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

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмних засобів для сприяння розвитку логічних здібностей на прикладі настільної гри Dangeons & Dragons»

Виконав: студент IV курсу
групи 5ПІ- 21Б
спеціальності
121 – Інженерія програмного забезпечення
Байдалюк В. О.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Ткаченко О.М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Кадук О. В.
(прізвище та ініціали)

Допущено до захисту

Завідувач кафедри ПЗ Романюк О. Н.
«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
д.т.н., професор
О. Н. Романюк_____
«21» березня 2025р.

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

Байдалюку Владиславу Олеговичу

1. Тема роботи – Розробка програмних засобів для сприяння розвитку логічних здібностей на прикладі настільної гри Dangeons & Dragons.

Керівник роботи: Ткаченко Олександр Миколайович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: Вихідні дані до роботи: середовище розробки Visual Studio Code, мова розробки – JavaScript, операційна система – Windows.

4.Зміст текстової частини: вступ; аналіз розвитку технологій веб-систем для гравців та аналіз допоміжних інструментів; розробка структури та алгоритмів застосунку; розробка програмних модулів для реалізації калькулятора та рандомайзера параметрів; тестування програми; висновки; список використаних джерел; додатки.

5. Перелік ілюстративної частини: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, новизна, Порівняльний аналіз з аналогами; Діаграма діяльності вебсайту; Блок-схема алгоритму роботи програмного модуля, Удосконалений метод випадкової зміни, модуль для збереження персонажів, модуль для калькулятора складності; Розробка інтерфейсу вебдодатку; Тестування в реальних умовах; Апробація та публікації результатів роботи; Висновки; кінцевий слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ткаченко О.М., к.т.н., доцент кафедри ПЗ	24.03.25	01.04.25

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 вебзастосунків для гравців та аналіз допоміжних інструментів	25.03.2025 – 07.04.25	вик
2.	Реалізація методів створення та обробки персонажів, розрахунку рівня небезпеки та обробка архітектури програмної системи	08.04.25 – 22.04.25	вик
3.	Розробка інтерактивного вебзастосунку обробки та розрахунку створених персонажів	02.05.25 – 01.05.25	вик
4.	Тестування програми	02.05.25 – 14.05. 25	вик

Студент _____ (підпис) **Байдалюк В.О.** (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Ткаченко О.М.** (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 93 сторінок формату А4, на яких представлено 18 рисунків, 5 таблиці та список використаних джерел, що налічує 20 найменувань.

У бакалаврській кваліфікаційній роботі було проведено аналіз існуючих програмних систем для створення ігрових сесій. Розглянуто наявні аналоги, визначено їхні переваги та недоліки, що дозволило обґрунтувати доцільність розробки власного веб-застосунку.

Розроблений веб-застосунок покликаний автоматизувати ключові аспекти організації ігрової сесії у настільній грі, спрямованій на розвиток логічного мислення. Система надає користувачеві можливість швидко створювати персонажів, керувати їх характеристиками, формувати сценарії та обчислювати рівень складності ігрових подій. Застосунок підтримує зберігання внутрішніх даних у структурованому форматі, реалізує зручний пошук і пропонує візуально зрозумілий інтерфейс, адаптований під потреби користувачів.

Веб-застосунок реалізовано за допомогою мови програмування JavaScript із використанням бібліотеки React, яка забезпечує компонентний підхід до побудови інтерфейсу. Зберігання даних реалізовано у форматі SQL, що дозволяє легко зчитувати, зберігати та модифікувати внутрішні конфігурації. Розробка здійснювалась у середовищі Visual Studio Code, яке надало всі необхідні інструменти для керування проєктом, включно з контролем версій, автодоповненням та інтеграцією з React-бібліотеками.

У результаті виконання бакалаврської кваліфікаційної роботи створено веб-застосунок, що покращує організацію настільної гри та сприяє розвитку логічного мислення завдяки інтуїтивно зрозумілому інтерфейсу та автоматизованим функціям. Отримані результати можуть бути використані для подальшого вдосконалення цифрових рішень у сфері освітніх ігор.

Ключові слова: застосунок, Dangeons & Dragons, майстер, персонаж.

ANNOTATION

The bachelor's qualification thesis consists of 93 A4 pages, featuring 18 figures, 5 tables, and a list of 20 references.

This work presents an analysis of existing software systems designed for organizing game sessions. The study examines available analogues, identifies their strengths and weaknesses, and justifies the need for developing a custom web application.

The developed web application is intended to automate key aspects of organizing a tabletop game session focused on developing logical thinking. The system enables users to quickly create characters, manage their attributes, generate scenarios, and calculate the difficulty level of game events. It supports structured internal data storage, provides a convenient search function, and offers a visually clear interface tailored to user needs.

The web application was developed using the JavaScript programming language and the React library, which ensures a component-based approach to interface design. Data storage is implemented in SQL format, allowing for easy reading, saving, and modification of internal configurations. Development was carried out in the Visual Studio Code environment, which provided all necessary project management tools, including version control, autocomplete, and integration with React libraries.

As a result of this bachelor's qualification thesis, a web application was created that enhances the organization of tabletop games and fosters the development of logical thinking through an intuitive interface and automated features. The outcomes of this work can be used to further improve digital solutions in the field of educational games.

Keywords: application, Dungeons & Dragons, game master, character.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження

Розробка програмного засобу, орієнтованого на підтримку розвитку логічного мислення шляхом автоматизації процесів у настільних іграх типу Dungeons & Dragons, є актуальною в контексті зростаючого інтересу до освітніх ігор, гейміфікації навчання та використання цифрових інструментів у неформальній освіті. Dungeons & Dragons відома своєю складною системою характеристик, багаторівневими механіками бою, розвитку персонажів та інтерактивною взаємодією між гравцями. Ці особливості роблять гру придатною для формування логічних зв’язків, стратегічного мислення та навичок прийняття рішень. Проте підготовка ігрової сесії потребує багато часу, зокрема на обчислення рівня небезпеки, підбір статистик персонажів або монстрів, формування балансу між гравцями та супротивниками.

У зв’язку з цим активно розробляються різноманітні вебзастосунки та утиліти, які покликані спростити процеси створення ігрового контенту. Серед них можна виділити генератори пригод, бази даних монстрів, редактори персонажів і калькулятори бойових параметрів. Проте чимало з цих засобів мають низку обмежень: обмежений функціонал, відсутність підтримки індивідуальних сценаріїв, недостатньо гнучкі системи редагування або орієнтація виключно на англомовну аудиторію. Крім того, деякі з них не дозволяють одночасно здійснювати розрахунки, зберігати сесії чи адаптувати матеріал під локальні правила. У результаті гравці та майстри змушені користуватися кількома окремими інструментами, що уповільнює підготовку та ускладнює організацію.

