abilities': abilities,
        'hit_points': max(1, hit_points),  # Мінімум 1 хіт-поінт
        'proficiency_bonus': 2,  # Для 1 рівня
        'speed': 30,  # Стандартна швидкість
        'equipment': generate_random_equipment(char_class),
        'features': []
    }
def roll_ability():
    """Кидає 4d6, відкидає найнижчий результат і повертає суму"""
    import random
    rolls = [random.randint(1, 6) for _ in range(4)]
    return sum(sorted(rolls)[1:])
def generate_random_name(race):
    """Генерує випадкове ім'я на основі раси"""
    import random
    human_names = ['Артем', 'Богдан', 'Василь', 'Галина', 'Дарина', 'Євген', 'Жанна',
'Зоряна', 'Ігор', 'Катерина']
    elf_names = ['Елронд', 'Аеріс', 'Леголас', 'Галадріель', 'Тауріель', 'Вандіель',
'Аранніс', 'Фелоріел']
    dwarf_names = ['Торін', 'Гімлі', 'Балін', 'Двалін', 'Даїн', 'Фунгін', 'Грімдур',
'Тордек']
    if race == 'Людина':
        return random.choice(human_names)
    elif race in ['Ельф', 'Напівельф']:
        return random.choice(elf_names)
    elif race == 'Дворф':
        return random.choice(dwarf_names)
    else:
        # Для інших рас використовуємо комбінацію
        all_names = human_names + elf_names + dwarf_names
        return random.choice(all_names)
def generate_random_equipment(char_class):
    """Генерує базове спорядження для класу"""
    import random
    # Базове спорядження для кожного класу
    equipment = {
        'Варвар': ['Сокира', 'Шкіряний обладунок', 'Рюкзак', 'Спальний мішок', 'Раціон на 5
днів'],
        'Бард': ['Короткий меч', 'Лютня', 'Шкіряний обладунок', 'Рюкзак', 'Набір
мандрівника'],
```

```
        'Жрець': ['Булава', 'Кольчуга', 'Святий символ', 'Щит', 'Набір цілителя'],
        'Друїд': ['Дерев\'яний посох', 'Шкіряний обладунок', 'Фокус заклинань', 'Набір
мандрівника'],
        'Воїн': ['Довгий меч', 'Щит', 'Кольчуга', 'Короткий лук', 'Стріли (20)', 'Рюкзак'],
        'Монах': ['Короткий меч', 'Дротики (10)', 'Рюкзак', 'Набір мандрівника'],
        'Паладин': ['Довгий меч', 'Щит', 'Кольчуга', 'Святий символ', 'Набір цілителя'],
        'Рейнджер': ['Довгий лук', 'Стріли (20)', 'Короткий меч', 'Шкіряний обладунок',
'Набір мандрівника'],
        'Розбійник': ['Короткий меч', 'Короткий лук', 'Стріли (20)', 'Шкіряний обладунок',
'Набір злодія'],
        'Чародій': ['Кинджал', 'Компонентна сумка', 'Рюкзак', 'Набір мандрівника'],
        'Варлок': ['Кинджал', 'Шкіряний обладунок', 'Фокус заклинань', 'Рюкзак', 'Книга
знань'],
        'Чарівник': ['Кинджал', 'Компонентна сумка', 'Книга заклинань', 'Рюкзак', 'Набір
вченого']
    }
    # Додаємо базове спорядження для класу
    result = equipment.get(char_class, ['Кинджал', 'Рюкзак', 'Раціон на 5 днів', 'Набір
мандрівника'])
    # Додаємо випадкові предмети
    additional_items = ['Мотузка (50 футів)', 'Кремінь та кресало', 'Ліхтар', 'Олія
(фляга)', 'Сокира', 'Ніж', 'Точильний камінь', 'Плащ', 'Ковдра', 'Їжа (1 день)']
    # Додаємо 1-3 випадкових предмети
    for _ in range(random.randint(1, 3)):
        item = random.choice(additional_items)
        if item not in result:
            result.append(item)
    return result
def translate_class_name_to_ukrainian(name):
    """Перекладає назву класу з англійської на українську"""
    class_translations = {
        'Barbarian': 'Варвар',
        'Bard': 'Бард',
        'Cleric': 'Жрець',
        'Druid': 'Друїд',
        'Fighter': 'Воїн',
        'Monk': 'Монах',
        'Paladin': 'Паладин',
        'Ranger': 'Рейнджер',
        'Rogue': 'Розбійник',
        'Sorcerer': 'Чародій',
        'Warlock': 'Варлок',
        'Wizard': 'Чарівник'
    }
    return class_translations.get(name, name)
def translate_race_name_to_ukrainian(name):
    """Перекладає назву раси з англійської на українську"""
    race_translations = {
        'Human': 'Людина',
        'Dwarf': 'Дворф',
        'Elf': 'Ельф',
        'Halfling': 'Півельф',
        'Dragonborn': 'Драконороджений',
        'Gnome': 'Гном',
        'Half-Elf': 'Напівельф',
        'Half-Orc': 'Напіворк',
        'Tiefling': 'Тифлінг'
    }
    return race_translations.get(name, name)
# Декоратор для перевірки авторизації
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        # Перевіряємо наявність користувача в сесії
        if 'user_id' not in session:
```

```
            # Перевіряємо заголовок авторизації
            auth_header = request.headers.get('X-User-ID')
            if auth_header:
                # Перевіряємо користувача в базі
                db = get_db()
                user = db.execute('SELECT * FROM users WHERE id = ?',
(auth_header,)).fetchone()
                if user:
                    # Встановлюємо дані сесії
                    session['user_id'] = user['id']
                    session['role'] = user['role']
                    return f(*args, **kwargs)
            return jsonify({'error': 'Необхідна авторизація'}), 401
        return f(*args, **kwargs)
    return decorated_function
# Декоратор для перевірки ролі DM
def dm_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        # Перевіряємо наявність користувача в сесії
        if 'user_id' not in session:
            return jsonify({'error': 'Необхідна авторизація'}), 401
        # Перевіряємо, чи користувач є DM
        if session.get('role') != 'dm':
            return jsonify({'error': 'Ця функція доступна тільки Dungeon Master\'у'}), 403
        return f(*args, **kwargs)
    return decorated_function
# Декоратор для перевірки ролі гравця
def player_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        # Перевіряємо наявність користувача в сесії
        if 'user_id' not in session:
            return jsonify({'error': 'Необхідна авторизація'}), 401
        # Перевіряємо, чи користувач є гравцем
        if session.get('role') != 'player':
            return jsonify({'error': 'Ця функція доступна тільки гравцям'}), 403
        return f(*args, **kwargs)
    return decorated_function
# Маршрути API
# Реєстрація та авторизація
@app.route('/api/register', methods=['POST'])
def register():
    data = request.get_json()
    if not data or not data.get('username') or not data.get('password'):
        return jsonify({'error': 'Необхідно вказати логін та пароль'}), 400
    # Додаємо роль користувача
    role = data.get('role', 'player')
    if role not in ['player', 'dm']:
        return jsonify({'error': 'Невірно вказана роль. Доступні ролі: player, dm'}), 400
    db = get_db()
    user = db.execute('SELECT id FROM users WHERE username = ?',
(data['username'],)).fetchone()
    if user:
        return jsonify({'error': 'Користувач із таким іменем вже існує'}), 400
    db.execute(
        'INSERT INTO users (username, password, role) VALUES (?, ?, ?)',
        (data['username'], generate_password_hash(data['password']), role)
    )
    db.commit()
    return jsonify({'message': 'Користувача успішно зареєстровано'}), 201
@app.route('/api/login', methods=['POST'])
def login():
    data = request.get_json()
    if not data or not data.get('username') or not data.get('password'):
```

```
        return jsonify({'error': 'Необхідно вказати логін та пароль'}), 400
    db = get_db()
    user = db.execute('SELECT id, username, password, role FROM users WHERE username = ?',
(data['username'],)).fetchone()
    if not user or not check_password_hash(user['password'], data['password']):
        return jsonify({'error': 'Невірний логін або пароль'}), 401
    # Зберігаємо дані користувача в сесії
    session['user_id'] = user['id']
    session['role'] = user['role']
    return jsonify({
        'message': 'Успішна авторизація',
        'user_id': user['id'],
        'username': user['username'],
        'role': user['role']
    }), 200
@app.route('/api/logout', methods=['POST'])
def logout():
    session.pop('user_id', None)
    session.pop('role', None)
    return jsonify({'message': 'Вихід виконано успішно'}), 200
# API для класів
@app.route('/api/classes', methods=['GET'])
def get_classes():
    # Отримуємо параметри для фільтрації
    use_dnd_api = request.args.get('use_dnd_api', 'false').lower() == 'true'
    if use_dnd_api:
        # Отримуємо класи з D&D 5e API
        classes_data = fetch_from_dnd_api('classes')
        if classes_data and 'results' in classes_data:
            result = []
            for cls_info in classes_data['results']:
                # Отримуємо деталі класу
                cls_data = fetch_from_dnd_api(f"classes/{cls_info['index']}")
                if cls_data:
                    # Формуємо опис
                    description = get_class_description(cls_data['name'])
                    # Визначаємо основну характеристику
                    primary_ability = determine_primary_ability(cls_data['name'],
                                                                ', '.join([p.get('name', '')
for p in cls_data.get('proficiency_choices', [])]))
                    # Отримуємо рятувальні кидки
                    saving_throws = ', '.join([p.get('name', '').replace('Saving Throw: ',
'')
                                               for p in cls_data.get('saving_throws', [])])
                    # Додаємо клас до результату
                    result.append({
                        'id': cls_info['index'],
                        'name': translate_class_name_to_ukrainian(cls_data['name']),
                        'description': description,
                        'hit_die': cls_data.get('hit_die', 8),
                        'primary_ability': primary_ability,
                        'saving_throw_proficiencies': saving_throws
                    })
            return jsonify(result)
    # Якщо не використовуємо API або виникла помилка, використовуємо локальну базу даних
    db = get_db()
    classes = db.execute('SELECT * FROM classes').fetchall()
    result = []
    for cls in classes:
        result.append({
            'id': cls['id'],
            'name': cls['name'],
            'description': cls['description'],
            'hit_die': cls['hit_die'],
            'primary_ability': cls['primary_ability'],
```

```
            'saving_throw_proficiencies': cls['saving_throw_proficiencies']
        })
    return jsonify(result)
@app.route('/api/classes/<class_id>', methods=['GET'])
def get_class(class_id):
    # Перевіряємо, чи потрібно використовувати API
    use_dnd_api = request.args.get('use_dnd_api', 'false').lower() == 'true'
    if use_dnd_api and not class_id.isdigit():
        # Отримуємо дані про клас з D&D 5e API
        cls_data = fetch_from_dnd_api(f"classes/{class_id}")
        if cls_data:
            # Формуємо опис
            description = get_class_description(cls_data['name'])
            # Визначаємо основну характеристику
            primary_ability = determine_primary_ability(cls_data['name'],
                                                        ', '.join([p.get('name', '') for p in
cls_data.get('proficiency_choices', [])]))
            # Отримуємо рятувальні кидки
            saving_throws = ', '.join([p.get('name', '').replace('Saving Throw: ', '')
                                       for p in cls_data.get('saving_throws', [])])
            # Формуємо результат
            result = {
                'id': cls_data['index'],
                'name': translate_class_name_to_ukrainian(cls_data['name']),
                'description': description,
                'hit_die': cls_data.get('hit_die', 8),
                'primary_ability': primary_ability,
                'saving_throw_proficiencies': saving_throws
            }
            return jsonify(result)
    # Якщо не використовуємо API або виникла помилка, використовуємо локальну базу даних
    db = get_db()
    cls = db.execute('SELECT * FROM classes WHERE id = ?', (class_id,)).fetchone()
    if not cls:
        return jsonify({'error': 'Клас не знайдено'}), 404
    result = {
        'id': cls['id'],
        'name': cls['name'],
        'description': cls['description'],
        'hit_die': cls['hit_die'],
        'primary_ability': cls['primary_ability'],
        'saving_throw_proficiencies': cls['saving_throw_proficiencies']
    }
    return jsonify(result)
# API для мультикласів
@app.route('/api/multiclass-builds', methods=['GET'])
def get_multiclass_builds():
    db = get_db()
    builds = db.execute('SELECT * FROM multiclass_builds').fetchall()
    result = []
    for build in builds:
        result.append({
            'id': build['id'],
            'name': build['name'],
            'description': build['description'],
            'classes_combination': build['classes_combination'],
            'strengths': build['strengths'],
            'weaknesses': build['weaknesses'],
            'is_optimized': bool(build['is_optimized'])
        })
    return jsonify(result)
# API для рас
@app.route('/api/races', methods=['GET'])
def get_races():
    # Перевіряємо, чи потрібно використовувати API
```

```
    use_dnd_api = request.args.get('use_dnd_api', 'false').lower() == 'true'
    if use_dnd_api:
        # Отримуємо раси з D&D 5e API
        races_data = fetch_from_dnd_api('races')
        if races_data and 'results' in races_data:
            result = []
            for race_info in races_data['results']:
                # Отримуємо деталі раси
                race_data = fetch_from_dnd_api(f"races/{race_info['index']}")
                if race_data:
                    # Формуємо здібності
                    ability_bonuses = []
                    for bonus in race_data.get('ability_bonuses', []):
                        ability = race_data.get('ability_scores', [])
                        ability_name = bonus.get('ability_score', {}).get('name', '')
                        ability_bonus = bonus.get('bonus', 0)
                        ability_bonuses.append({
                            'ability': ability_name,
                            'bonus': ability_bonus
                        })
                    # Формуємо особливості
                    traits = []
                    for trait in race_data.get('traits', []):
                        trait_data = fetch_from_dnd_api(f"traits/{trait['index']}")
                        if trait_data:
                            traits.append({
                                'name': trait_data.get('name', ''),
                                'description': trait_data.get('desc', [''])[0]
                            })
                    # Додаємо расу до результату
                    result.append({
                        'id': race_info['index'],
                        'name': race_data.get('name', ''),
                        'name_uk': translate_race_name_to_ukrainian(race_data.get('name',
''')),
                        'speed': race_data.get('speed', 30),
                        'ability_bonuses': ability_bonuses,
                        'alignment': race_data.get('alignment', ''),
                        'size': race_data.get('size', 'Medium'),
                        'languages': ', '.join([lang.get('name', '') for lang in
race_data.get('languages', [])]),
                        'traits': traits
                    })
            return jsonify(result)
    # Якщо не використовуємо API або виникла помилка, використовуємо локальні дані
    return jsonify(get_basic_races())
# API для заклинань
@app.route('/api/spells', methods=['GET'])
def get_spells():
    # Перевіряємо, чи потрібно використовувати API
    use_dnd_api = request.args.get('use_dnd_api', 'false').lower() == 'true'
    # Отримуємо параметри для фільтрації
    spell_class = request.args.get('class')
    spell_level = request.args.get('level')
    if use_dnd_api:
        # Формуємо параметри запиту
        params = {}
        if spell_class:
            params['class'] = spell_class
        if spell_level:
            params['level'] = spell_level
        # Отримуємо заклинання з D&D 5e API
        endpoint = 'spells'
        if params:
            query_string = '&'.join([f"{key}={value}" for key, value in params.items()])
```

```
            endpoint = f"spells?{query_string}"
        spells_data = fetch_from_dnd_api(endpoint)
        if spells_data and 'results' in spells_data:
            result = []
            for spell_info in spells_data['results'][:20]:  # Обмежуємо кількість для
прикладу
                # Отримуємо деталі заклинання
                spell_data = fetch_from_dnd_api(f"spells/{spell_info['index']}")
                if spell_data:
                    result.append({
                        'id': spell_info['index'],
                        'name': spell_data.get('name', ''),
                        'level': spell_data.get('level', 0),
                        'school': spell_data.get('school', {}).get('name', 'Unknown'),
                        'casting_time': spell_data.get('casting_time', 'Unknown'),
                        'range': spell_data.get('range', 'Unknown'),
                        'components': ', '.join(spell_data.get('components', [])),
                        'duration': spell_data.get('duration', 'Unknown'),
                        'description': '\n'.join(spell_data.get('desc', [''])),
                        'classes': ', '.join([cls.get('name', '') for cls in
spell_data.get('classes', [])])
                    })
                # Невелика пауза, щоб не перевантажувати API
                time.sleep(0.1)
            return jsonify(result)
    # Якщо не використовуємо API або виникла помилка, використовуємо локальну базу даних
    db = get_db()
    query = 'SELECT * FROM spells'
    params = []
    if spell_class:
        query += ' WHERE classes LIKE ?'
        params.append(f'%{spell_class}%')
    if spell_level:
        if 'WHERE' in query:
            query += ' AND level = ?'
        else:
            query += ' WHERE level = ?'
        params.append(int(spell_level))
    spells = db.execute(query, params).fetchall()
    result = []
    for spell in spells:
        result.append({
            'id': spell['id'],
            'name': spell['name'],
            'level': spell['level'],
            'school': spell['school'],
            'casting_time': spell['casting_time'],
            'range': spell['range'],
            'components': spell['components'],
            'duration': spell['duration'],
            'description': spell['description'],
            'classes': spell['classes']
        })
    return jsonify(result)
@app.route('/api/spells/<spell_id>', methods=['GET'])
def get_spell_details(spell_id):
    # Перевіряємо, чи потрібно використовувати API
    use_dnd_api = request.args.get('use_dnd_api', 'false').lower() == 'true'
    if use_dnd_api and not spell_id.isdigit():
        # Отримуємо дані про заклинання з D&D 5e API
        spell_data = fetch_from_dnd_api(f"spells/{spell_id}")
        if spell_data:
            # Формуємо результат
            result = {
                'id': spell_data.get('index', ''),
```