Оскільки сучасні настільні ігри дедалі частіше використовуються не лише як форма дозвілля, а й як інструмент для навчання та розвитку, виникає потреба в цифрових засобах, які здатні спростити організаційні етапи. Особливу увагу заслуговують модулі, що дають змогу проводити розрахунки рівня складності зіткнень, генерувати параметри персонажів за заданими критеріями, автоматично перевіряти баланс сили сторін, а також зберігати та повторно використовувати

створені шаблони.

Таким чином, питання створення зручного вебзастосунку, що пришвидшує підготовку ігрових сценаріїв та дозволяє автоматизувати окремі етапи підрахунків, є актуальним і заслуговує детального дослідження.

Зв’язок роботи з науковими програмами, планами, темами. Дослідження було проведено згідно з планом наукових досліджень на кафедрі програмного забезпечення.

Мета та задачі дослідження. Метою бакалаврської кваліфікаційної роботи є зменшення часу створення ігрової сесії настільної гри Dangeons & Dragons за рахунок автоматизаціїї процесу створення персонажів.

Для досягнення поставленої мети необхідно розв’язати наступні задачі.

- організувати збір даних із різних веб джерел (сайти для гравців, бестіарії);

- проаналізувати методи створення, обробки, та підрахунку рівня небезпеки для персонажів;

- реалізувати методи створення обробки та підрахунку рівня небезпеки для персонажів;

- спроектувати базу даних для зберігання даних про створених персонажів;

- обрати мову програмування;

- обрати середовище розробки.

Об'єктом дослідження є процес розробки модулів комп'ютерної системи для організації часу та менеджменту для людей які беруть участь у розробці ігрової партії.

Предметом дослідження є методи та програмні засоби, що забезпечують ефективне створення атрибутів та виконання розрахунків з урахуванням індивідуальних потреб користувачів.

Методи дослідження. Під час проведення дослідження були застосовані такі методи: порівняльний аналіз функціональних можливостей програм-аналогів для виявлення їхніх обмежень і визначення завдань подальшої розробки; опрацювання теоретичних основ, розрахунок алгоритмічних формул для побудови й візуалізації

роботи програмного забезпечення; комп’ютерне моделювання з метою перевірки працездатності створених рішень і підтвердження обґрунтованості теоретичних положень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод заповнення атрибутів для процесу створення персонажа, в якому, на відміну від існуючих, реалізовано випадковий спосіб заповнення атрибутів з урахуванням можливого діапазону значеннь, що дозволить підвищити ефективність процесу створення контенту для ігрових кампаній.

Практичне значення отриманих результатів. Розроблено інтерактивну веб-систему створення ігрового контенту, який зменьшує когнітивне навантаження на розробника, а також часові витрати на створення сценарія на 8%.

Особистий внесок здобувача.

Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. Наукові праці опубліковано одноосібно.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати роботи було представлено на Міжнародній науково-практичній інтернет-конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2025) (Вінниця, 2025 р.).

Публікації. Результати роботи опубліковано в матеріалах конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2025) [1].

1 АНАЛІЗ ВЕБЗАСТОСУНКІВ ДЛЯ ГРАВЦІВ ТА ДОПОМІЖНИХ ІНСТРУМЕНТІВ ДЛЯ ЇХ СТВОРЕННЯ

1.1 Аналіз предметної області інструментів для гравців

Інструменти для гравців у контексті настільних рольових ігор (HPI), зокрема Dungeons & Dragons (D&D), – це елементи, засоби та цифрові або фізичні ресурси, що використовуються для ведення ігрового процесу, взаємодії між учасниками та підтримання занурення в ігровий світ. До таких інструментів належать: ігрові аркуші персонажів, довідники правил, генератори, карти, кубики, системи трекінгу характеристик, візуальні матеріали, музичне оформлення тощо[2].

Основними учасниками гри є:

Гравці – особи, які керують окремими персонажами, беручи участь у пригоді, виконуючи завдання та приймаючи рішення від імені свого героя.

Майстер гри (Dungeon Master, DM) – особа, яка веде гру, описує світ, контролює події, грає за всіх неігрових персонажів (NPC), а також судить наслідки дій гравців[3].

Основні категорії інструментів гри

Інструменти для настільних рольових ігор можна класифікувати за типами користувачів та функціональністю (табл. 1.1).

Таблиця 1.1 – Класифікація типів користувачів та функціональністю

Категорія інструментів	Тип інструменту	Опис
Універсальний інструмент	Генератори персонажів	Програмні засоби для автоматичного створення персонажа відповідно до правил гри.

Продовження таблиці 1.1

Категорія інструментів	Тип інструменту	Опис
Інструменти для майстра гри	Генератори пригод	Автоматизовані засоби для створення сюжетів, квестів, NPC, монстрів тощо.
	Менеджери кампаній	Системи для планування пригод, опису світу, створення неігрових персонажів та керування подіями.
Інструменти для гравців	Аркуші персонажів	Формалізовані таблиці для запису характеристик, умінь, спорядження, заклять та іншої інформації про персонажа.
	Трекери станів	Засоби для відстеження поточного рівня здоров’я, заклять, умовних ефектів та ресурсу персонажа.

В основі настільної рольової гри лежить взаємодія між гравцями та майстром гри: гравці приймають рішення, що впливають на хід історії, тоді як майстер модерує світ, реагує на їхні дії та задає умови випробувань. Така взаємодія реалізується за допомогою різних інструментів: майстер описує ситуації, персонажів і локації (усно або з використанням візуальних матеріалів), гравці повідомляють про дії своїх персонажів, кидають кубики та звертаються до аркушів персонажа, а майстер визначає наслідки дій, спираючись на правила гри, таблиці складності та характеристики супротивників.

Актуальність розробки програмного забезпечення для підтримки гравців у настільних рольових іграх полягає у потребі спростити процеси створення, редагування та управління персонажами. У класичних системах, таких як Dungeons & Dragons, створення персонажа передбачає опрацювання великої кількості даних:

вибір раси, класу, фонового опису, здібностей, заклять, спорядження, а також точний розрахунок характеристик та модифікаторів. Цей процес часто вимагає ручних обчислень або звернення до численних джерел, що створює труднощі як для новачків, так і для досвідчених гравців.

1.2 Порівняльний аналіз існуючих програмних систем для гравців

На сучасному ринку програмного забезпечення існує велика кількість інструментів та вебсайтів, призначених для підтримки процесу створення ігрових сесій, кампаній або логічних сценаріїв. Однак лише незначна частина з них включає практичні функції, пов’язані з автоматичними розрахунками, такими як оцінка складності бою чи балансування персонажів. Саме тому для здійснення порівняльного аналізу були відібрані чотири популярні застосунки, які, попри різну спеціалізацію, мають значення в контексті підтримки ігрових процесів: Obsidian, Kobold Fight Club, Donjon.bin.sh та Dungeon Scrawl. У подальшому ці інструменти буде порівняно за такими критеріями, як зручність використання, простота в освоєнні, наявність системи підрахунків бойової складності, а також можливості створення ігрового контенту — як з технічної, так і з творчої точки зору.

Obsidian [4] (рис. 1.1) — це текстовий редактор, орієнтований на створення та організацію знань у форматі зв’язаної бази даних. Користувачі можуть створювати завдання, отримувати за їх виконання віртуальні нагороди та прокачувати персонажа. Проте ключовим недоліком є відсутність вбудованих інструментів для автоматичних підрахунків, оцінки складності боїв або генерації

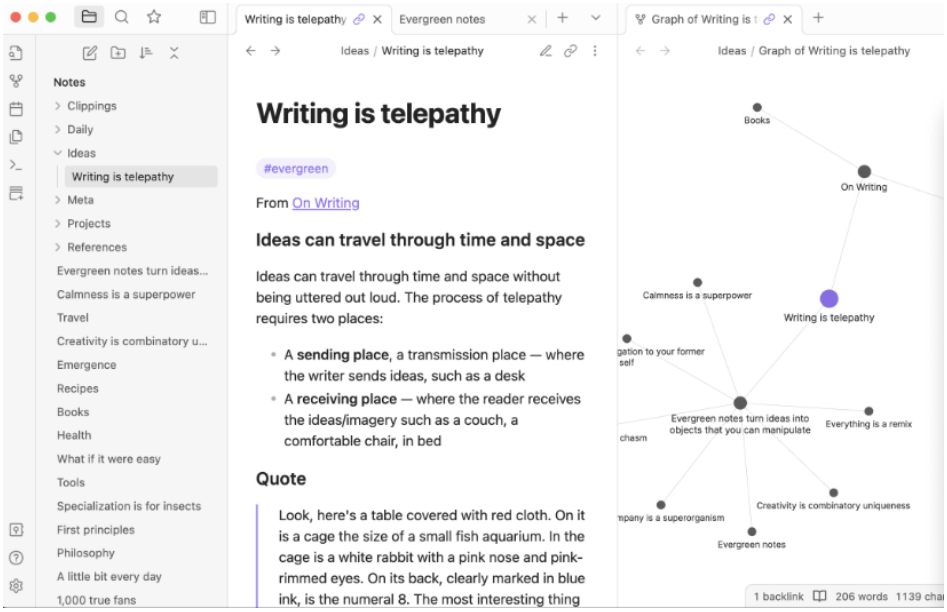

Рисунок 1.1 – Подання інтерфейсу програмного рішення Obsidian

Dungeon Scrawl [5] (рис. 1.2) — це вебзастосунок, орієнтований на швидке та інтуїтивне створення карт підземель і ігрових локацій. Користувачі можуть усього за кілька хвилин створити візуальну схему локації з можливістю експорту у високій якості. Додаток підтримує різні стилі карт і дозволяє застосовувати анотації. Водночас Dungeon Scrawl не включає жодних елементів роботи з персонажами, статистикою, сюжетами чи бойовими механіками, тому використовується переважно як допоміжний інструмент для візуального оформлення.

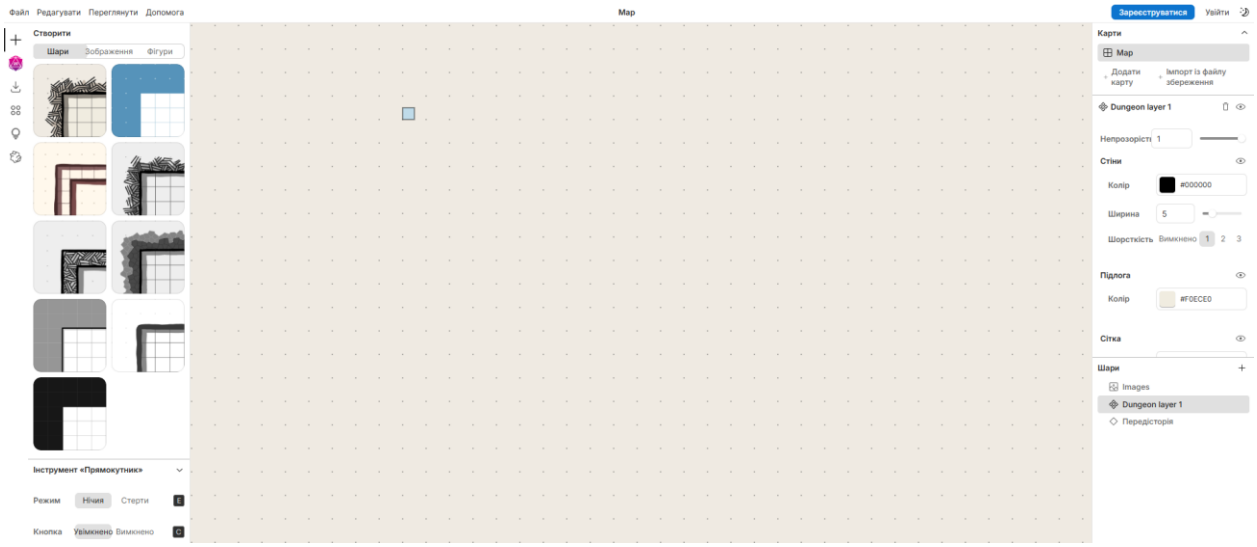

Рисунок 1.2 – Подання інтерфейсу програмного рішення Dungeon Scrawl

Donjon.bin.sh [6] (рис. 1.3) — це набір генераторів ігрового контенту, який охоплює фентезійної кампанії: від генерації підземель, імен, скарбів, заклять до створення повноцінних пригод. Його перевагою є швидкість отримання результату та широка варіативність готових шаблонів. Проте Donjon не дозволяє зберігати, структурувати або редагувати створений контент у довгостроковій перспективі, не має системи обліку складності боїв.