```
                'name': spell_data.get('name', ''),
                'level': spell_data.get('level', 0),
                'school': spell_data.get('school', {}).get('name', 'Unknown'),
                'casting_time': spell_data.get('casting_time', 'Unknown'),
                'range': spell_data.get('range', 'Unknown'),
                'components': ', '.join(spell_data.get('components', [])),
                'duration': spell_data.get('duration', 'Unknown'),
                'description': '\n'.join(spell_data.get('desc', [''])),
                'higher_level': '\n'.join(spell_data.get('higher_level', [''])),
                'material': spell_data.get('material', ''),
                'ritual': spell_data.get('ritual', False),
                'concentration': spell_data.get('concentration', False),
                'classes': ', '.join([cls.get('name', '') for cls in
spell_data.get('classes', [])])
            }
            return jsonify(result)
    # Якщо не використовуємо API або виникла помилка, використовуємо локальну базу даних
    db = get_db()
    spell = db.execute('SELECT * FROM spells WHERE id = ?', (spell_id,)).fetchone()
    if not spell:
        return jsonify({'error': 'Заклинання не знайдено'}), 404
    result = {
        'id': spell['id'],
        'name': spell['name'],
        'level': spell['level'],
        'school': spell['school'],
        'casting_time': spell['casting_time'],
        'range': spell['range'],
        'components': spell['components'],
        'duration': spell['duration'],
        'description': spell['description'],
        'classes': spell['classes']
   }
    return jsonify(result)
# API для випадкового персонажа
@app.route('/api/random-character', methods=['GET'])
def random_character():
   """Створює випадкового персонажа D&D 5e"""
   character_data = generate_basic_random_character()
   return jsonify(character_data)
# API для перевірки білду
@app.route('/api/check-build', methods=['POST'])
@login_required
def check_build():
   data = request.get_json()
   if not data or not data.get('build'):
       return jsonify({'error': 'Необхідно вказати білд для перевірки'}), 400
   build = data['build']
   # Детальний аналіз білду
   # Парсимо інформацію про класи та рівні
   parsed_classes = []
   classes_in_build = build.split('/')
   total_level = 0
   for class_entry in classes_in_build:
       parts = class_entry.strip().split()
       if len(parts) >= 2:
           class_name = parts[0]
           try:
               class_level = int(parts[1])
               total_level += class_level
               parsed_classes.append({'name': class_name, 'level': class_level})
           except ValueError:
               return jsonify({'error': f'Неправильний формат рівня для класу
{class_name}'}), 400
   # Перевірка на загальний рівень
```

```
    if total_level > 20:
        return jsonify({'error': 'Загальний рівень персонажа не може перевищувати 20'}), 400
    # Аналіз синергій між класами на основі їх механік
    synergies = []
    conflicts = []
    # Структури даних для аналізу
    class_types = {
        'Варвар': {'type': 'martial', 'stat': 'str', 'resource': 'rage'},
        'Бард': {'type': 'caster', 'stat': 'cha', 'resource': 'inspiration',
'spell_slot_type': 'full'},
        'Жрець': {'type': 'caster', 'stat': 'wis', 'resource': 'channel_divinity',
'spell_slot_type': 'full'},
        'Друїд': {'type': 'caster', 'stat': 'wis', 'resource': 'wild_shape',
'spell_slot_type': 'full'},
        'Воїн': {'type': 'martial', 'stat': 'str/dex', 'resource': 'action_surge',
'spell_slot_type': None},
        'Монах': {'type': 'martial', 'stat': 'dex/wis', 'resource': 'ki'},
        'Паладин': {'type': 'half-caster', 'stat': 'str/cha', 'resource': 'lay_on_hands',
'spell_slot_type': 'half'},
        'Рейнджер': {'type': 'half-caster', 'stat': 'dex/wis', 'resource': None,
'spell_slot_type': 'half'},
        'Розбійник': {'type': 'martial', 'stat': 'dex', 'resource': 'sneak_attack'},
        'Чародій': {'type': 'caster', 'stat': 'cha', 'resource': 'sorcery_points',
'spell_slot_type': 'full'},
        'Варлок': {'type': 'caster', 'stat': 'cha', 'resource': 'pact_magic',
'spell_slot_type': 'pact'},
        'Чарівник': {'type': 'caster', 'stat': 'int', 'resource': 'arcane_recovery',
'spell_slot_type': 'full'}
    }
    # Ключові рівні, на яких класи отримують важливі здібності
    key_levels = {
        'Варвар': [1, 2, 3, 5, 9],  # Лють, БР, шлях, друга атака, брутальні критичні
        'Бард': [1, 3, 5, 6, 10],  # Натхнення, колегія, третій рівень заклинань, додаткова
магія, магічні таємниці
        'Жрець': [1, 2, 5, 8],  # Заклинання, божественний канал, третій рівень заклинань,
знищення нежиті
        'Друїд': [2, 3, 5, 10, 18, 20],  # Звіроформа, круг, третій рівень заклинань,
перевтілення в стихію, звіроформа без обмеження, архідруїд
        'Воїн': [1, 2, 3, 5, 11, 17, 20],  # Бойовий стиль, сплеск дії, архетип, друга атака,
третя атака, четверта атака, чемпіон
        'Монах': [1, 2, 3, 5, 6, 7],  # БР, кі, шлях, прийоми, приголомшуючий удар, ухилення
        'Паладин': [2, 3, 5, 6, 11],  # Удар божественною силою, клятва, друга атака, аура
захисту, покращений удар
        'Рейнджер': [2, 3, 5, 11],  # Бойовий стиль, архетип, друга атака, архетипна
здібність
        'Розбійник': [1, 2, 3, 5, 7, 11, 13],  # Раптовий удар, хитрі дії, архетип, викрут,
вивертливість, надійний талант, сліпе чуття
        'Чародій': [1, 2, 3, 5, 6, 14, 18],  # Чаротворення, чари, метамагія, третій рівень
заклинань, додатковий метамагічний варіант, втілення магії
        'Варлок': [1, 2, 3, 5, 9, 11, 17],  # Потойбічний заступник, заклинання, договір,
третій рівень заклинань, п'ятий рівень заклинань, містичний аркан, дев'ятий рівень заклинань
        'Чарівник': [1, 2, 3, 5, 6, 10, 14, 18]  # Відновлення аркан, традиція, третій рівень
заклинань, додаткова особливість традиції, магічні таємниці
    }
    # Перевірка на ключові рівні
    class_key_abilities = {}
    for class_info in parsed_classes:
        class_name = class_info['name']
        class_level = class_info['level']
        if class_name in key_levels:
            abilities = []
            for level in key_levels[class_name]:
                if class_level >= level:
                    abilities.append(level)
            class_key_abilities[class_name] = abilities
```

```
    # Аналіз механік та взаємодій між класами
    # 1. Аналіз основних характеристик (MAD - Multiple Ability Dependency)
    used_primary_stats = set()
    for class_info in parsed_classes:
        class_name = class_info['name']
        if class_name in class_types:
            stats = class_types[class_name]['stat'].split('/')
            for stat in stats:
                used_primary_stats.add(stat)
    if len(used_primary_stats) > 3:
        conflicts.append("Цей білд потребує занадто багато основних характеристик (MAD -
Multiple Ability Dependency), що ускладнює оптимізацію")
    # 2. Аналіз слотів заклинань
    caster_levels = {
        'full': 0,
        'half': 0,
        'third': 0,
        'pact': 0
    }
    has_caster = False
    for class_info in parsed_classes:
        class_name = class_info['name']
        if class_name in class_types and class_types[class_name].get('spell_slot_type'):
            has_caster = True
            spell_type = class_types[class_name]['spell_slot_type']
            if spell_type == 'full':
                caster_levels['full'] += class_info['level']
            elif spell_type == 'half':
                caster_levels['half'] += class_info['level']
            elif spell_type == 'third':
                caster_levels['third'] += class_info['level']
            elif spell_type == 'pact':
                caster_levels['pact'] += class_info['level']
    # Розрахунок ефективного рівня заклинателя
    effective_caster_level = caster_levels['full'] + (caster_levels['half'] // 2) +
(caster_levels['third'] // 3)
    # 3. Детальний аналіз специфічних синергій та конфліктів між класами
    # Варвар та магічні класи
    if any(class_info['name'] == 'Варвар' for class_info in parsed_classes) and \
       any(class_info['name'] in ['Чарівник', 'Чародій', 'Варлок', 'Бард'] for class_info in
parsed_classes):
        conflicts.append("Варвар не може використовувати заклинання або концентруватися на
них під час люті, що зменшує ефективність магічних класів")
    # Паладин та інші ча-класи
    if any(class_info['name'] == 'Паладин' for class_info in parsed_classes) and \
       any(class_info['name'] in ['Чародій', 'Бард', 'Варлок'] for class_info in
parsed_classes):
        synergies.append("Поєднання Паладина з класами, які використовують Харизму, дозволяє
ефективно розподілити характеристики")
        if any(class_info['name'] == 'Чародій' for class_info in parsed_classes):
            synergies.append("Комбінація Паладин/Чародій дозволяє використовувати слоти
заклинань для божественного удару і метамагію для гнучкості")
            # Перевірка рівнів для божественної кари
            paladin_level = next((class_info['level'] for class_info in parsed_classes if
class_info['name'] == 'Паладин'), 0)
            sorcerer_level = next((class_info['level'] for class_info in parsed_classes if
class_info['name'] == 'Чародій'), 0)
            if paladin_level >= 2 and sorcerer_level >= 3:
                synergies.append("Палазородін (Паладин 2+ / Чародій 3+): Доступ до слотів
заклинань 2+ рівня для посилених божественних ударів та метамагії")
    # Аналіз загальної ефективності білду
    score = 7  # Починаємо з оцінки вище середньої
    # Коригуємо оцінку на основі синергій та конфліктів
    score += min(len(synergies), 3)  # Максимум +3 за синергії
    score -= min(len(conflicts), 5)  # Максимум -5 за конфлікти
```

```
    # Перевірка на MAD (Multiple Ability Dependency)
    score -= (len(used_primary_stats) - 2) if len(used_primary_stats) > 2 else 0
    # Перевірка на рівень заклинателя
    if has_caster and effective_caster_level < total_level / 2:
        score -= 1  # Штраф за низький рівень заклинателя в білді з кастерами
    # Перевірка на складність гри
    complexity_score = 0
    if len(parsed_classes) > 2:
        complexity_score += len(parsed_classes) - 2
    # Класи з підвищеною складністю гри
    complex_classes = ['Чародій', 'Друїд', 'Чарівник']
    if any(class_info['name'] in complex_classes for class_info in parsed_classes):
        complexity_score += 1
    if complexity_score >= 3:
        conflicts.append(f"Білд має високу складність гри через поєднання багатьох класів
та/або класів зі складними механіками")
        score -= 1
    # Обмежуємо оцінку від 1 до 10
    score = max(1, min(10, score))
    # Визначаємо рекомендацію на основі оцінки
    if score >= 8:
        recommendation = "Оптимальний білд"
    elif score >= 6:
        recommendation = "Хороший білд з незначними недоліками"
    elif score >= 4:
        recommendation = "Збалансований білд, але потребує доопрацювання"
    else:
        recommendation = "Неоптимальний білд, рекомендується переглянути комбінацію класів"
    # Додаткові рекомендації
    additional_recommendations = []
    # Рекомендації по рівням
    for class_info in parsed_classes:
        class_name = class_info['name']
        class_level = class_info['level']
        if class_name in key_levels:
            next_key_level = next((level for level in key_levels[class_name] if level >
class_level), None)
            if next_key_level and next_key_level <= 6 and (next_key_level - class_level) <=
2:
                additional_recommendations.append(f"Розгляньте підвищення рівня класу
{class_name} до {next_key_level} для отримання ключової здібності")
    # Рекомендації по синергіям
    if 'Варвар' in [c['name'] for c in parsed_classes] and not any(c['name'] == 'Воїн' for c
in parsed_classes):
        if any(c['name'] == 'Варвар' and c['level'] >= 5 for c in parsed_classes):
            additional_recommendations.append("Розгляньте додавання 2 рівнів Воїна для Action
Surge, що добре поєднується з Варваром")
    if 'Розбійник' in [c['name'] for c in parsed_classes] and not any(c['name'] == 'Воїн' for
c in parsed_classes):
        rogue_level = next((c['level'] for c in parsed_classes if c['name'] == 'Розбійник'),
0)
        if rogue_level >= 3:
            additional_recommendations.append("Розгляньте додавання 2 рівнів Воїна для
бойового стилю та Action Surge, що підвищить ефективність Розбійника")
    return jsonify({
        'build': build,
        'optimization_score': score,
        'conflicts': conflicts,
        'synergies': synergies,
        'recommendation': recommendation,
        'additional_recommendations': additional_recommendations,
        'class_breakdown': [{'name': c['name'], 'level': c['level'], 'key_abilities':
class_key_abilities.get(c['name'], [])} for c in parsed_classes],
        'effective_caster_level': effective_caster_level if has_caster else 0,
        'primary_stats': list(used_primary_stats),
```

```
        'complexity': complexity_score
    })
# API для персонажів
@app.route('/api/characters', methods=['GET'])
@login_required
def get_characters():
    db = get_db()
    # Різні запити в залежності від ролі користувача
    if session.get('role') == 'dm':
        # DM бачить всіх персонажів у своїх партіях
        characters = db.execute(
            '''SELECT c.*, u.username as owner_name
               FROM characters c
               JOIN users u ON c.user_id = u.id
               LEFT JOIN party_members pm ON c.id = pm.character_id
               LEFT JOIN parties p ON pm.party_id = p.id
               WHERE p.dm_id = ? OR c.user_id = ?''',
            (session['user_id'], session['user_id'])
        ).fetchall()
    else:
        # Гравці бачать лише своїх персонажів
        characters = db.execute('SELECT * FROM characters WHERE user_id = ?',
(session['user_id'],)).fetchall()
    result = []
    for char in characters:
        char_dict = {
            'id': char['id'],
            'name': char['name'],
            'level': char['level'],
            'str': char['strength'],
            'dex': char['dexterity'],
            'con': char['constitution'],
            'int': char['intelligence'],
            'wis': char['wisdom'],
            'cha': char['charisma'],
            'race': char['race'],
            'background': char['background'],
            'classes_combination': char['classes_combination']
        }
        # Додаємо ім'я власника для DM
        if 'owner_name' in char.keys():
            char_dict['owner_name'] = char['owner_name']
        result.append(char_dict)
    return jsonify(result)
@app.route('/api/characters', methods=['POST'])
@login_required
def create_character():
    data = request.get_json()
    if not data:
        return jsonify({'error': 'Необхідно вказати дані персонажа'}), 400
    required_fields = ['name', 'level', 'str', 'dex', 'con', 'int', 'wis', 'cha']
    for field in required_fields:
        if field not in data:
            return jsonify({'error': f'Поле {field} є обов\'язковим'}), 400
    db = get_db()
    db.execute(
        '''INSERT INTO characters
           (user_id, name, level, strength, dexterity, constitution, intelligence, wisdom,
charisma,
            race, background, classes_combination, dndbeyond_id, avatar_url)
           VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)''',
        (session['user_id'], data['name'], data['level'], data['str'], data['dex'],
data['con'],
         data['int'], data['wis'], data['cha'],
```