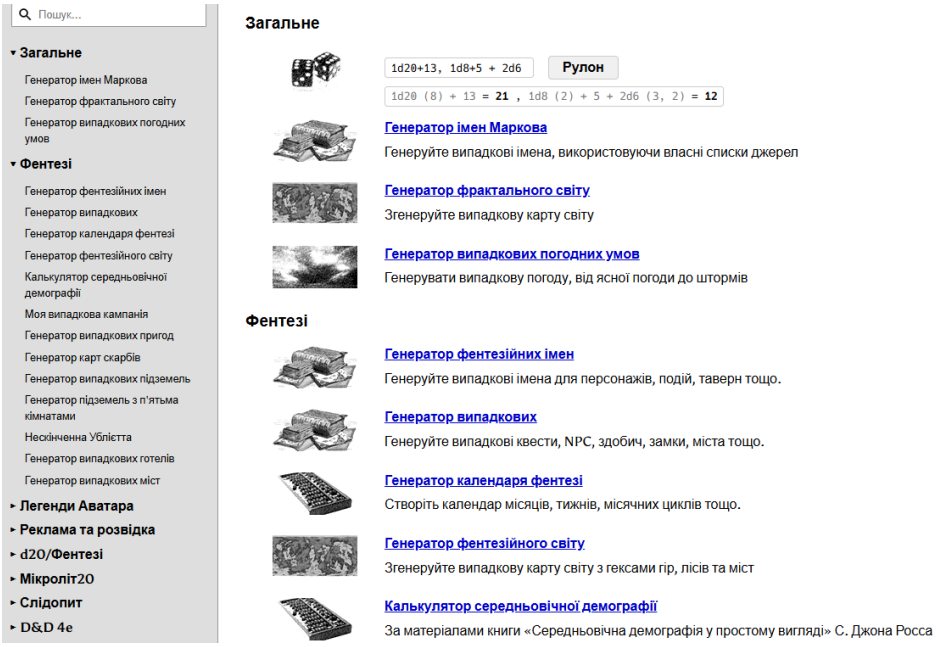

Рисунок 1.3 – Подання інтерфейсу програмного рішення Donjon.bin.sh

Kobold Fight Club [7] (рис. 1.4) — це онлайн-інструмент, створений спеціально для побудови бойових сцен у системі Dungeons & Dragons. Основна функціональність полягає в автоматичному підборі монстрів відповідно до рівня складності бою, кількості гравців та їхнього рівня. Сервіс дозволяє зручно оцінювати баланс битви й дає змогу майстрам швидко формувати зустрічі. Проте він має доволі вузьку спеціалізацію — інструмент орієнтований виключно на бої й не включає можливостей для збереження сюжетної інформації, візуалізації мап або створення гнучких записів. Крім того, підтримка нових редакцій або кастомного контенту є обмеженою.

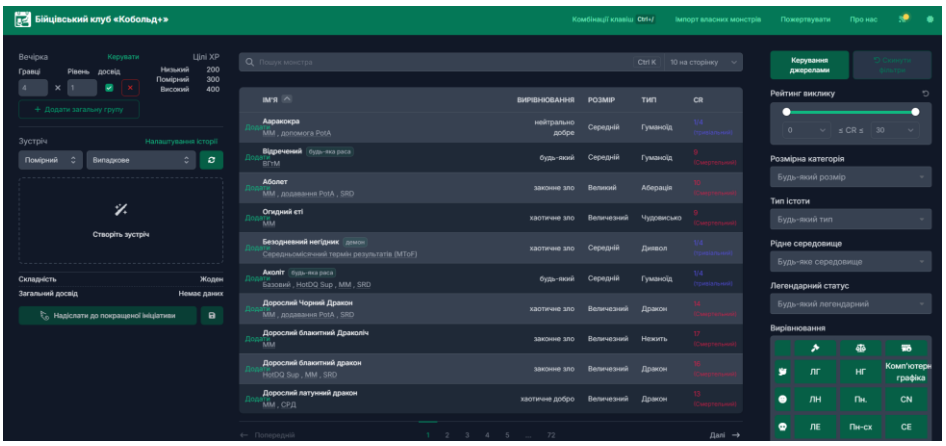

Рисунок 1.4 – Подання інтерфейсу програмного рішення Kobold Fight Club

Перш ніж переходити до розробки власного програмного рішення, доцільно здійснити порівняльний аналіз існуючих систем, що пропонують покращення процесу створення. Такий аналіз дозволяє виявити не лише сильні сторони популярних застосунків, але й визначити їхні обмеження в контексті використання користувачами. Виокремлення недоліків дозволяє сформулювати чіткі вимоги до майбутньої системи, уникнути повторення типових помилок та зосередити розробку на тих функціональних елементах, які справді покращують користувацький досвід для цільової аудиторії.

Результати порівняння аналогів зведено в табл. 1.2.

Таблиця 1.2 – Порівняльна характеристика аналогів

Критерії	Kobold Fight Club	Donjon	bin.sh	Dungeon Scrawl
Система для створення нових персонажів	-	+	+	+
Встроєна можливість опису	-	-	+	-
Калькулятор для підрахування рівня небезпеки	+	-	-	-
автономність	-	-	-	-
Простий інтерфейс	+	+	-	-

Аналіз існуючих програмних рішень для настільних рольових ігор свідчить про потребу в комплексній системі, що дозволяє не лише створювати ігровий контент, а й зручно його редагувати та зберігати. Наприклад, Obsidian є потужним інструментом для створення пов’язаних записів і знань, дозволяючи користувачам фіксувати події кампанії, персонажів та завдання. Втім, цей інструмент не надає функцій для автоматизованого підрахунку характеристик персонажів, генерації пригод чи оцінки боїв, що обмежує його застосування саме в контексті гри за правилами D&D.

Dungeon Scrawl добре справляється з візуалізацією мап, однак цілковито ігнорує роботу з персонажами, статистикою та сюжетом, що робить його лише допоміжним інструментом. Donjon.bin.sh надає зручний набір генераторів, що дозволяє швидко отримувати шаблонні імена, підземелля або пригоди, однак його функціональність не підтримує довготривале збереження даних чи гнучке редагування. Kobold Fight Club демонструє важливість автоматизованого підбору ворогів відповідно до рівня гравців, проте його сфокусованість виключно на боях обмежує можливості в контексті загального ведення кампанії.

Новий вебзастосунок міг би поєднати сильні сторони перелічених платформ — надаючи гнучкий інтерфейс для створення та редагування персонажів, збереження сюжетної інформації, побудови та оцінки боїв, а також підрахугки рівня небезпеки. Такий підхід дозволить підтримувати як новачків, яким важливо мати інтуїтивну навігацію та підказки, так і досвідчених майстрів, що прагнуть автоматизації й кастомізації. У результаті система стане не лише засобом генерації, а й повноцінним робочим середовищем для проведення настільних ігор.

1.3 Аналіз методів створення та обробки ігрових персонажів

Розробка програмного забезпечення для розвитку логічних здібностей, зокрема у вигляді модульної системи на основі настільної гри Dungeons & Dragons, вимагає ретельного добору технологій і методів проєктування. У цьому контексті особливе значення мають логіка взаємодії, масштабованість функціоналу та

можливість поступового розширення. Система повинна не тільки дозволяти створення та зміну персонажів, а й забезпечувати їхню обробку, аналіз та зберігання з урахуванням складності та ігрового балансу. Основна увага приділяється підтримці інструментів, що сприяють розвитку логічного мислення: підрахунок ігрових параметрів, моделювання ситуацій, визначення рівня небезпеки тощо.

Клієнтська частина реалізується на базі Java з використанням JavaFX[8] — бібліотеки для створення інтерфейсів користувача. Вона дозволяє побудувати гнучку архітектуру графічних елементів, реалізувати візуальні модулі для перегляду та редагування персонажів, а також інтегрувати динамічні елементи, наприклад, редактор характеристик або форму вибору класу, раси чи здібностей. Завдяки підтримці FXML можна чітко розділити логіку та подання, що полегшує подальший розвиток і модифікацію інтерфейсу.

Для зберігання даних використовується MySQL[9] — документно-орієнтована база даних, яка дозволяє гнучко працювати з вкладеними структурами, наприклад, подіями, що мають власні властивості повторення, кількість ложок тощо. MySQL добре підходить для прототипів і реальних застосунків, які мають динамічно змінювану структуру об'єктів.

Ще одним аспектом э створення чіткого та мінімалістичного дизайну, де користувач швидко зможе опанувати застосунок та чітко виконувати поставлені задачі

1.4 Аналіз методів розрахунку рівня небезпеки персонажів

Одним із ключових функціональних модулів є розрахунок рівня небезпеки [10] (threat level) персонажів. Він базується на спеціальній формулі, яка враховує бойові параметри, кількість життів, тип зброї, магічні здібності та інші характеристики. Модуль дозволяє автоматично оцінити кожного обраного персонажа та порівняти їхню складність у межах однієї партії, що значно спрощує балансування сценаріїв. Це рішення може використовуватися як ігровими майстрами для підготовки пригод, так і гравцями для формування збалансованої

команди.

Окремий модуль відповідає за розрахунок рівня небезпеки (CR) для групи персонажів або ворогів. Формули враховують кількість істот, їхні базові характеристики (HP, атака, броня) та можливі магічні здібності. Такий підхід дає змогу створювати збалансовані ігрові ситуації автоматично, без потреби у ручному налаштуванні.

Для зберігання складних структур об'єктів, таких як атрибути персонажів, список умінь або спорядження, використовується формат MySQL. Завдяки підтримці типу у MySQL, система може гнучко обробляти динамічні або вкладені дані без необхідності створення численних зв'язків між таблицями. Це дає змогу легко додавати нові характеристики, масштабувати функціонал і спрощує серіалізацію даних під час обміну між клієнтом і сервером через REST API, що реалізоване у Spring Boot[11].

Завдяки модульній архітектурі на базі Spring Boot та інкапсуляції логіки в окремих сервісах, систему можна легко розширити. Наприклад, у майбутньому можливе підключення модулів генерації карт пригод, автоматичного створення NPC або музичного супроводу для сесії. Кожен модуль працює незалежно, що спрощує тестування та інтеграцію.

Усі вхідні дані перевіряються на відповідність допустимим значенням: атрибути не можуть перевищувати встановлений діапазон, а поля, що мають бути обов'язковими, контролюються як на клієнтському, так і на серверному рівні. Це запобігає некоректній роботі ігрових механік та підтримує цілісність бази даних.

Таким чином, у межах розробки програмного забезпечення для настільної гри було реалізовано низку технічних рішень, які дозволяють системі бути гнучкою, розширюваною та орієнтованою на користувача. Застосування модульної архітектури, зручних форматів обміну даними інтеграція з базою даних та реалізація пошукової і розрахункової логіки забезпечують повноцінну підтримку ігрового процесу. У результаті ми отримуємо програмний продукт, здатний не лише автоматизувати обчислення рівня небезпеки, а й значно спростити підготовку та проведення ігрових сесій для гравців та майстрів гри.

1.5 Висновки

У першому розділі було проведено всебічний аналіз сучасних підходів до створення програмних рішень, що сприяють розвитку логічного мислення, зокрема на основі гейміфікованих систем, таких як настільна гра Dungeons & Dragons. Розглянуто як приклади наявних платформ, так і технології, що можуть бути використані для створення власного застосунку. Особливу увагу приділено модульній архітектурі, яка дозволяє розширювати функціонал системи без втрати цілісності, а також обґрунтовано необхідність адаптивного інтерфейсу для зручності користувачів з різними рівнями підготовки.

Було окреслено базові технічні підходи до реалізації таких систем: використання Java як основної мови розробки, застосування JavaFX або Swing для клієнтської частини, інтеграція з базою даних (MySQL) для зберігання інформації про персонажів та сесії. Проаналізовано можливість створення пошукового модуля, інтерфейсу для заповнення характеристик персонажів, а також спеціального модуля для обчислення рівня небезпеки, який базується на співвідношенні параметрів ворогів і гравців.

У процесі вивчення методів розробки сформовано перелік задач, які мають бути реалізовані в межах проєкту. Кожна з них спрямована на створення ефективного інструменту для гравців і майстрів гри, що дозволить автоматизувати рутинні процеси, поліпшити підготовку до сесій та сприятиме розвитку логічного мислення через ігрову діяльність.

Таким чином, доведено актуальність і доцільність створення власного програмного забезпечення для підтримки настільних рольових ігор, що поєднує в собі функціональність, гнучкість та освітній потенціал.

2 РОЗРОБКА МОДЕЛЕЙ, МЕТОДІВ ТА ІНТЕРФЕЙСУ ПРОГРАМНОГО МОДУЛЯ

2.1 Розробка методу заповнення атрибутів

Процес створення персонажа є базовим етапом у будь-якій настільній рольовій грі, зокрема у Dungeons & Dragons. Він включає визначення ряду ключових атрибутів, таких як сила, спритність, тіловитривалість, інтелект, мудрість і харизма. Ці параметри впливають на бойову систему, взаємодію з ігровим середовищем та рольову поведінку.

Автоматизація цього процесу спрощує підготовку до гри, зменшує кількість помилок і дозволяє генерувати збалансовані персонажі як вручну, так і випадковим чином. У рамках нашого програмного модуля пропонується метод, що реалізує розрахунок початкових атрибутів за допомогою формул, заснованих на початкових характеристиках раси, класу і рівня гравця.

Метод передбачає поетапне заповнення атрибутів, виходячи з таких компонентів:

- Базове значення атрибутів (від 8 до 15), задане користувачем або згенероване випадково;
- Расові модифікатори (наприклад, +2 до сили у гномів);
- Класові бонуси (наприклад, у варвара +1 до тіла на 1-му рівні);
- Рівень персонажа (кожні 4 рівні +1 до двох різних характеристик).