```
        data.get('race', ''), data.get('background', ''), data.get('classes_combination',
''),
        data.get('dndbeyond_id', ''), data.get('avatar_url', ''))
    )
    db.commit()
    # Отримуємо ID нового персонажа
    character_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    # Якщо є спорядження, додаємо його
    if 'equipment' in data and isinstance(data['equipment'], list):
        for item in data['equipment']:
            if isinstance(item, dict):
                db.execute(
                    '''INSERT INTO character_inventory
                       (character_id, item_name, item_type, quantity, description, weight,
cost, equipped)
                       VALUES (?, ?, ?, ?, ?, ?, ?, ?)''',
                    (character_id, item.get('name', ''), item.get('type', 'gear'),
                     item.get('quantity', 1), item.get('description', ''),
                     item.get('weight', 0), item.get('cost', ''), item.get('equipped', 0))
                )
            elif isinstance(item, str):
                db.execute(
                    '''INSERT INTO character_inventory
                       (character_id, item_name, item_type)
                       VALUES (?, ?, ?)''',
                    (character_id, item, 'gear')
                )
    # Якщо є здібності, додаємо їх
    if 'abilities' in data and isinstance(data['abilities'], list):
        for ability in data['abilities']:
            if isinstance(ability, dict):
                db.execute(
                    '''INSERT INTO character_abilities
                       (character_id, name, description, source, uses_per_day,
uses_remaining)
                       VALUES (?, ?, ?, ?, ?, ?)''',
                    (character_id, ability.get('name', ''), ability.get('description', ''),
                     ability.get('source', ''), ability.get('uses_per_day', 0),
                     ability.get('uses_remaining', 0))
                )
    # Якщо є заклинання, додаємо їх
    if 'spells' in data and isinstance(data['spells'], list):
        for spell in data['spells']:
            if isinstance(spell, dict):
                db.execute(
                    '''INSERT INTO character_spells
                       (character_id, spell_id, spell_name, prepared)
                       VALUES (?, ?, ?, ?)''',
                    (character_id, spell.get('id'), spell.get('name', ''),
                     spell.get('prepared', 0))
                )
    db.commit()
    # Перевірка на запит рекомендацій від ШІ
    result = {'message': 'Персонажа успішно створено', 'character_id': character_id}
    if data.get('get_ai_recommendations', False):
        # В реальному додатку тут був би виклик до ШІ API
        # Для прикладу, просто додаємо базові рекомендації
        result['ai_recommendations'] = {
            'success': True,
            'recommendations': f'''## Рекомендації для персонажа {data['name']}
### Сильні сторони
- Ваш персонаж має хороші показники в ключових характеристиках для вашого класу.
- Рівень {data['level']} дає доступ до кількох потужних здібностей.
### Що можна покращити
- Розгляньте можливість підвищення показника Мудрості для кращих рятувальних кидків.
```

```
- Додайте більше деталей до історії вашого персонажа, щоб зробити його більш цікавим.
### Рекомендоване спорядження
- Додайте до інвентарю зілля лікування для екстрених ситуацій.
- Розгляньте можливість придбання магічних предметів, які покращують ваші основні
здібності.'''
        }
    return jsonify(result), 201
@app.route('/api/characters/<int:character_id>', methods=['GET'])
@login_required
def get_character(character_id):
    db = get_db()
    # Перевіряємо, чи має користувач доступ до персонажа
    character = None
    if session.get('role') == 'dm':
        # DM бачить всіх персонажів у своїх партіях
        character = db.execute(
            '''SELECT c.*, u.username as owner_name
               FROM characters c
               JOIN users u ON c.user_id = u.id
               LEFT JOIN party_members pm ON c.id = pm.character_id
               LEFT JOIN parties p ON pm.party_id = p.id
               WHERE c.id = ? AND (p.dm_id = ? OR c.user_id = ?)''',
            (character_id, session['user_id'], session['user_id'])
        ).fetchone()
    else:
        # Гравці бачать лише своїх персонажів
        character = db.execute(
            'SELECT * FROM characters WHERE id = ? AND user_id = ?',
            (character_id, session['user_id'])
        ).fetchone()
    if not character:
        return jsonify({'error': 'Персонажа не знайдено або у вас немає до нього доступу'}),
404
    # Отримуємо спорядження персонажа
    inventory = db.execute(
        'SELECT * FROM character_inventory WHERE character_id = ?',
        (character_id,)
    ).fetchall()
    # Отримуємо здібності персонажа
    abilities = db.execute(
        'SELECT * FROM character_abilities WHERE character_id = ?',
        (character_id,)
    ).fetchall()
    # Отримуємо заклинання персонажа
    spells = db.execute(
        '''SELECT cs.*, s.level, s.school, s.description
           FROM character_spells cs
           LEFT JOIN spells s ON cs.spell_id = s.id
           WHERE cs.character_id = ?''',
        (character_id,)
    ).fetchall()
    # Формуємо результат
    result = {
        'id': character['id'],
        'name': character['name'],
        'level': character['level'],
        'abilities': {
            'str': character['strength'],
            'dex': character['dexterity'],
            'con': character['constitution'],
            'int': character['intelligence'],
            'wis': character['wisdom'],
            'cha': character['charisma']
        },
        'race': character['race'],
```

```
            'background': character['background'],
            'classes_combination': character['classes_combination'],
            'avatar_url': character['avatar_url'],
            'dndbeyond_id': character['dndbeyond_id'],
            'inventory': [],
            'character_abilities': [],
            'spells': []
        }
        # Додаємо спорядження
        for item in inventory:
            result['inventory'].append({
                'id': item['id'],
                'name': item['item_name'],
                'type': item['item_type'],
                'quantity': item['quantity'],
                'description': item['description'],
                'weight': item['weight'],
                'cost': item['cost'],
                'equipped': bool(item['equipped'])
            })
        # Додаємо здібності
        for ability in abilities:
            result['character_abilities'].append({
                'id': ability['id'],
                'name': ability['name'],
                'description': ability['description'],
                'source': ability['source'],
                'uses_per_day': ability['uses_per_day'],
                'uses_remaining': ability['uses_remaining']
            })
        # Додаємо заклинання
        for spell in spells:
            spell_info = {
                'id': spell['id'],
                'spell_id': spell['spell_id'],
                'name': spell['spell_name'],
                'prepared': bool(spell['prepared'])
            }
            # Додаємо додаткову інформацію, якщо є
            if spell['level'] is not None:
                spell_info['level'] = spell['level']
            if spell['school'] is not None:
                spell_info['school'] = spell['school']
            if spell['description'] is not None:
                spell_info['description'] = spell['description']
            result['spells'].append(spell_info)
        # Додаємо ім'я власника для DM
        if 'owner_name' in character.keys():
            result['owner_name'] = character['owner_name']
        return jsonify(result)
    @app.route('/api/optimize-multiclass', methods=['POST'])
    @login_required
    def optimize_multiclass():
        data = request.get_json()
        if not data:
            return jsonify({'error': 'Необхідно надати дані персонажа'}), 400
        # Отримуємо характеристики персонажа
        abilities = {
            'str': data.get('str', 10),
            'dex': data.get('dex', 10),
            'con': data.get('con', 10),
            'int': data.get('int', 10),
            'wis': data.get('wis', 10),
            'cha': data.get('cha', 10)
        }
```

```
    # Визначаємо класи, доступні для мультикласу
    available_classes = determine_available_multiclass_options(abilities)
    # Шукаємо рекомендовані білди
    recommended_builds = find_recommended_builds(abilities, available_classes)
    return jsonify({
        'available_classes': available_classes,
        'recommended_builds': recommended_builds
    })
def determine_available_multiclass_options(abilities):
    """Визначає, які класи доступні для мультикласу на основі характеристик"""
    available = []
    # Мінімальні вимоги для мультикласу
    requirements = {
        'Варвар': {'str': 13},
        'Бард': {'cha': 13},
        'Жрець': {'wis': 13},
        'Друїд': {'wis': 13},
        'Воїн': {'str': 13, 'dex': 13}, # Потрібна або сила, або спритність >= 13
        'Монах': {'dex': 13, 'wis': 13},
        'Паладин': {'str': 13, 'cha': 13},
        'Рейнджер': {'dex': 13, 'wis': 13},
        'Розбійник': {'dex': 13},
        'Чародій': {'cha': 13},
        'Варлок': {'cha': 13},
        'Чарівник': {'int': 13}
    }
    for class_name, req in requirements.items():
        can_multiclass = True
        # Особливий випадок для Воїна (потрібна або сила, або спритність >= 13)
        if class_name == 'Воїн':
            if abilities['str'] < 13 and abilities['dex'] < 13:
                can_multiclass = False
        else:
            # Перевіряємо всі вимоги
            for stat, min_value in req.items():
                if abilities[stat] < min_value:
                    can_multiclass = False
                    break
        if can_multiclass:
            available.append(class_name)
    return available
def find_recommended_builds(abilities, available_classes):
    """Шукає рекомендовані білди на основі характеристик персонажа"""
    db = get_db()
    all_builds = db.execute('SELECT * FROM multiclass_builds').fetchall()
    recommended = []
    for build in all_builds:
        # Парсимо комбінацію класів
        classes_in_build = [cls.strip().split()[0] for cls in
build['classes_combination'].split('/')]
        # Перевіряємо, чи всі класи в білді доступні
        if all(cls in available_classes for cls in classes_in_build):
            # Рахуємо сумісність білда з характеристиками персонажа
            compatibility_score = calculate_compatibility_score(build['classes_combination'],
abilities)
            build_info = {
                'id': build['id'],
                'name': build['name'],
                'description': build['description'],
                'classes_combination': build['classes_combination'],
                'strengths': build['strengths'],
                'weaknesses': build['weaknesses'],
                'is_optimized': bool(build['is_optimized']),
                'compatibility_score': compatibility_score
            }
```

```
            recommended.append(build_info)
    # Сортуємо за сумісністю
    recommended.sort(key=lambda x: x['compatibility_score'], reverse=True)
    # Повертаємо найкращі 5 білдів
    return recommended[:5]
def calculate_compatibility_score(classes_combination, abilities):
    """Рахує оцінку сумісності білда з характеристиками персонажа"""
    # Основні характеристики для кожного класу
    primary_stats = {
        'Варвар': ['str', 'con'],
        'Бард': ['cha', 'dex'],
        'Жрець': ['wis', 'con'],
        'Друїд': ['wis', 'con'],
        'Воїн': ['str', 'dex', 'con'],
        'Монах': ['dex', 'wis'],
        'Паладин': ['str', 'cha'],
        'Рейнджер': ['dex', 'wis'],
        'Розбійник': ['dex', 'int'],
        'Чародій': ['cha', 'con'],
        'Варлок': ['cha', 'con'],
        'Чарівник': ['int', 'con']
    }
    # Парсимо комбінацію класів
    classes = []
    for class_entry in classes_combination.split('/'):
        parts = class_entry.strip().split()
        if len(parts) >= 2:
            class_name = parts[0]
            try:
                class_level = int(parts[1])
                classes.append((class_name, class_level))
            except ValueError:
                pass
    # Розраховуємо базову сумісність
    total_weight = 0
    total_score = 0
    for class_name, class_level in classes:
        if class_name in primary_stats:
            # Вага класу залежить від його рівня
            weight = class_level
            total_weight += weight
            # Оцінка для кожної основної характеристики
            for stat in primary_stats[class_name]:
                if stat in abilities:
                    if abilities[stat] >= 16:
                        stat_score = 100
                    elif abilities[stat] >= 14:
                        stat_score = 75
                    elif abilities[stat] >= 12:
                        stat_score = 50
                    else:
                        stat_score = 25
                    total_score += stat_score * weight
    # Розраховуємо середню сумісність
    if total_weight > 0:
        compatibility = total_score / (total_weight * 100) * 100
    else:
        compatibility = 50  # Значення за замовчуванням
    # Округляємо до цілого числа
    return round(compatibility)
# API для партій
@app.route('/api/parties', methods=['GET'])
@login_required
def get_parties():
    db = get_db()
```

```
    if session.get('role') == 'dm':
        # DM бачить створені ним партії
        parties = db.execute(
            'SELECT * FROM parties WHERE dm_id = ?',
            (session['user_id'],)
        ).fetchall()
    else:
        # Гравці бачать партії, в яких вони є членами
        parties = db.execute(
            '''SELECT p.* FROM parties p
               JOIN party_members pm ON p.id = pm.party_id
               WHERE pm.user_id = ?''',
            (session['user_id'],)
        ).fetchall()
    result = []
    for party in parties:
        # Отримуємо кількість членів партії
        member_count = db.execute(
            'SELECT COUNT(*) FROM party_members WHERE party_id = ?',
            (party['id'],)
        ).fetchone()[0]
        result.append({
            'id': party['id'],
            'name': party['name'],
            'description': party['description'],
            'dm_id': party['dm_id'],
            'created_at': party['created_at'],
            'member_count': member_count
        })
    return jsonify(result)
@app.route('/api/parties/<party_id>', methods=['GET'])
@login_required
def get_party(party_id):
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    party = db.execute(
        '''SELECT p.* FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone()
    if not party:
        return jsonify({'error': 'Партію не знайдено або у вас немає до неї доступу'}), 404
    # Отримуємо членів партії та їхніх персонажів
    members = db.execute(
        '''SELECT u.id as user_id, u.username, u.role,
           c.id as character_id, c.name as character_name,
           c.level as character_level, c.race as character_race,
           c.classes_combination as character_classes
           FROM party_members pm
           JOIN users u ON pm.user_id = u.id
           LEFT JOIN characters c ON pm.character_id = c.id
           WHERE pm.party_id = ?''',
        (party_id,)
    ).fetchall()
    members_data = []
    for member in members:
        member_info = {
            'user_id': member['user_id'],
            'username': member['username'],
            'role': member['role'],
            'character': None
        }
        if member['character_id']:
            member_info['character'] = {
```

```
                'id': member['character_id'],
                'name': member['character_name'],
                'level': member['character_level'],
                'race': member['character_race'],
                'classes_combination': member['character_classes']
            }
        members_data.append(member_info)
    # Додаємо інформацію про DM
    dm_info = db.execute(
        '''SELECT u.id, u.username FROM users u
           JOIN parties p ON u.id = p.dm_id
           WHERE p.id = ?''',
        (party_id,)
    ).fetchone()
    result = {
        'id': party['id'],
        'name': party['name'],
        'description': party['description'],
        'dm_id': party['dm_id'],
        'dm_username': dm_info['username'] if dm_info else None,
        'created_at': party['created_at'],
        'members': members_data,
        'is_dm': party['dm_id'] == session['user_id']
    }
    return jsonify(result)
@app.route('/api/parties', methods=['POST'])
@login_required
@dm_required
def create_party():
    data = request.get_json()
    if not data or not data.get('name'):
        return jsonify({'error': 'Необхідно вказати назву партії'}), 400
    db = get_db()
    db.execute(
        'INSERT INTO parties (name, dm_id, description) VALUES (?, ?, ?)',
        (data['name'], session['user_id'], data.get('description', ''))
    )
    db.commit()
    party_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    return jsonify({
        'message': 'Партію успішно створено',
        'party_id': party_id
    }), 201
@app.route('/api/parties/<party_id>/members', methods=['POST'])
@login_required
@dm_required
def add_party_member(party_id):
    data = request.get_json()
    if not data or not data.get('user_id'):
        return jsonify({'error': 'Необхідно вказати ID користувача'}), 400
    db = get_db()
    # Перевіряємо, чи існує партія і чи поточний користувач є її DM
    party = db.execute(
        'SELECT * FROM parties WHERE id = ? AND dm_id = ?',
        (party_id, session['user_id'])
    ).fetchone()
    if not party:
        return jsonify({'error': 'Партію не знайдено або ви не є її Dungeon Master\'ом'}),
404
    # Перевіряємо, чи існує користувач
    user = db.execute(
        'SELECT * FROM users WHERE id = ?',
        (data['user_id'],)
    ).fetchone()
    if not user:
```

```
        return jsonify({'error': 'Користувача не знайдено'}), 404
    # Перевіряємо, чи користувач вже є членом партії
    existing_member = db.execute(
        'SELECT * FROM party_members WHERE party_id = ? AND user_id = ?',
        (party_id, data['user_id'])
    ).fetchone()
    if existing_member:
        return jsonify({'error': 'Користувач вже є членом цієї партії'}), 400
    # Додаємо користувача до партії
    db.execute(
        'INSERT INTO party_members (party_id, user_id, character_id) VALUES (?, ?, ?)',
        (party_id, data['user_id'], data.get('character_id'))
    )
    db.commit()
    return jsonify({'message': 'Користувача успішно додано до партії'}), 201
@app.route('/api/parties/<party_id>/members/<user_id>', methods=['DELETE'])
@login_required
@dm_required
def remove_party_member(party_id, user_id):
    db = get_db()
    # Перевіряємо, чи існує партія і чи поточний користувач є її DM
    party = db.execute(
        'SELECT * FROM parties WHERE id = ? AND dm_id = ?',
        (party_id, session['user_id'])
    ).fetchone()
    if not party:
        return jsonify({'error': 'Партію не знайдено або ви не є її Dungeon Master\'ом'}),
404
    # Видаляємо користувача з партії
    db.execute(
        'DELETE FROM party_members WHERE party_id = ? AND user_id = ?',
        (party_id, user_id)
    )
    db.commit()
    return jsonify({'message': 'Користувача успішно видалено з партії'}), 200
@app.route('/api/parties/<party_id>/members/<user_id>/character', methods=['PUT'])
@login_required
def update_party_member_character(party_id, user_id):
    data = request.get_json()
    if not data or not data.get('character_id'):
        return jsonify({'error': 'Необхідно вказати ID персонажа'}), 400
    db = get_db()
    # Перевіряємо, чи поточний користувач є DM партії або власником персонажа
    is_dm = db.execute(
        'SELECT * FROM parties WHERE id = ? AND dm_id = ?',
        (party_id, session['user_id'])
    ).fetchone() is not None
    is_owner = session['user_id'] == int(user_id)
    if not (is_dm or is_owner):
        return jsonify({'error': 'Ви не маєте прав для оновлення цього персонажа в партії'}),
403
    # Перевіряємо, чи існує персонаж і чи належить він указаному користувачу
    character = db.execute(
        'SELECT * FROM characters WHERE id = ? AND user_id = ?',
        (data['character_id'], user_id)
    ).fetchone()
    if not character:
        return jsonify({'error': 'Персонажа не знайдено або він не належить вказаному
користувачу'}), 404
    # Оновлюємо персонажа у партії
    db.execute(
        'UPDATE party_members SET character_id = ? WHERE party_id = ? AND user_id = ?',
        (data['character_id'], party_id, user_id)
    )
    db.commit()
```