Формалізація розрахунку

Нехай:

A_i – базове значення і-го атрибута;

R_i – расовий бонус до і-го атрибута;

C_i бонус від класу до і-го атрибута;

$\Delta_i(L)$ – рівень персонажа;

$\Delta_i(L)$ – приріст і-го атрибута при досягненні певного рівня;

Тоді значення і-го атрибута визначається за формулою:

$$A_i^{final} = A_i + R_i + C_i + \Delta_i(L) \quad (1.1)$$

Приклад:

Раса: гном (Сила = +2);

Клас: варвар (Сила = + 1);

Рівень: 5 (на 4-му рівні +1 до сили та +1 до тіла);

Базові значення:

Сила = 14;

Тіловитривалість = 12;

Розрахунок:

$$A_{\text{Сила}}^{final} = 14 + 2 + 1 + 1 = 18 \quad (1.3)$$

$$A_{\text{Тіло}}^{final} = 14 + 0 + 0 + 1 = 13 \quad (1.4)$$

2.2 Розробка методу випадкової зміни

Метод повної генерації атрибутів (randomizeAttributes()) є підходом до моделювання, який реалізує принцип повної випадковості згідно з правилами настільної гри Dungeons & Dragons. Цей метод не передбачає поступових змін характеристик, а повністю скидає попередні значення і створює нову конфігурацію з нуля. Він ґрунтується на ідеї стохастичного старту, де всі основні атрибути персонажа (Сила, Спритність, Статура, Інтелект, Мудрість, Харизма) отримують нові незалежні значення з заданого діапазону. На рисунку 2.1 подано фрагмент коду, де є реалізація методу.

У Dungeons & Dragons базові значення кожного атрибута можуть коливатися від 1 до 20. Метод використовує генератор випадкових чисел, який рівномірно обирає значення в межах цього діапазону. Таким чином, створюється абсолютно новий персонаж з унікальними характеристиками, що не базуються на жодних попередніх даних.

```
public void randomizeAttributes() {
    Random random = new Random();

    this.strength = random.nextInt(20) + 1;
    this.dexterity = random.nextInt(20) + 1;
    this.constitution = random.nextInt(20) + 1;
    this.intelligence = random.nextInt(20) + 1;
    this.wisdom = random.nextInt(20) + 1;
    this.charisma = random.nextInt(20) + 1;
}
```

Рисунок 2.1 – Метод випадкової змінної

Спочатку визначаються назви ключових атрибутів, що формують основу для будь-якого ігрового персонажа:

Сила (Strength), Спритність (Dexterity), Статура (Constitution), Інтелект (Intelligence), Мудрість (Wisdom), Харизма (Charisma). Це і є конфігурація

Далі на ітерації t = 1 кожен a_i (атрибут) набуває значення, яке обирається за формулою: a_i = random(1, 20);

Сила = random(1, 20);

Інтелект = random(1, 20).

У новій конфігурації всі значення є незалежними та не пов’язані з жодними попередніми станами. Це забезпечує максимально "свіжий старт" для персонажа.

Приклад реалізації: Припустимо, персонаж створюється вперше. Метод rand omizeAttributes() генерує наступні значення: сила 17, спритність 9, статура 14, інтелект 12, мудрість 5, Харизма 10.

Отриманий набір параметрів повністю унікальний і може використовуватись для початку гри, тестування сценаріїв або генерації випадкових неігрових персонажів (NPC). Такий підхід дозволяє легко симулювати сотні варіацій без втручання гравця чи додаткових правил.

Переваги методу випадкової зміни:

- Простота реалізації – не потребує обчислення похідних чи складної аналітики;

- Гнучкість – підходить для ситуацій, де важко сформулювати точну модель

залежностей;

- Пошук несподіваних конфігурацій – здатен знайти нестандартні варіанти персонажів, які не були б обрані в рамках суворих правил.

Обмеження методу

- Низька стабільність результатів – при відсутності додаткових критеріїв або механізмів відбору можлива генерація випадкових або непрактичних конфігурацій;

- Повільна збіжність – особливо у великих просторах характеристик, де випадкові зміни рідко приводять до покращень;

- Відсутність гарантії оптимальності – метод не гарантує досягнення глобального оптимуму характеристик;

- Метод випадкової зміни добре підходить для експериментів із варіативністю персонажів, генерації нових комбінацій або гнучкого налаштування початкових даних у системах на базі ігрової логіки.

2.3 Розробка методу випадкової зміни

У межах створення програмної системи для гри за типом Dungeons & Dragons було реалізовано метод підрахування рівня небезпеки (Encounter Difficulty Rating, скорочено — EDR). Його мета — визначити ступінь складності бойового зіткнення з урахуванням потужності ворогів і можливостей гравців.

Для цього використано формалізовану математичну модель, яка враховує ключові характеристики бою: кількість ворогів, їх рівень складності, кількість гравців і середній рівень кожного з них. Метод підрахування рівня небезпеки виглядає так:

$$\boldsymbol{EDR} = \frac{\sum_{i=1}^{M} \boldsymbol{CR_i \cdot W_i}}{\boldsymbol{P \cdot L_{avg}}} \qquad (1.4),$$

де C$\boldsymbol{R_i}$ це рівень складності і-го ворога;

$\boldsymbol{W_i}$ це ваговий коефіцієнт ворога залежно від його типу (сильний = 1.5, середній = 1.0, слабкий = 0.5);

M – кількість ворогів;

P – кількість гравців;

$\boldsymbol{L_{avg}}$ – середній рівень гравців.

Таким чином, у чисельнику обчислюється сумарна вага складності ворогів, а в знаменнику — загальна сила гравців. Це дозволяє оцінити співвідношення між загрозою та потенціалом протидії.

Додатково було вдосконалено та додано унікальну змінну X – коефіцієнт випадку. В залежності від ситуації, будь то засада або атака з укриття, до ворогів буде враховуватися додатковий коефіцієнт що збільшить їх рівень. Тому додатково потрібно враховувати коефіцієнт випадку. В залежності від умов коефіцієнт буде брати перемінну відповідно до списку.

Для прикладу створимо 3 ворогів зі складністю 2,1,0.5 і відповідно вагами $\boldsymbol{W_1 = 1.5, W_2 = 1.0, W_3 = 0.5}$. В свою чергу гратимуть 4 персонажі середнього рівня 3, то:

$$\boldsymbol{EDR = \frac{2 \cdot 1.5 \cdot 1 \cdot 1.0 + 0.5 \cdot 1.1 \cdot 0.5}{4 \cdot 3} = \frac{3 + 1 + 0.25}{12} = 0.354} \qquad (\boldsymbol{1.3})$$

Такий результат, означає що бій буде досить легким для гравців.

2.4 Розробка інтерфейсу вебдодатку

Вікно головної створінки матиме інструкцію та 4 головних частини. Призначення цього вікна – надати користувачеві всю необхідну інструкцію та пояснення користування застосунком. Також можливість для зручного переміщення між задачами створення, обробки та підрахунків персонажів.

Структура інтерфейсу вікна чату наведена на рисунку 2.2.

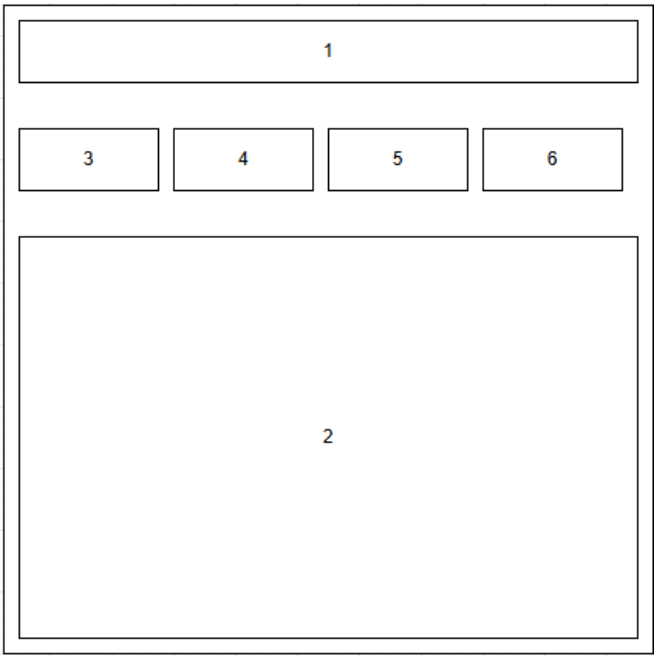

Рисунок 2.2 — Структурна схема інтерфейсу головного вікна

Складові елементи інтерфейсу головного вікна:

1. Назва сторінки;
2. Область опису та інструкції застосунку;
3. Кнопка для переміщення на сторінку «головна»;
4. Кнопка для переміщення на сторінку «створення персонажу»;
5. Кнопка для переміщення на сторінку «створенні персонажі»;
6. Кнопка для переміщення на сторінку «калькулятор».

Основні кнопки зберігаються на всіх вікнах за для ефективного пересування між частинами застосунку.

Вікно «створення персонажу» з областю заповнення атрибутів. Це вікно дозволяє створити персонажа шляхом заповнення необхідних полів назви атрибутів та опису. Структура інтерфейсу трекера настрою наведена на рисунку 2.3.

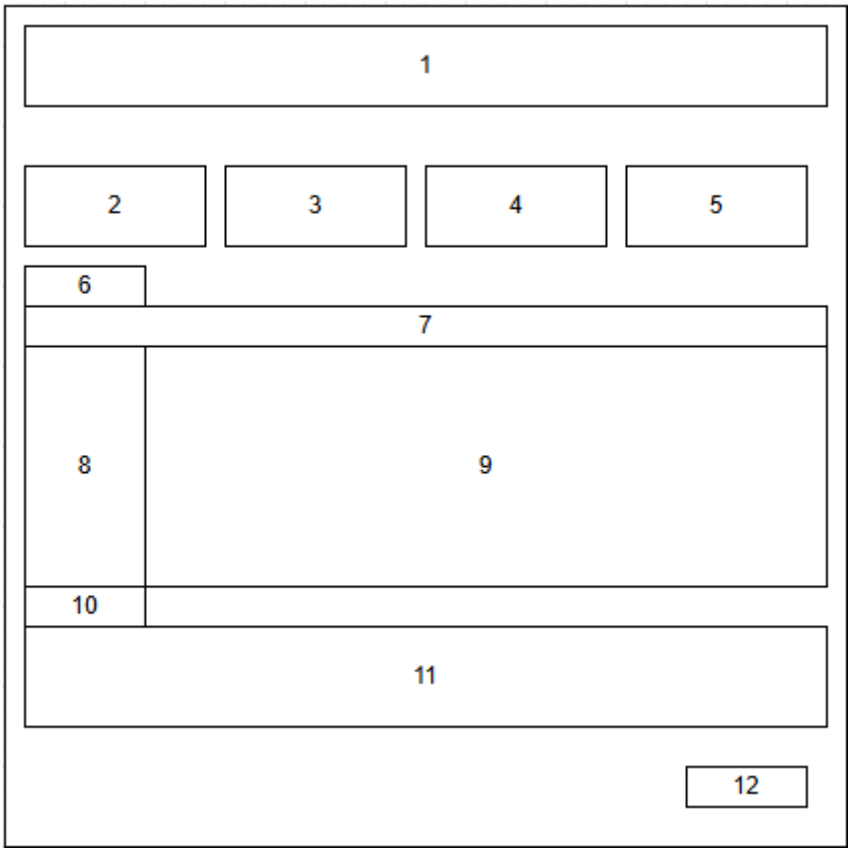

Рисунок 2.3 — Структурна схема інтерфейсу створення персонажу

Складові елементи інтерфейсу трекера настрою:

1. Назва сторінки;
2. Кнопка для переміщення на сторінку «головна»;
3. Кнопка для переміщення на сторінку «створення персонажу»;
4. Кнопка для переміщення на сторінку «створенні персонажі»;
5. Кнопка для переміщення на сторінку «калькулятор»;
6. Поле напису «ім’я персонажа»;
7. Поле для заповнення імені персонажа;
8. Таблиця назв кожного атрибуту для заповнення;
9. Поля для заповнення відповідних атрибутів;
10. Поле напису «опис»;
11. Поле для заповнення опису;
12. Кнопка для створення персонажа.

Вікно створення персонажа має всі необхідні підказки для швидкого заповнення назви атрибутів та опису персонажа, такоє кнопка створення реалізовує

перенесення інформації про створенного персонажа в вікно «створенні персонажі»

Вікно створених персонажів має список усіх створених персонажів для перегляду, редагування та видалення персонажів, які були створенні попередньо.

Вікно щоденника думок наведено на рисунку 2.4.