```
    return jsonify({'message': 'Персонажа успішно оновлено в партії'}), 200
# API для чату партії
@app.route('/api/parties/<party_id>/chat', methods=['GET'])
@login_required
def get_party_chat(party_id):
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    is_member = db.execute(
        '''SELECT 1 FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone() is not None
    if not is_member:
        return jsonify({'error': 'У вас немає доступу до чату цієї партії'}), 403
    # Отримуємо повідомлення чату
    messages = db.execute(
        '''SELECT cm.id, cm.sender_id, u.username as sender_name, u.role as sender_role,
           cm.message, cm.message_type, cm.sent_at
           FROM chat_messages cm
           JOIN users u ON cm.sender_id = u.id
           WHERE cm.party_id = ?
           ORDER BY cm.sent_at ASC''',
        (party_id,)
    ).fetchall()
    result = []
    for msg in messages:
        result.append({
            'id': msg['id'],
            'sender_id': msg['sender_id'],
            'sender_name': msg['sender_name'],
            'sender_role': msg['sender_role'],
            'message': msg['message'],
            'message_type': msg['message_type'],
            'sent_at': msg['sent_at'],
            'is_own': msg['sender_id'] == session['user_id']
        })
    return jsonify(result)
@app.route('/api/parties/<party_id>/chat', methods=['POST'])
@login_required
def send_party_chat_message(party_id):
    data = request.get_json()
    if not data or not data.get('message'):
        return jsonify({'error': 'Необхідно вказати текст повідомлення'}), 400
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    is_member = db.execute(
        '''SELECT 1 FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone() is not None
    if not is_member:
        return jsonify({'error': 'У вас немає доступу до чату цієї партії'}), 403
    # Додаємо повідомлення в чат
    db.execute(
        'INSERT INTO chat_messages (sender_id, party_id, message, message_type, sent_at)
VALUES (?, ?, ?, ?, datetime("now"))',
        (session['user_id'], party_id, data['message'], data.get('message_type', 'text'))
    )
    db.commit()
    # Отримуємо ID нового повідомлення
    message_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    # Отримуємо дані відправника
    sender = db.execute(
```

```
        'SELECT username, role FROM users WHERE id = ?',
        (session['user_id'],)
    ).fetchone()
    return jsonify({
        'message': 'Повідомлення успішно відправлено',
        'message_data': {
            'id': message_id,
            'sender_id': session['user_id'],
            'sender_name': sender['username'],
            'sender_role': sender['role'],
            'message': data['message'],
            'message_type': data.get('message_type', 'text'),
            'sent_at': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            'is_own': True
        }
    }), 201
# API для приватного чату
@app.route('/api/chats/<user_id>', methods=['GET'])
@login_required
def get_private_chat(user_id):
    db = get_db()
    # Перевіряємо, чи існує такий користувач
    user = db.execute('SELECT id, username, role FROM users WHERE id = ?',
(user_id,)).fetchone()
    if not user:
        return jsonify({'error': 'Користувача не знайдено'}), 404
    # Отримуємо повідомлення чату
    messages = db.execute(
        '''SELECT cm.id, cm.sender_id, u.username as sender_name, u.role as sender_role,
           cm.message, cm.message_type, cm.sent_at
           FROM chat_messages cm
           JOIN users u ON cm.sender_id = u.id
           WHERE (cm.sender_id = ? AND cm.recipient_id = ?) OR (cm.sender_id = ? AND
cm.recipient_id = ?)
           ORDER BY cm.sent_at ASC''',
        (session['user_id'], user_id, user_id, session['user_id'])
    ).fetchall()
    result = []
    for msg in messages:
        result.append({
            'id': msg['id'],
            'sender_id': msg['sender_id'],
            'sender_name': msg['sender_name'],
            'sender_role': msg['sender_role'],
            'message': msg['message'],
            'message_type': msg['message_type'],
            'sent_at': msg['sent_at'],
            'is_own': msg['sender_id'] == session['user_id']
        })
    return jsonify({
        'user': {
            'id': user['id'],
            'username': user['username'],
            'role': user['role']
        },
        'messages': result
    })
@app.route('/api/chats/<user_id>', methods=['POST'])
@login_required
def send_private_message(user_id):
    data = request.get_json()
    if not data or not data.get('message'):
        return jsonify({'error': 'Необхідно вказати текст повідомлення'}), 400
    db = get_db()
    # Перевіряємо, чи існує такий користувач
```

```
    user = db.execute('SELECT id, username, role FROM users WHERE id = ?',
(user_id,)).fetchone()
    if not user:
        return jsonify({'error': 'Користувача не знайдено'}), 404
    # Додаємо повідомлення
    db.execute(
        'INSERT INTO chat_messages (sender_id, recipient_id, message, message_type, sent_at)
VALUES (?, ?, ?, ?, datetime("now"))',
        (session['user_id'], user_id, data['message'], data.get('message_type', 'text'))
    )
    db.commit()
    # Отримуємо ID нового повідомлення
    message_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    # Отримуємо дані відправника
    sender = db.execute(
        'SELECT username, role FROM users WHERE id = ?',
        (session['user_id'],)
    ).fetchone()
    return jsonify({
        'message': 'Повідомлення успішно відправлено',
        'message_data': {
            'id': message_id,
            'sender_id': session['user_id'],
            'sender_name': sender['username'],
            'sender_role': sender['role'],
            'message': data['message'],
            'message_type': data.get('message_type', 'text'),
            'sent_at': datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            'is_own': True
        }
    }), 201
# API для отримання списку користувачів
@app.route('/api/users', methods=['GET'])
@login_required
def get_chat_users():
    db = get_db()
    # DM бачить всіх користувачів, а гравці - тільки DM та учасників своїх партій
    if session.get('role') == 'dm':
        users = db.execute(
            'SELECT id, username, role FROM users WHERE id != ?',
            (session['user_id'],)
        ).fetchall()
    else:
        # Гравці бачать DM та інших гравців зі своїх партій
        users = db.execute(
            '''SELECT DISTINCT u.id, u.username, u.role
               FROM users u
               LEFT JOIN parties p ON u.id = p.dm_id
               LEFT JOIN party_members pm1 ON p.id = pm1.party_id
               LEFT JOIN party_members pm2 ON u.id = pm2.user_id
               WHERE u.id != ? AND (u.role = 'dm' OR
                     pm2.party_id IN (SELECT party_id FROM party_members WHERE user_id =
?))''',
            (session['user_id'], session['user_id'])
        ).fetchall()
    result = []
    for user in users:
        result.append({
            'id': user['id'],
            'username': user['username'],
            'role': user['role']
        })
    return jsonify(result)
# API для кампаній та сесій
@app.route('/api/parties/<party_id>/sessions', methods=['GET'])
```

```
@login_required
def get_sessions(party_id):
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    is_member = db.execute(
        '''SELECT 1 FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone() is not None
    if not is_member:
        return jsonify({'error': 'У вас немає доступу до цієї партії'}), 403
    # Отримуємо сесії кампанії
    sessions = db.execute(
        '''SELECT cs.*, u.username as creator_name
           FROM campaign_sessions cs
           LEFT JOIN users u ON cs.created_by = u.id
           WHERE cs.party_id = ?
           ORDER BY cs.scheduled_date DESC''',
        (party_id,)
    ).fetchall()
    result = []
    for sess in sessions:
        result.append({
            'id': sess['id'],
            'title': sess['title'],
            'scheduled_date': sess['scheduled_date'],
            'duration': sess['duration'],
            'status': sess['status'],
            'notes': sess['notes'],
            'created_by': sess['created_by'],
            'creator_name': sess['creator_name']
        })
    return jsonify(result)
@app.route('/api/parties/<party_id>/sessions', methods=['POST'])
@login_required
def create_session(party_id):
    data = request.get_json()
    if not data or not data.get('title'):
        return jsonify({'error': 'Необхідно вказати назву сесії'}), 400
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    is_member = db.execute(
        '''SELECT 1 FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone() is not None
    if not is_member:
        return jsonify({'error': 'У вас немає доступу до цієї партії'}), 403
    # Додаємо сесію
    db.execute(
        '''INSERT INTO campaign_sessions
           (party_id, title, scheduled_date, duration, status, notes, created_by)
           VALUES (?, ?, ?, ?, ?, ?, ?)''',
        (party_id, data['title'], data.get('scheduled_date'), data.get('duration'),
         data.get('status', 'planned'), data.get('notes'), session['user_id'])
    )
    db.commit()
    # Отримуємо ID нової сесії
    session_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    return jsonify({
        'message': 'Сесію успішно створено',
        'session_id': session_id
    }), 201
```

```
# API для нотаток кампанії
@app.route('/api/parties/<party_id>/notes', methods=['GET'])
@login_required
def get_notes(party_id):
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    is_member = db.execute(
        '''SELECT 1 FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone() is not None
    if not is_member:
        return jsonify({'error': 'У вас немає доступу до цієї партії'}), 403
    # Отримуємо нотатки партії
    notes_query = '''
        SELECT cn.*, u.username as creator_name
        FROM campaign_notes cn
        LEFT JOIN users u ON cn.created_by = u.id
        WHERE cn.party_id = ? AND (cn.is_public = 1 OR cn.created_by = ?'''
    # DM бачить всі нотатки
    is_dm = db.execute(
        'SELECT 1 FROM parties WHERE id = ? AND dm_id = ?',
        (party_id, session['user_id'])
    ).fetchone() is not None
    if is_dm:
        notes_query += ' OR 1=1)'
    else:
        notes_query += ')'
    notes = db.execute(notes_query, (party_id, session['user_id'])).fetchall()
    result = []
    for note in notes:
        result.append({
            'id': note['id'],
            'title': note['title'],
            'content': note['content'],
            'is_public': bool(note['is_public']),
            'created_at': note['created_at'],
            'updated_at': note['updated_at'],
            'created_by': note['created_by'],
            'creator_name': note['creator_name']
        })
    return jsonify(result)
@app.route('/api/parties/<party_id>/notes', methods=['POST'])
@login_required
def create_note(party_id):
    data = request.get_json()
    if not data or not data.get('title'):
        return jsonify({'error': 'Необхідно вказати назву нотатки'}), 400
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    is_member = db.execute(
        '''SELECT 1 FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone() is not None
    if not is_member:
        return jsonify({'error': 'У вас немає доступу до цієї партії'}), 403
    # Додаємо нотатку
    db.execute(
        '''INSERT INTO campaign_notes
           (party_id, title, content, is_public, created_by)
           VALUES (?, ?, ?, ?, ?)''',
        (party_id, data['title'], data.get('content', ''),
```

```
            data.get('is_public', False), session['user_id'])
    )
    db.commit()
    # Отримуємо ID нової нотатки
    note_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    return jsonify({
        'message': 'Нотатку успішно створено',
        'note_id': note_id
    }), 201
# API для бою
@app.route('/api/parties/<party_id>/combat', methods=['GET'])
@login_required
def get_combat_encounters(party_id):
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    is_member = db.execute(
        '''SELECT 1 FROM parties p
           LEFT JOIN party_members pm ON p.id = pm.party_id
           WHERE p.id = ? AND (p.dm_id = ? OR pm.user_id = ?)''',
        (party_id, session['user_id'], session['user_id'])
    ).fetchone() is not None
    if not is_member:
        return jsonify({'error': 'У вас немає доступу до цієї партії'}), 403
    # Отримуємо бойові сутички
    encounters = db.execute(
        '''SELECT *
           FROM combat_encounters
           WHERE party_id = ?
           ORDER BY created_at DESC''',
        (party_id,)
    ).fetchall()
    result = []
    for encounter in encounters:
        # Отримуємо учасників бою
        participants = db.execute(
            '''SELECT *
               FROM combat_participants
               WHERE encounter_id = ?
               ORDER BY turn_order''',
            (encounter['id'],)
        ).fetchall()
        participants_data = []
        for participant in participants:
            participants_data.append({
                'id': participant['id'],
                'name': participant['name'],
                'initiative': participant['initiative'],
                'hp_max': participant['hp_max'],
                'hp_current': participant['hp_current'],
                'ac': participant['ac'],
                'is_pc': bool(participant['is_pc']),
                'character_id': participant['character_id'],
                'turn_order': participant['turn_order'],
                'is_active': bool(participant['is_active']),
                'status_effects': participant['status_effects']
            })
        result.append({
            'id': encounter['id'],
            'name': encounter['name'],
            'status': encounter['status'],
            'round': encounter['round'],
            'created_at': encounter['created_at'],
            'participants': participants_data
        })
    return jsonify(result)
```

```
@app.route('/api/parties/<party_id>/combat', methods=['POST'])
@login_required
@dm_required
def create_combat_encounter(party_id):
    data = request.get_json()
    if not data or not data.get('name'):
        return jsonify({'error': 'Необхідно вказати назву бойової сутички'}), 400
    db = get_db()
    # Перевіряємо, чи має користувач доступ до партії
    party = db.execute(
        'SELECT * FROM parties WHERE id = ? AND dm_id = ?',
        (party_id, session['user_id'])
    ).fetchone()
    if not party:
        return jsonify({'error': 'Партію не знайдено або ви не є її Dungeon Master\'ом'}),
404
    # Додаємо бойову сутичку
    db.execute(
        '''INSERT INTO combat_encounters
           (party_id, name, status, round)
           VALUES (?, ?, ?, ?)''',
        (party_id, data['name'], data.get('status', 'preparation'), 0)
    )
    db.commit()
    # Отримуємо ID нової бойової сутички
    encounter_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    # Додаємо учасників, якщо вони є у запиті
    if 'participants' in data and isinstance(data['participants'], list):
        for i, participant in enumerate(data['participants']):
            if isinstance(participant, dict):
                db.execute(
                    '''INSERT INTO combat_participants
                       (encounter_id, name, initiative, hp_max, hp_current, ac, is_pc,
character_id, turn_order, is_active, status_effects)
                       VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)''',
                    (encounter_id, participant.get('name', 'Учасник'),
                     participant.get('initiative', 0),
                     participant.get('hp_max', 0),
                     participant.get('hp_current', 0),
                     participant.get('ac', 0),
                     participant.get('is_pc', False),
                     participant.get('character_id'),
                     i, False, participant.get('status_effects', ''))
                )
    db.commit()
    return jsonify({
        'message': 'Бойову сутичку успішно створено',
        'encounter_id': encounter_id
    }), 201
@app.route('/api/combat/<encounter_id>/participants', methods=['POST'])
@login_required
@dm_required
def add_combat_participant(encounter_id):
    data = request.get_json()
    if not data or not data.get('name'):
        return jsonify({'error': 'Необхідно вказати ім\'я учасника'}), 400
    db = get_db()
    # Перевіряємо, чи існує бойова сутичка і чи має користувач права DM
    encounter = db.execute(
        '''SELECT ce.*
           FROM combat_encounters ce
           JOIN parties p ON ce.party_id = p.id
           WHERE ce.id = ? AND p.dm_id = ?''',
        (encounter_id, session['user_id'])
    ).fetchone()
```

```
    if not encounter:
        return jsonify({'error': 'Бойову сутичку не знайдено або у вас немає до неї
доступу'}), 404
    # Визначаємо порядок ходу нового учасника
    turn_order = db.execute(
        'SELECT COUNT(*) FROM combat_participants WHERE encounter_id = ?',
        (encounter_id,)
    ).fetchone()[0]
    # Додаємо учасника
    db.execute(
        '''INSERT INTO combat_participants
           (encounter_id, name, initiative, hp_max, hp_current, ac, is_pc, character_id,
turn_order, is_active, status_effects)
           VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)''',
        (encounter_id, data['name'],
         data.get('initiative', 0),
         data.get('hp_max', 0),
         data.get('hp_current', 0),
         data.get('ac', 0),
         data.get('is_pc', False),
         data.get('character_id'),
         turn_order, False, data.get('status_effects', ''))
    )
    db.commit()
    # Отримуємо ID нового учасника
    participant_id = db.execute('SELECT last_insert_rowid()').fetchone()[0]
    return jsonify({
        'message': 'Учасника успішно додано',
        'participant_id': participant_id
    }), 201
@app.route('/api/combat/<encounter_id>/start', methods=['POST'])
@login_required
@dm_required
def start_combat(encounter_id):
    db = get_db()
    # Перевіряємо, чи існує бойова сутичка і чи має користувач права DM
    encounter = db.execute(
        '''SELECT ce.*
           FROM combat_encounters ce
           JOIN parties p ON ce.party_id = p.id
           WHERE ce.id = ? AND p.dm_id = ?''',
        (encounter_id, session['user_id'])
    ).fetchone()
    if not encounter:
        return jsonify({'error': 'Бойову сутичку не знайдено або у вас немає до неї
доступу'}), 404
    # Сортуємо учасників за ініціативою
    participants = db.execute(
        'SELECT * FROM combat_participants WHERE encounter_id = ? ORDER BY initiative DESC',
        (encounter_id,)
    ).fetchall()
    # Оновлюємо порядок ходу
    for i, participant in enumerate(participants):
        db.execute(
            'UPDATE combat_participants SET turn_order = ? WHERE id = ?',
            (i, participant['id'])
        )
    # Встановлюємо активного учасника (першого за ініціативою)
    if participants:
        db.execute(
            'UPDATE combat_participants SET is_active = 1 WHERE id = ?',
            (participants[0]['id'],)
        )
    # Оновлюємо статус бою
    db.execute(
```

```
        'UPDATE combat_encounters SET status = ?, round = 1 WHERE id = ?',
        ('active', encounter_id)
    )
    db.commit()
    return jsonify({'message': 'Бій розпочато', 'current_round': 1}), 200
@app.route('/api/combat/<encounter_id>/next-turn', methods=['POST'])
@login_required
@dm_required
def next_combat_turn(encounter_id):
    db = get_db()
    # Перевіряємо, чи існує бойова сутичка і чи має користувач права DM
    encounter = db.execute(
        '''SELECT ce.*
           FROM combat_encounters ce
           JOIN parties p ON ce.party_id = p.id
           WHERE ce.id = ? AND p.dm_id = ?''',
        (encounter_id, session['user_id'])
    ).fetchone()
    if not encounter:
        return jsonify({'error': 'Бойову сутичку не знайдено або у вас немає до неї
доступу'}), 404
    # Знаходимо поточного активного учасника
    active_participant = db.execute(
        'SELECT * FROM combat_participants WHERE encounter_id = ? AND is_active = 1',
        (encounter_id,)
    ).fetchone()
    # Отримуємо всіх учасників, відсортованих за порядком ходу
    participants = db.execute(
        'SELECT * FROM combat_participants WHERE encounter_id = ? ORDER BY turn_order',
        (encounter_id,)
    ).fetchall()
    if not participants:
        return jsonify({'error': 'У бою немає учасників'}), 400
    # Визначаємо наступного учасника
    next_index = 0
    if active_participant:
        # Знімаємо активність з поточного учасника
        db.execute(
            'UPDATE combat_participants SET is_active = 0 WHERE id = ?',
            (active_participant['id'],)
        )
        # Знаходимо наступного учасника
        current_index = active_participant['turn_order']
        next_index = (current_index + 1) % len(participants)
        # Якщо це перший учасник в новому раунді, збільшуємо лічильник раундів
        if next_index == 0:
            db.execute(
                'UPDATE combat_encounters SET round = round + 1 WHERE id = ?',
                (encounter_id,)
            )
    # Встановлюємо активність наступному учаснику
    db.execute(
        'UPDATE combat_participants SET is_active = 1 WHERE id = ?',
        (participants[next_index]['id'],)
    )
    db.commit()
    # Отримуємо оновлені дані про бій
    updated_encounter = db.execute(
        'SELECT * FROM combat_encounters WHERE id = ?',
        (encounter_id,)
    ).fetchone()
    # Отримуємо оновленого активного учасника
    updated_active = db.execute(
        'SELECT * FROM combat_participants WHERE encounter_id = ? AND is_active = 1',
        (encounter_id,)
```

```
    ).fetchone()
    return jsonify({
        'message': 'Хід передано',
        'current_round': updated_encounter['round'],
        'active_participant': {
            'id': updated_active['id'],
            'name': updated_active['name'],
            'initiative': updated_active['initiative'],
            'hp_current': updated_active['hp_current'],
            'hp_max': updated_active['hp_max'],
            'ac': updated_active['ac'],
            'is_pc': bool(updated_active['is_pc']),
            'status_effects': updated_active['status_effects']
        }
    }), 200
@app.route('/api/combat/<encounter_id>/end', methods=['POST'])
@login_required
@dm_required
def end_combat(encounter_id):
    db = get_db()
    # Перевіряємо, чи існує бойова сутичка і чи має користувач права DM
    encounter = db.execute(
        '''SELECT ce.*
           FROM combat_encounters ce
           JOIN parties p ON ce.party_id = p.id
           WHERE ce.id = ? AND p.dm_id = ?''',
        (encounter_id, session['user_id'])
    ).fetchone()
    if not encounter:
        return jsonify({'error': 'Бойову сутичку не знайдено або у вас немає до неї
доступу'}), 404
    # Знімаємо активність з усіх учасників
    db.execute(
        'UPDATE combat_participants SET is_active = 0 WHERE encounter_id = ?',
        (encounter_id,)
    )
    # Оновлюємо статус бою
    db.execute(
        'UPDATE combat_encounters SET status = ? WHERE id = ?',
        ('completed', encounter_id)
    )
    db.commit()
    return jsonify({'message': 'Бій завершено'}), 200
# API для монстрів
@app.route('/api/monsters', methods=['GET'])
@login_required
def get_monsters():
    # Перевіряємо, чи потрібно використовувати API
    use_dnd_api = request.args.get('use_dnd_api', 'true').lower() == 'true'
    # Отримуємо параметри для фільтрації
    challenge_rating = request.args.get('cr')
    search_query = request.args.get('search')
    if use_dnd_api:
        # Отримуємо список монстрів з D&D 5e API
        monsters_data = fetch_from_dnd_api('monsters')
        if monsters_data and 'results' in monsters_data:
            result = []
            # Застосовуємо фільтрацію та пошук
            for monster_info in monsters_data['results']:
                # Обмежуємо кількість для прикладу
                if len(result) >= 6:
                    break
                # Перевіряємо пошуковий запит
                if search_query and search_query.lower() not in monster_info['name'].lower():
                    continue
```

```
                # Отримуємо деталі монстра
                monster_data = fetch_from_dnd_api(f"monsters/{monster_info['index']}")
                if not monster_data:
                    continue
                # Перевіряємо CR, якщо вказано
                if challenge_rating and monster_data.get('challenge_rating') !=
float(challenge_rating):
                    continue
                result.append({
                    'id': monster_data.get('index', ''),
                    'name': monster_data.get('name', ''),
                    'type': monster_data.get('type', ''),
                    'size': monster_data.get('size', ''),
                    'alignment': monster_data.get('alignment', ''),
                    'armor_class': monster_data.get('armor_class', 0),
                    'hit_points': monster_data.get('hit_points', 0),
                    'hit_dice': monster_data.get('hit_dice', ''),
                    'challenge_rating': monster_data.get('challenge_rating', 0),
                    'speed': monster_data.get('speed', {}),
                    'abilities': {
                        'str': monster_data.get('strength', 10),
                        'dex': monster_data.get('dexterity', 10),
                        'con': monster_data.get('constitution', 10),
                        'int': monster_data.get('intelligence', 10),
                        'wis': monster_data.get('wisdom', 10),
                        'cha': monster_data.get('charisma', 10)
                    }
                })
                # Невелика пауза, щоб не перевантажувати API
                time.sleep(0.01)
            return jsonify(result)
    # Якщо не використовуємо API або виникла помилка, повертаємо помилку
    return jsonify({'error': 'Локальна база даних монстрів відсутня. Використовуйте D&D 5e
API.'}), 404
@app.route('/api/monsters/<monster_id>', methods=['GET'])
@login_required
def get_monster_details(monster_id):
    # Перевіряємо, чи потрібно використовувати API
    use_dnd_api = request.args.get('use_dnd_api', 'true').lower() == 'true'
    if use_dnd_api:
        # Отримуємо деталі монстра з D&D 5e API
        monster_data = fetch_from_dnd_api(f"monsters/{monster_id}")
        if monster_data:
            # Формуємо дії монстра
            actions = []
            for action in monster_data.get('actions', []):
                actions.append({
                    'name': action.get('name', ''),
                    'description': action.get('desc', '')
                })
            # Формуємо спеціальні здібності
            special_abilities = []
            for ability in monster_data.get('special_abilities', []):
                special_abilities.append({
                    'name': ability.get('name', ''),
                    'description': ability.get('desc', '')
                })
            # Формуємо результат
            result = {
                'id': monster_data.get('index', ''),
                'name': monster_data.get('name', ''),
                'type': monster_data.get('type', ''),
                'size': monster_data.get('size', ''),
                'alignment': monster_data.get('alignment', ''),
                'armor_class': monster_data.get('armor_class', 0),
```

```
                'hit_points': monster_data.get('hit_points', 0),
                'hit_dice': monster_data.get('hit_dice', ''),
                'challenge_rating': monster_data.get('challenge_rating', 0),
                'xp': monster_data.get('xp', 0),
                'speed': monster_data.get('speed', {}),
                'abilities': {
                    'str': monster_data.get('strength', 10),
                    'dex': monster_data.get('dexterity', 10),
                    'con': monster_data.get('constitution', 10),
                    'int': monster_data.get('intelligence', 10),
                    'wis': monster_data.get('wisdom', 10),
                    'cha': monster_data.get('charisma', 10)
                },
                'saves': monster_data.get('saving_throws', []),
                'skills': monster_data.get('skills', {}),
                'damage_vulnerabilities': monster_data.get('damage_vulnerabilities', []),
                'damage_resistances': monster_data.get('damage_resistances', []),
                'damage_immunities': monster_data.get('damage_immunities', []),
                'condition_immunities': monster_data.get('condition_immunities', []),
                'senses': monster_data.get('senses', {}),
                'languages': monster_data.get('languages', ''),
                'actions': actions,
                'special_abilities': special_abilities,
                'legendary_actions': monster_data.get('legendary_actions', [])
            }
            return jsonify(result)
    # Якщо не використовуємо API або виникла помилка, повертаємо помилку
    return jsonify({'error': 'Монстра не знайдено'}), 404
# Запуск серверу
if __name__ == '__main__':
    # Перевіряємо, чи існує база даних
    if not os.path.exists(DATABASE):
        with app.app_context():
            init_db()
    app.run(debug=True)
```

Додаток Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

Розробка вебзастосунку для інтерактивного дослідження класів героїв у настільній грі "Dungeons and Dragons"

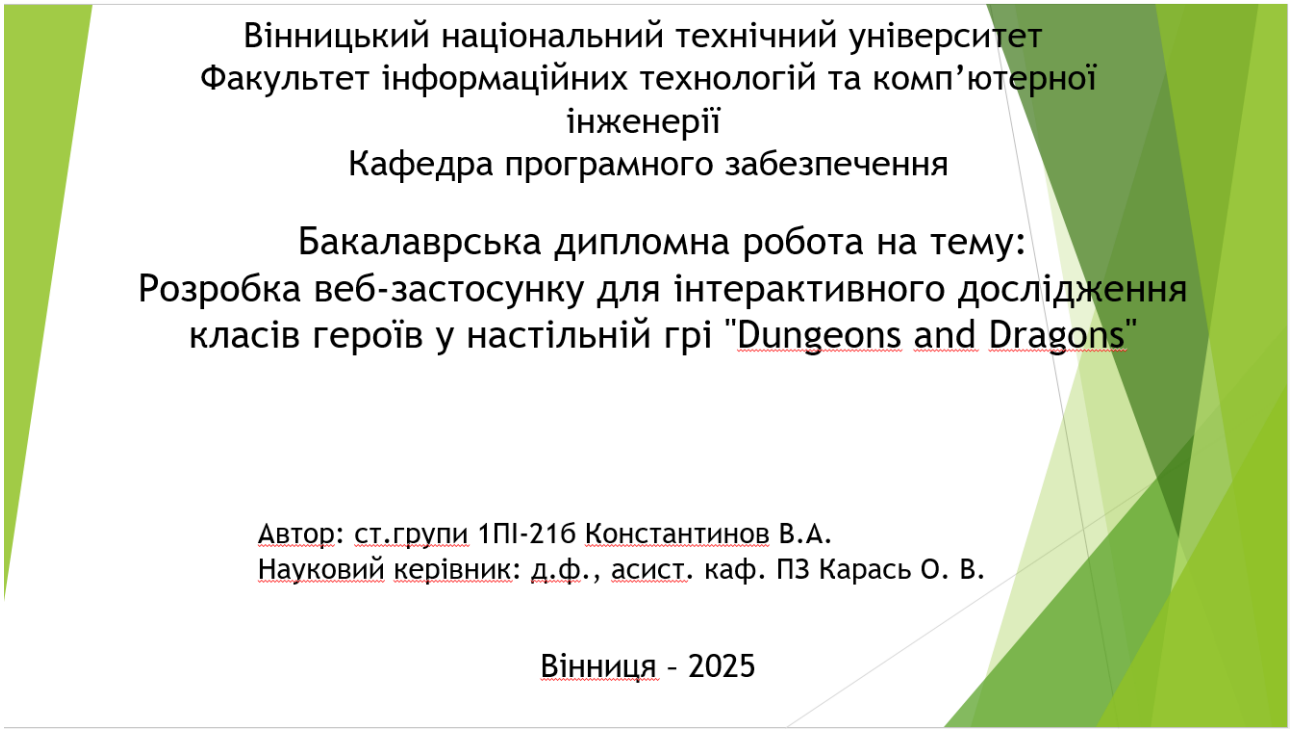


Рисунок Г.1 – Титульний слайд

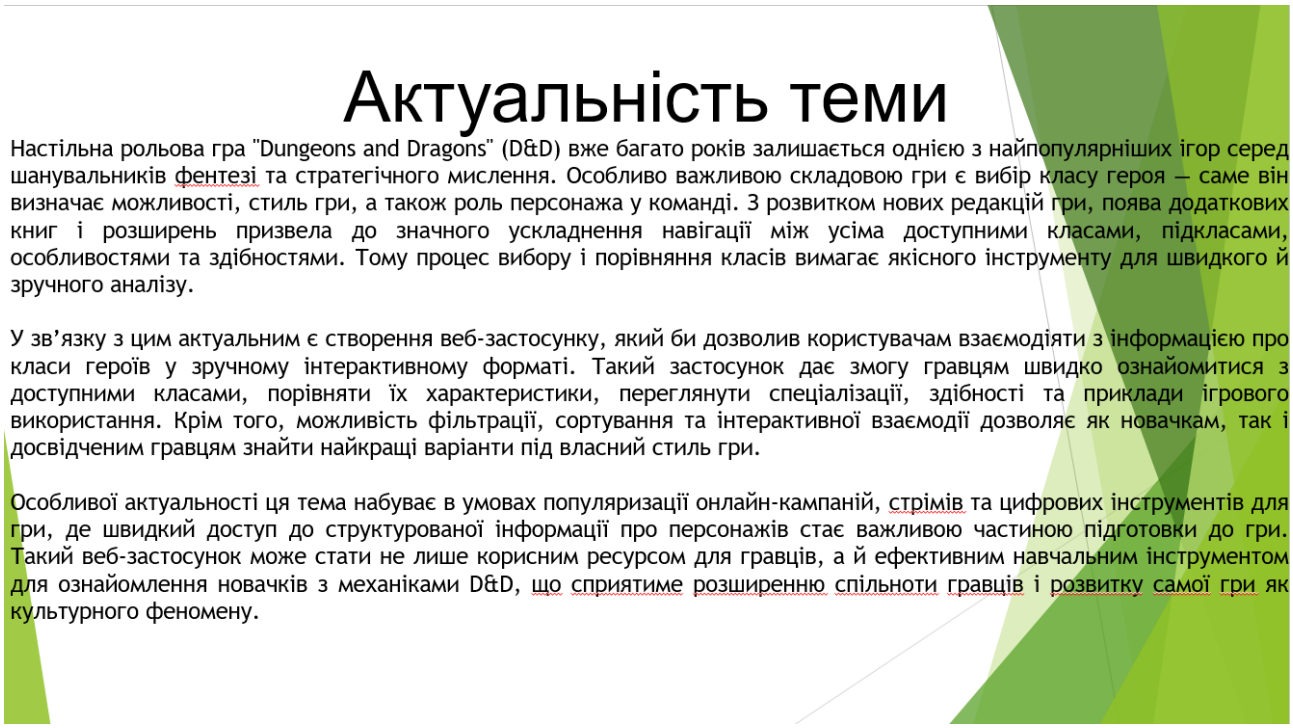


Рисунок Г.2 – Актуальність теми

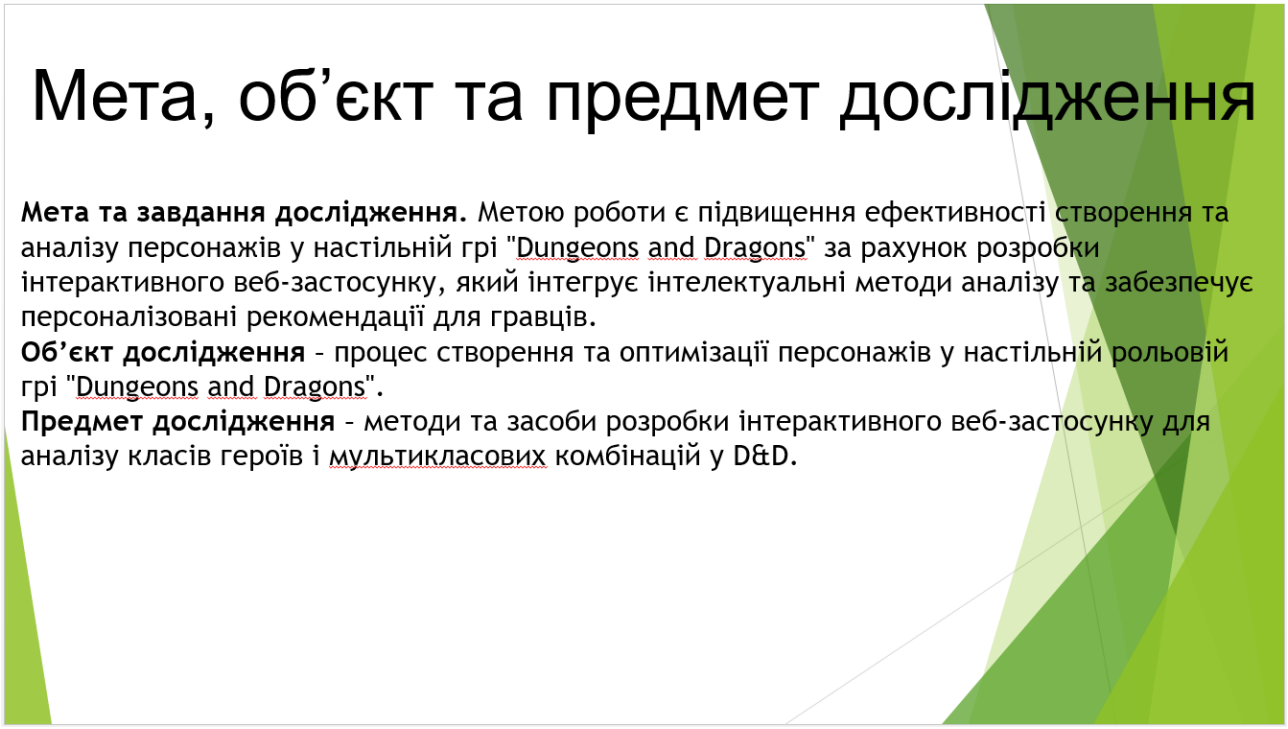


Рисунок Г.3 – Мета, об’єкт та предмет дослідження

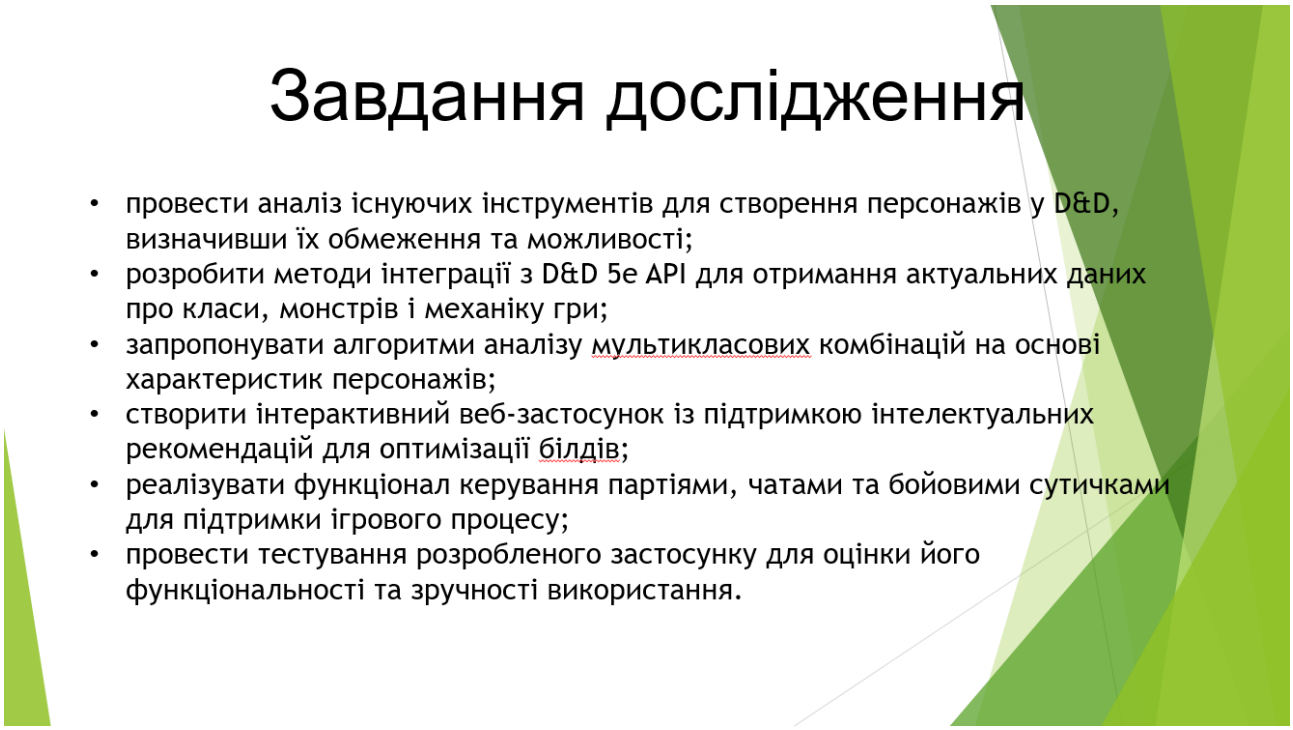


Рисунок Г.4 – Завдання дослідження

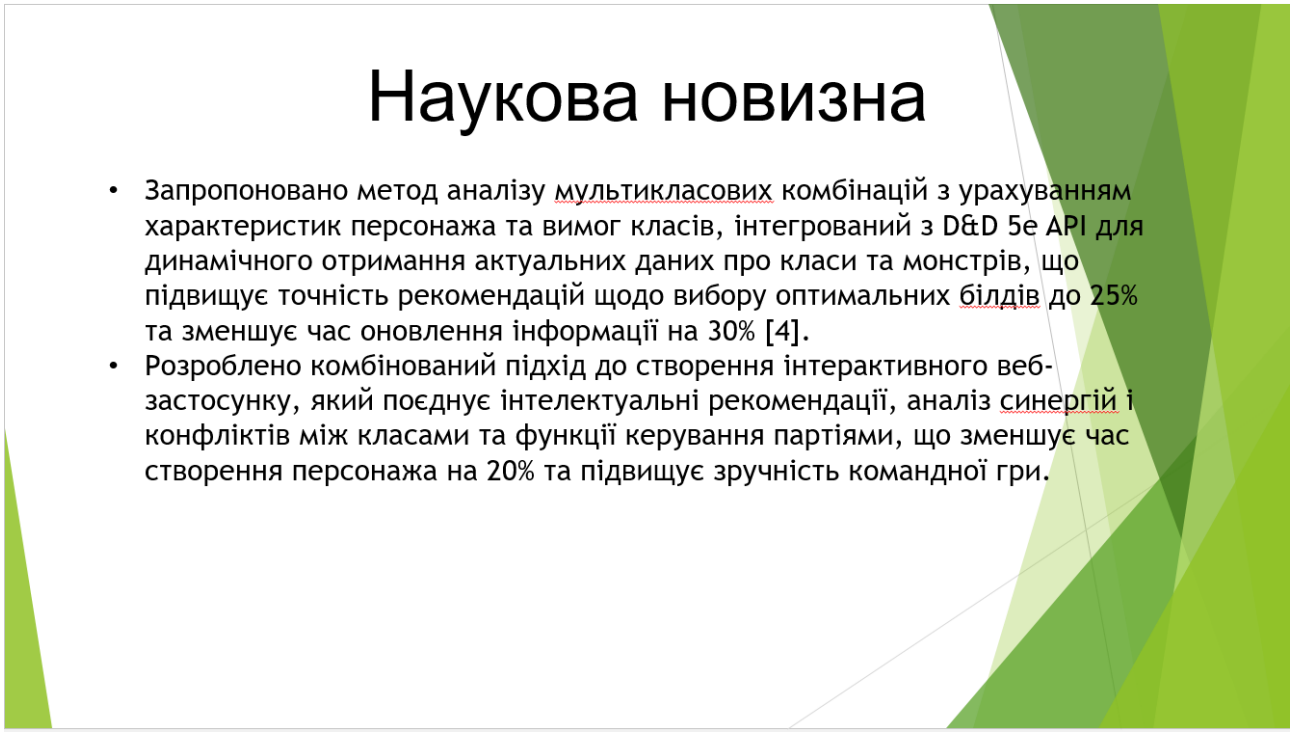


Рисунок Г.5 – Наукова новизна

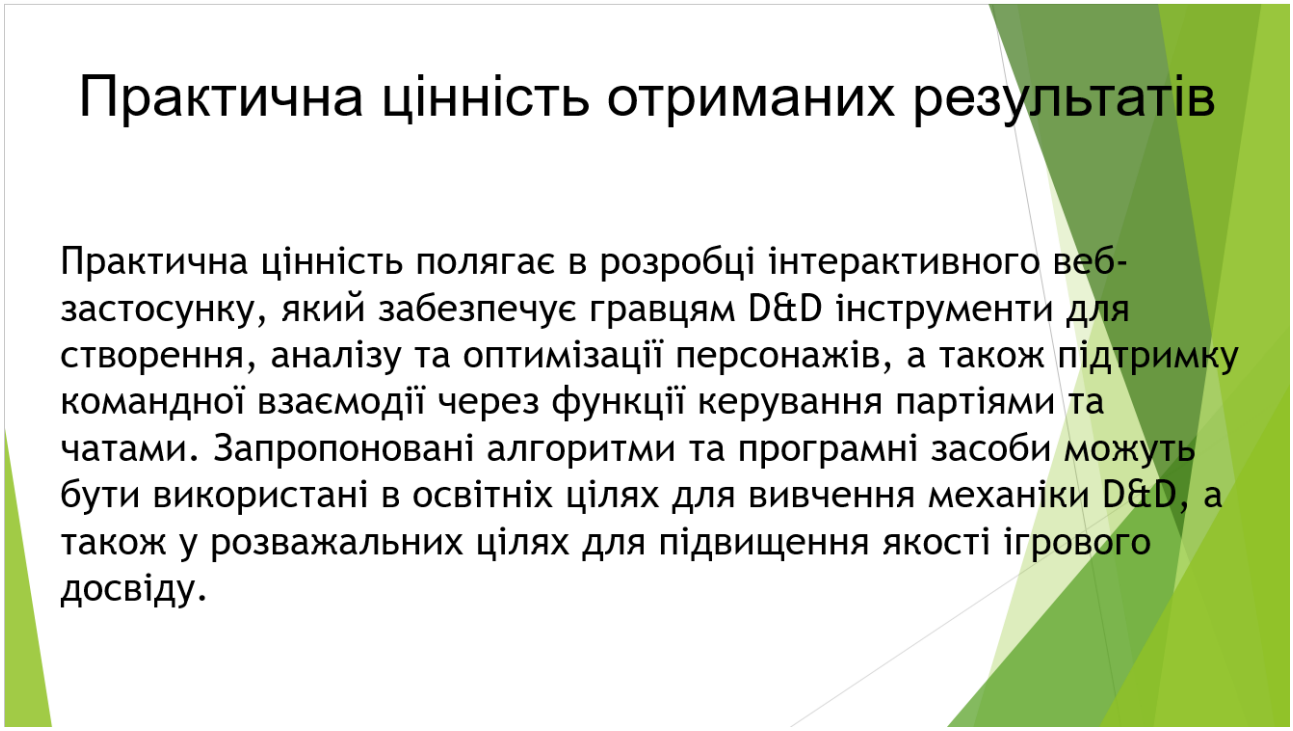
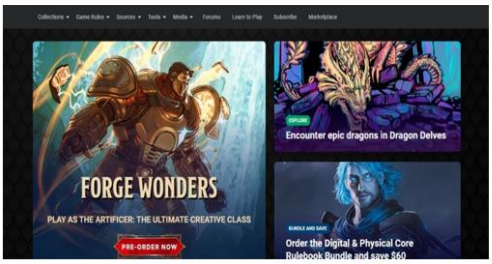
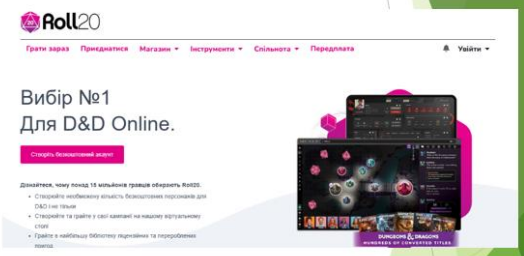


Рисунок Г.6 – Практична цінність отриманих результатів

Аналоги



D&D Beyond



Roll20

Рисунок Г.7 – Аналоги

Порівняльний аналіз аналогів

Критерії	D&D Beyond	Roll20	Fantasy Grounds	Pathbuilder 2e	Foundry VTT	Aurora Builder	Власний модуль
Точність даних	+	+	+	+	+	+	+
Підтримка мультикласу	+	+	+	+	+	+	+
Автоматизація аналізу	+	-	+	-	-	-	+
Доступ до офіційних правил	+	+	+	-	-	+	+
Зручність інтерфейсу	-	+	-	+	+	-	+
Загальна оцінка	4	4	4	3	3	3	5

Рисунок Г.8 – Порівняльний аналіз аналогів

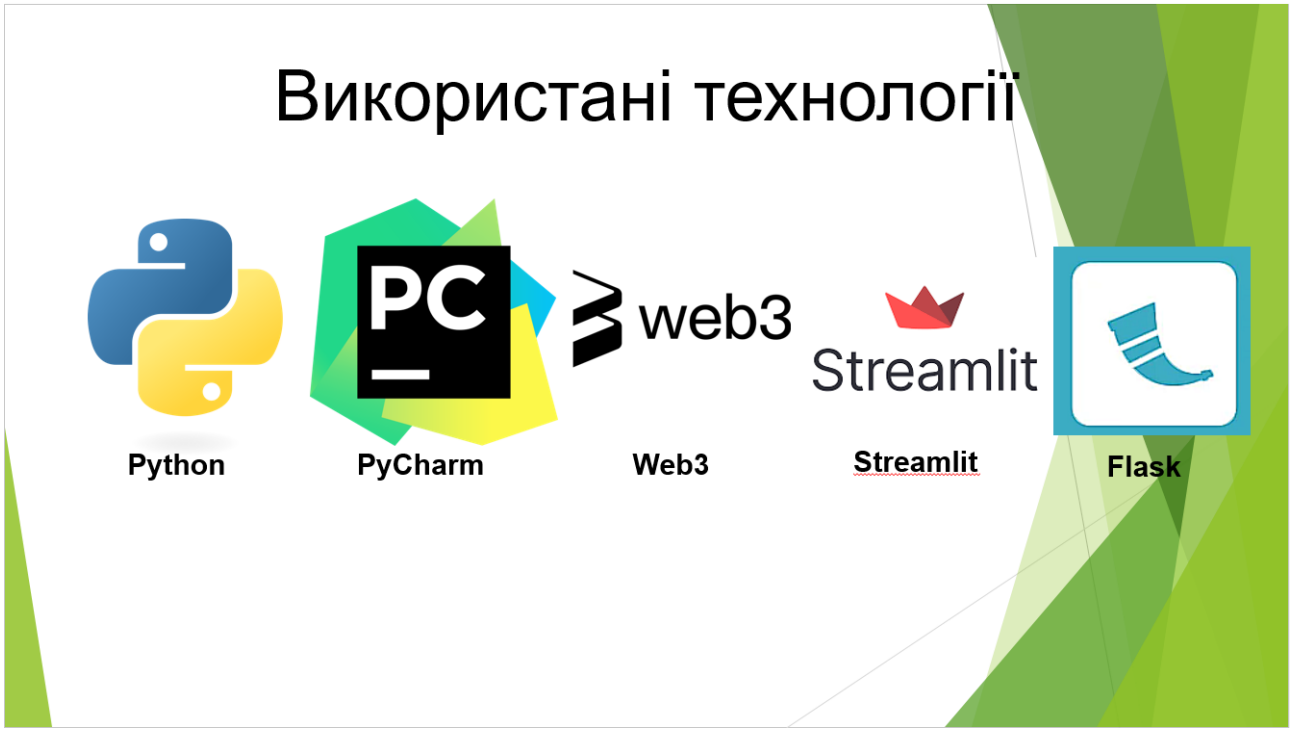


Рисунок Г.9 – Використані технології

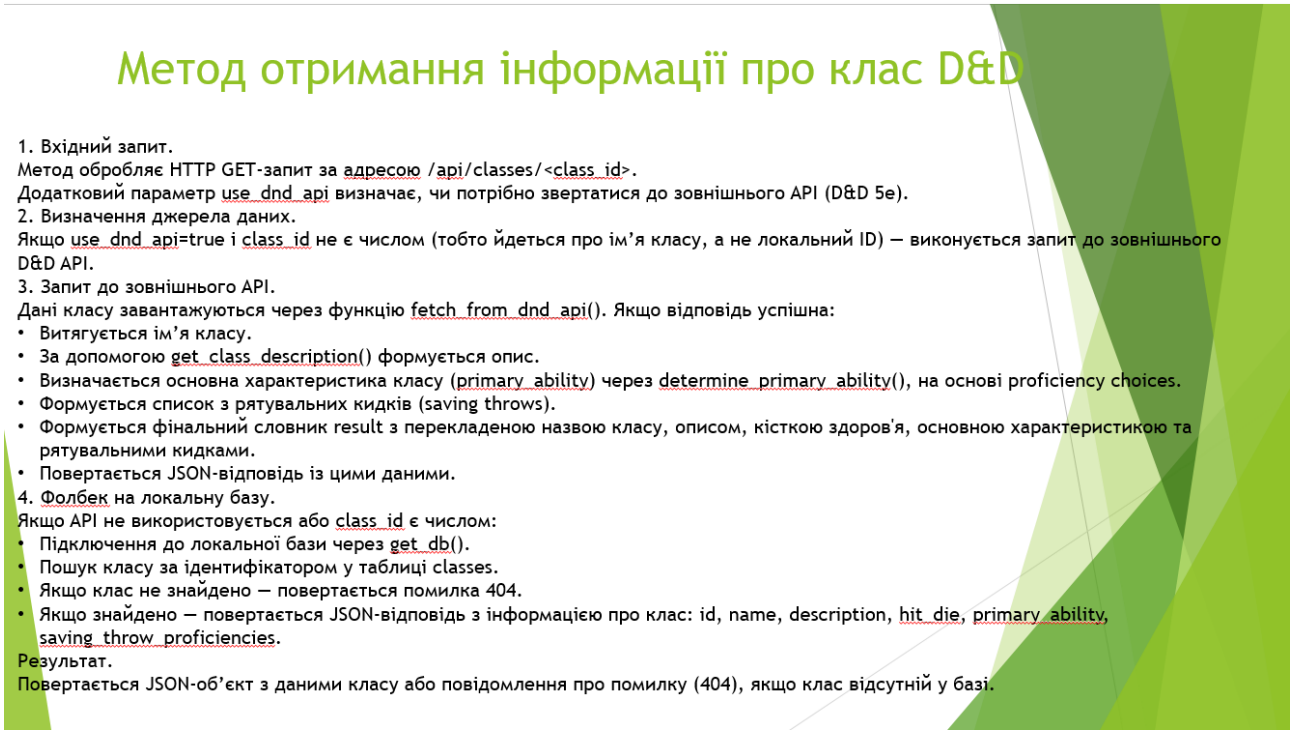


Рисунок Г.10 – Метод отримання інформації про клас D&D

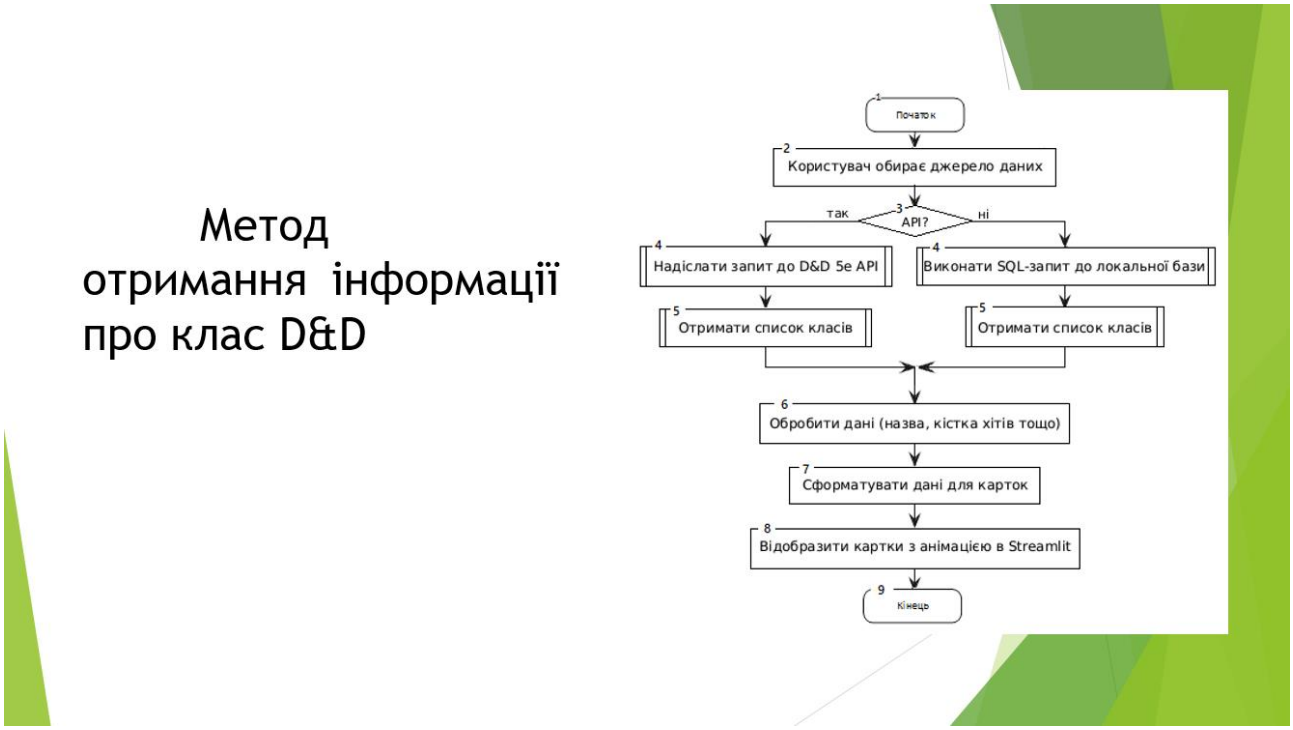


Рисунок Г.11 – Метод отримання інформації про клас D&D

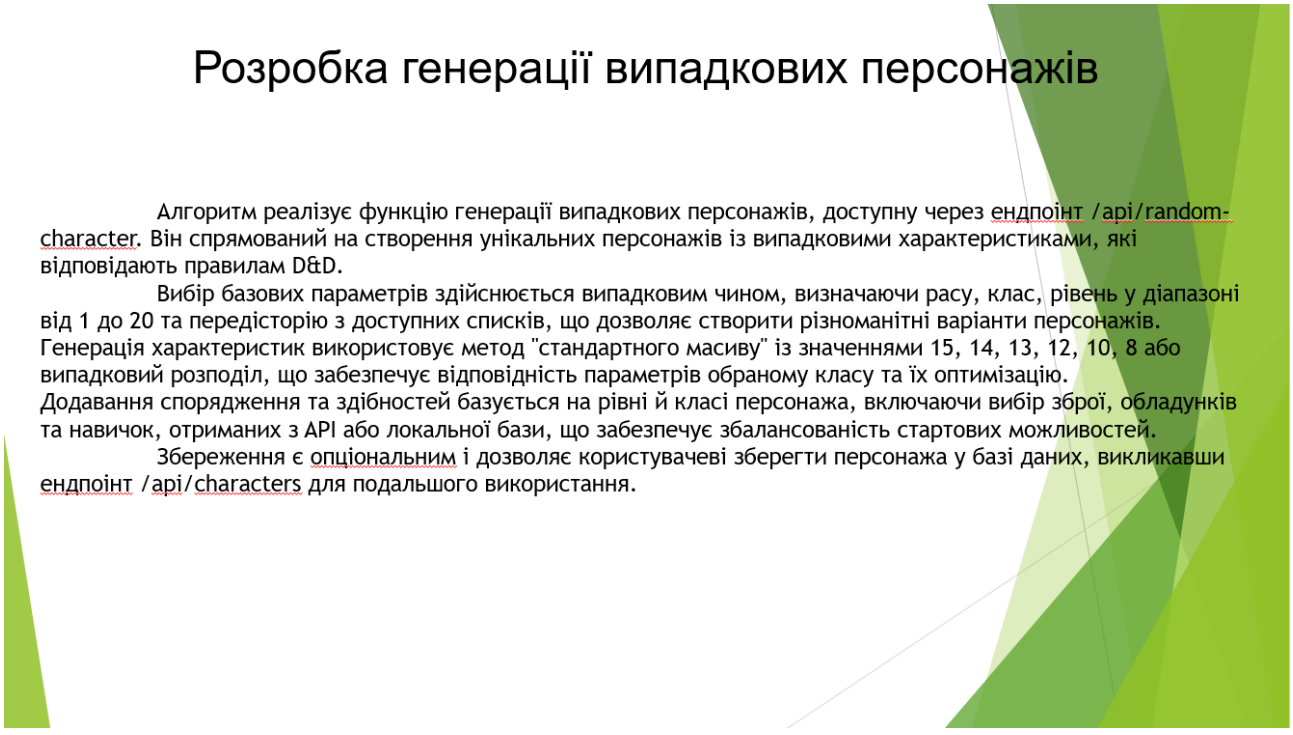


Рисунок Г.12 – Метод отримання інформації про клас D&D

Рисунок Г.13 – Блок-схема алгоритму генерації випадкових персонажів

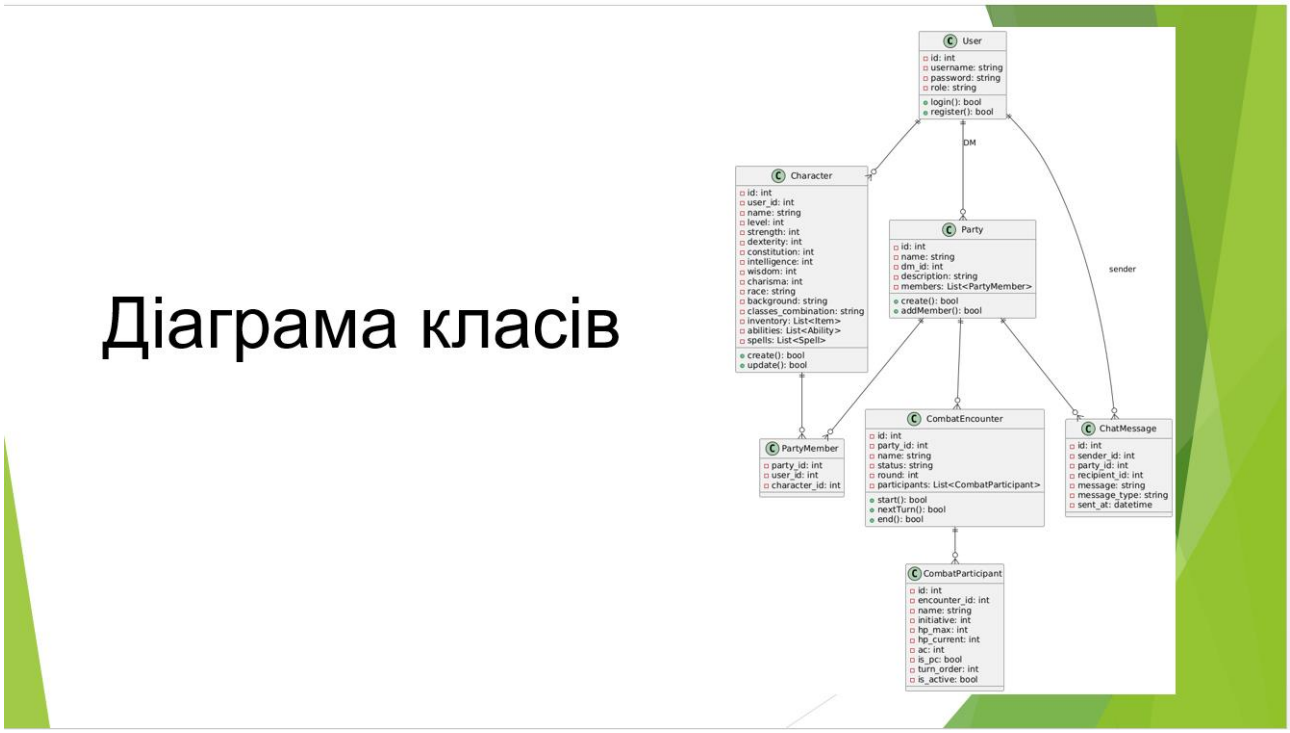


Рисунок Г.14 – Діаграма класів

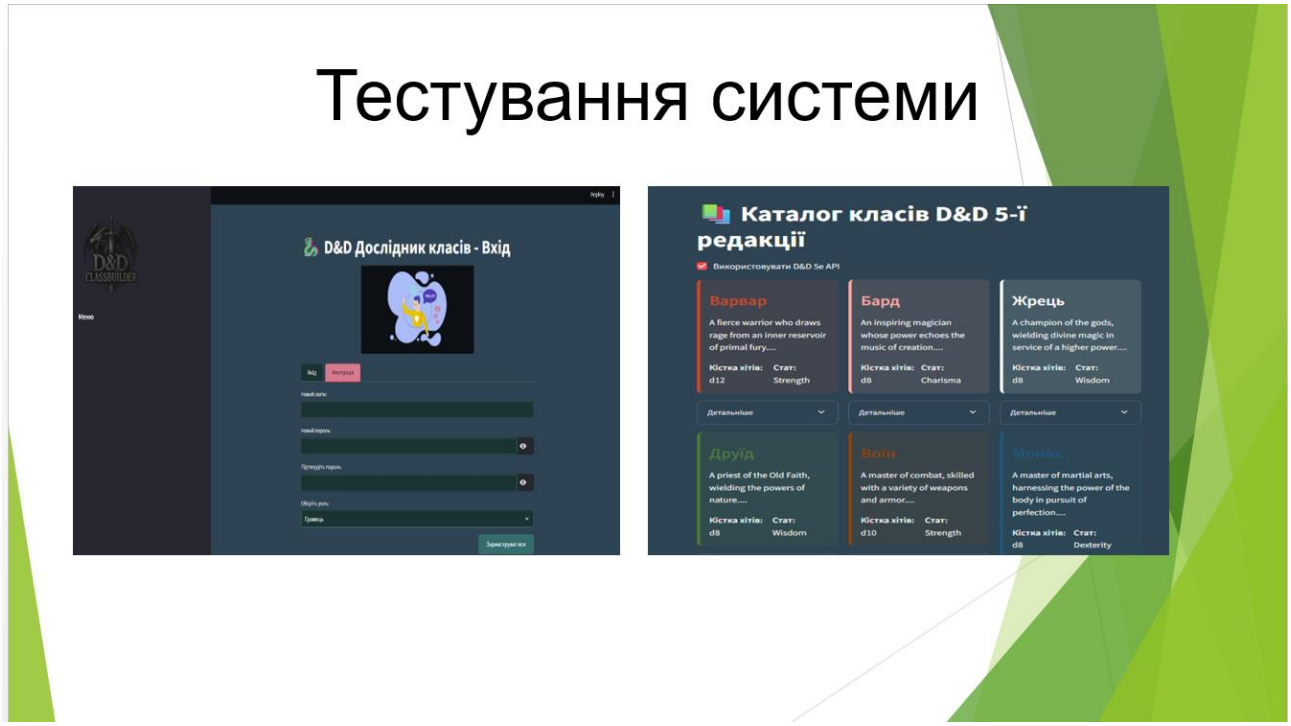


Рисунок Г.15 – Тестування системи

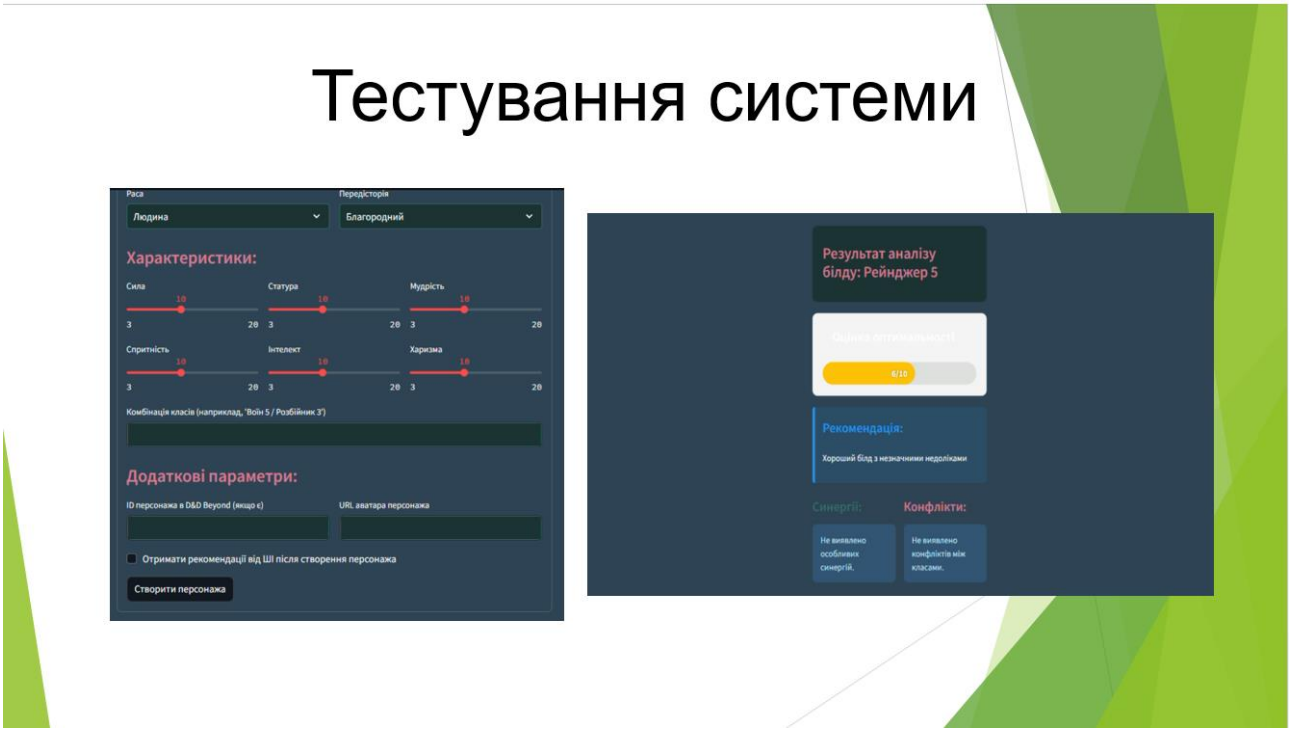


Рисунок Г.16 – Тестування системи

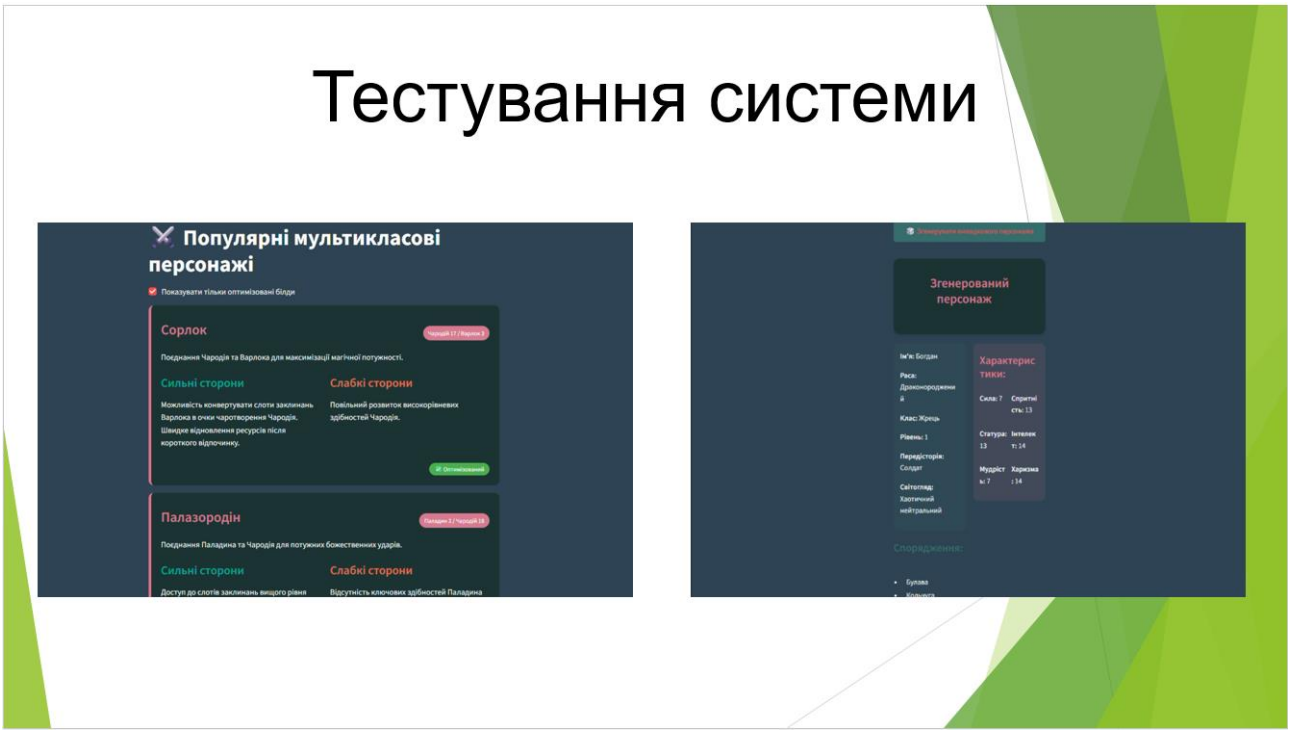


Рисунок Г.17 – Тестування системи

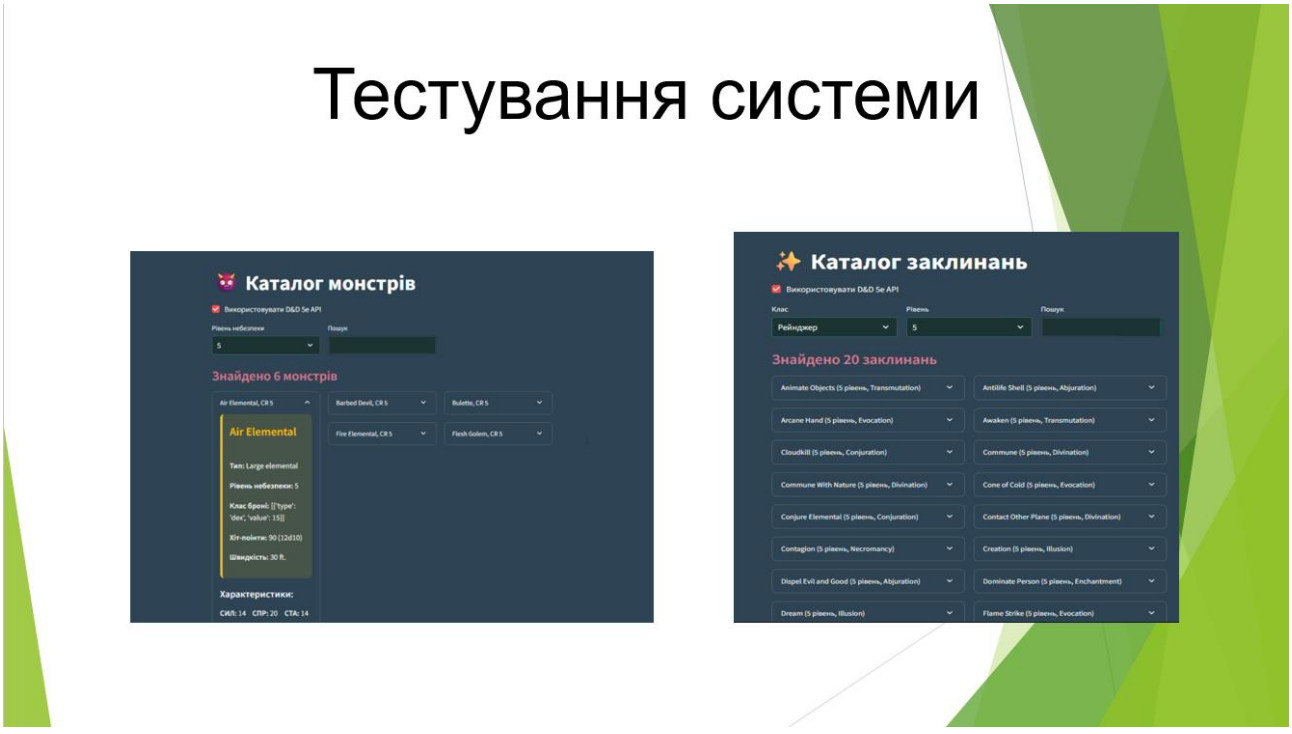


Рисунок Г.18 – Тестування системи

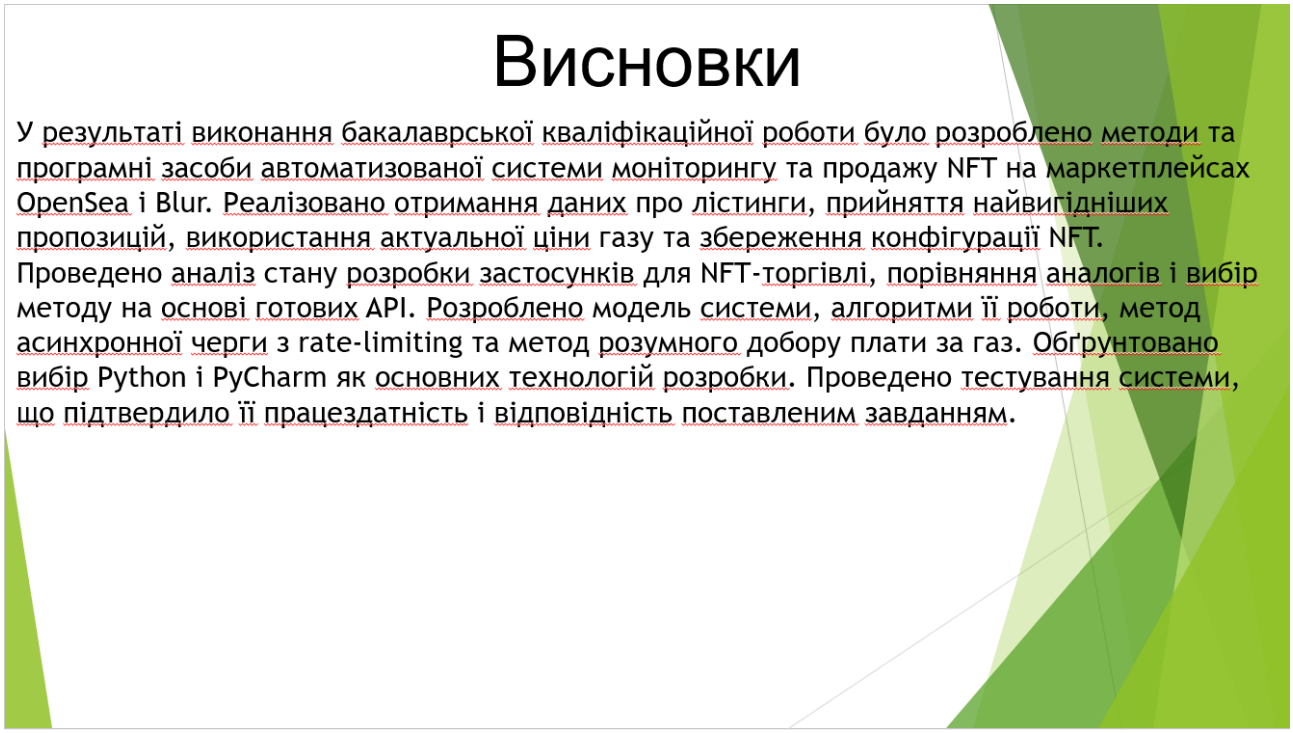


Рисунок Г.19 – Висновки

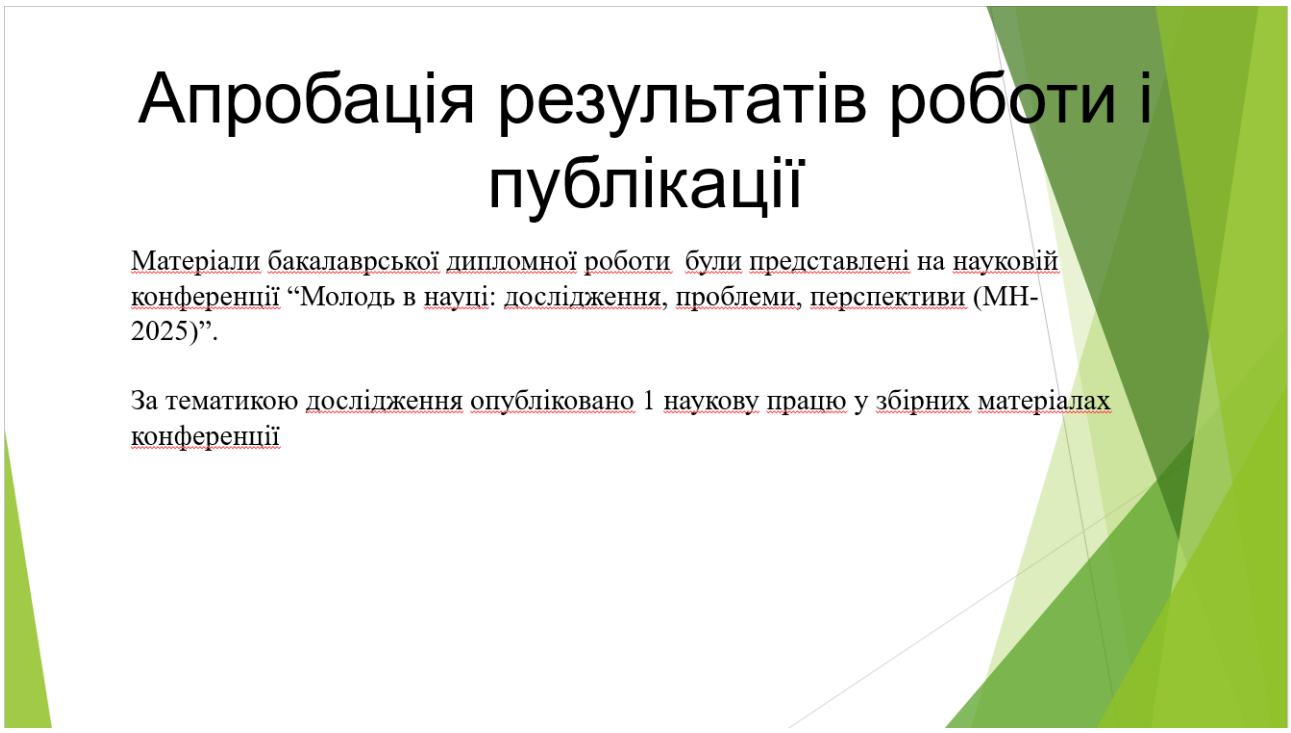


Рисунок Г.19 – Апробація результатів роботи і публікації

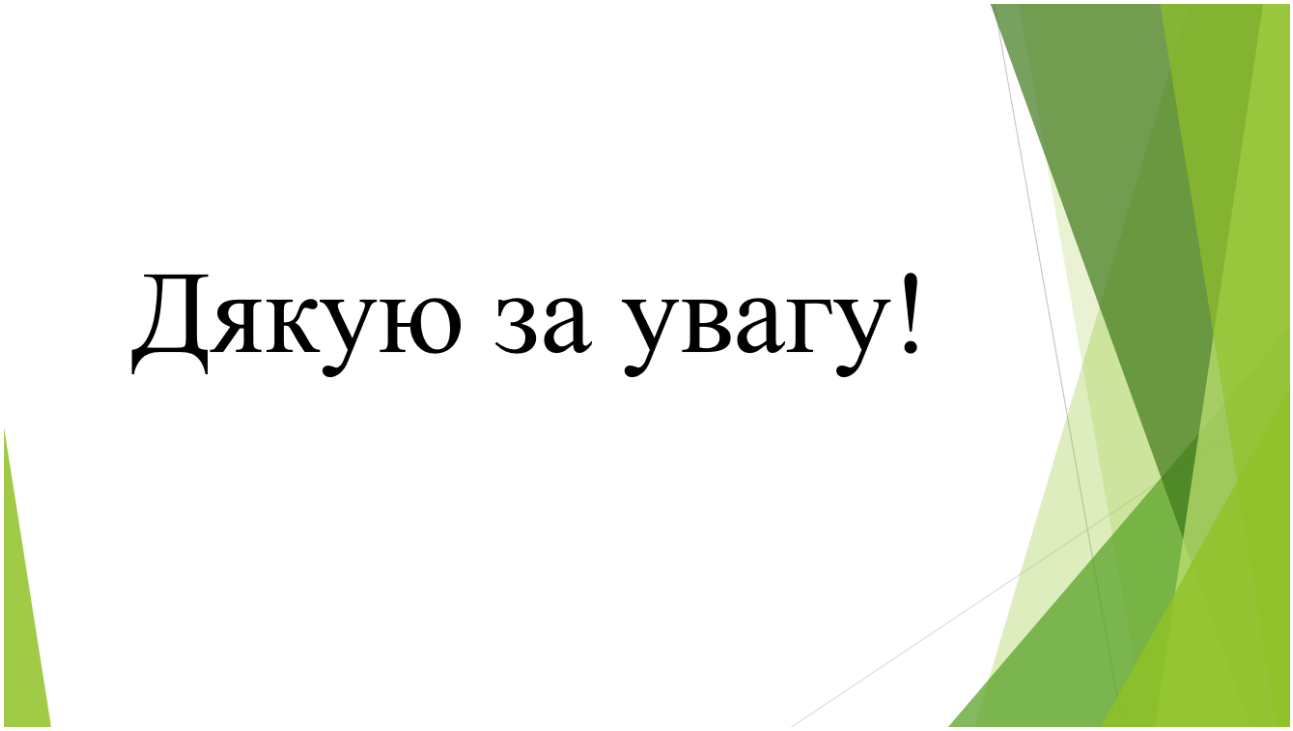


Рисунок Г.19 – Заключення