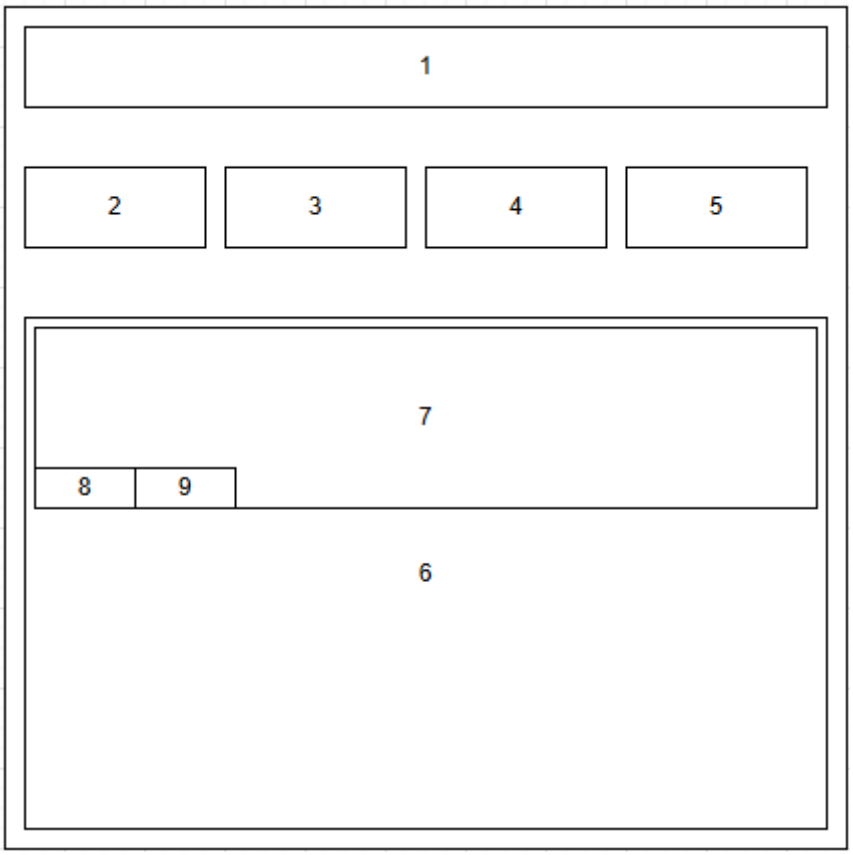

Рисунок 2.4 — Структурна схема інтерфейсу вікна журналу думок

Складові елементи вікна журналу думок:

1. Назва сторінки;
2. Кнопка для переміщення на сторінку «головна»;
3. Кнопка для переміщення на сторінку «створення персонажу»;
4. Кнопка для переміщення на сторінку «створенні персонажі»;
5. Кнопка для переміщення на сторінку «калькулятор»;
6. Область збережених персонажів;
7. Область списку певного збереженого персонажа зі списками збереженої інформації;
8. Кнопра редагування персонажа;
9. Кнопка видалення персонажа.

В області збережених персонажів будуть з’являтися списки нових збережених персонажів. Кожного з них можна окремо редагувати та видаляти.

На рисунку 2.5 наведено інтерфейс калькулятора, де буде проводитися основний процес обчислення рівня небезпеки ворогів відносно створеної групи гравців.

Мета цього компонента – провести обчислення рівня складності, враховуючи параметри створених персонажів, кількості їх та середнього рівня групи гравців.

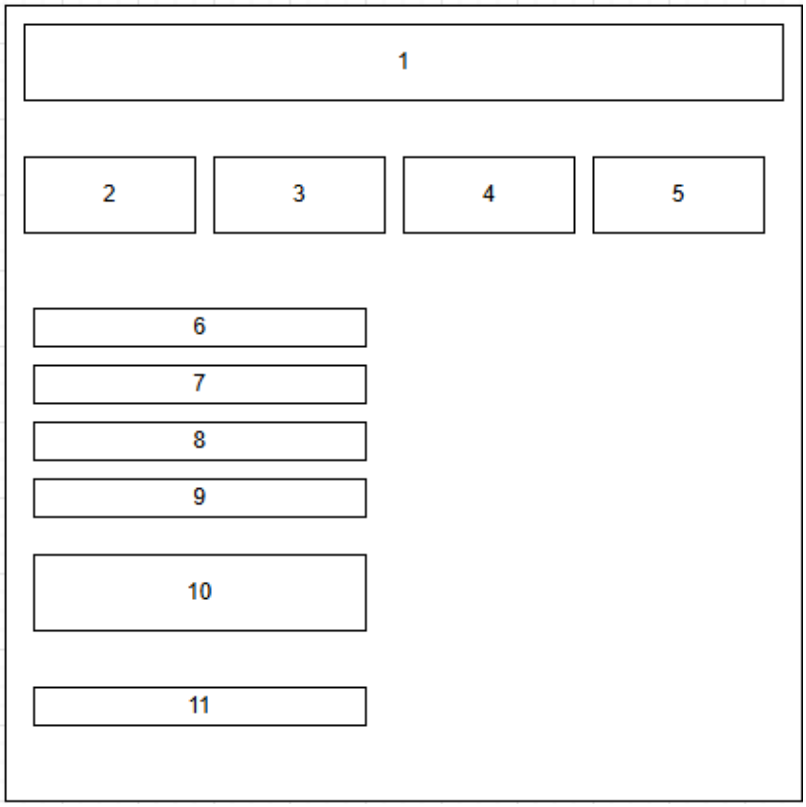

Рисунок 2.5 — Структура інтерфейсу запису вправи

Складові елементи інтерфейсу записи вправи:

1. Назва сторінки;
2. Кнопка для переміщення на сторінку «головна»;
3. Кнопка для переміщення на сторінку «створення персонажу»;
4. Кнопка для переміщення на сторінку «створенні персонажі»;
5. Кнопка для переміщення на сторінку «калькулятор»;
6. Поле для обирання ворога;
7. Поле для написання кількості гравців;

8. Поле для написання середнього рівеня героя;
9. Поле для напису кількості вогорів;
10. Кнопка розрахунку складності;
11. Поле рівня складності розрахованої після натискання кнопки. Таким чином, розроблений інтерфейс користувача дозволяє швидко проводити усі необхідні процеси для досягнення поставлених задач.

Кожен модуль побудований за чіткою методикою збереження концентрації та спрощення інтерфейсу, за для прискорення створення ігрових сесій. Користувач швидко переміщається по всім процесам: створення персонажів шляхом заповнення параметрів, перевірка та редагування списків, виконання обчислень. Таким чином інтерфейс був реалізований саме для автоматизації та прискорення процесів підготовки партії.

2.5 Розробка моделі системи

Для кращого розуміння принципу роботи програмних модулів вебсистеми було вирішено створити модель системи з використанням UML-діаграми діяльності (рис. 2.6)[15].

Ця діаграма ілюструє основні етапи взаємодії користувача з системою. Початковим кроком є вибір дії в інтерфейсі програми. Наприклад, якщо користувач має намір створити нового персонажа, він повинен ввести всі необхідні атрибути. У разі редагування вже створеного персонажа, зміни необхідно зберегти. Якщо користувач обирає функцію випадкової генерації характеристик, після цього він також має прийняти або скасувати зміни. Додатково передбачена можливість перегляду списку всіх збережених персонажів.

Після виконання будь-якої з дій користувач може перейти до іншої функції. Діаграма демонструє логіку взаємодії користувача із системою в наочній формі — у вигляді схеми або графічного інтерфейсу. Вона побудована таким чином, щоб полегшити розуміння послідовності кроків, необхідних для досягнення певної мети в системі. Серед них — введення інформації, вибір опцій, натискання елементів управління та перегляд реакцій системи на ці дії.

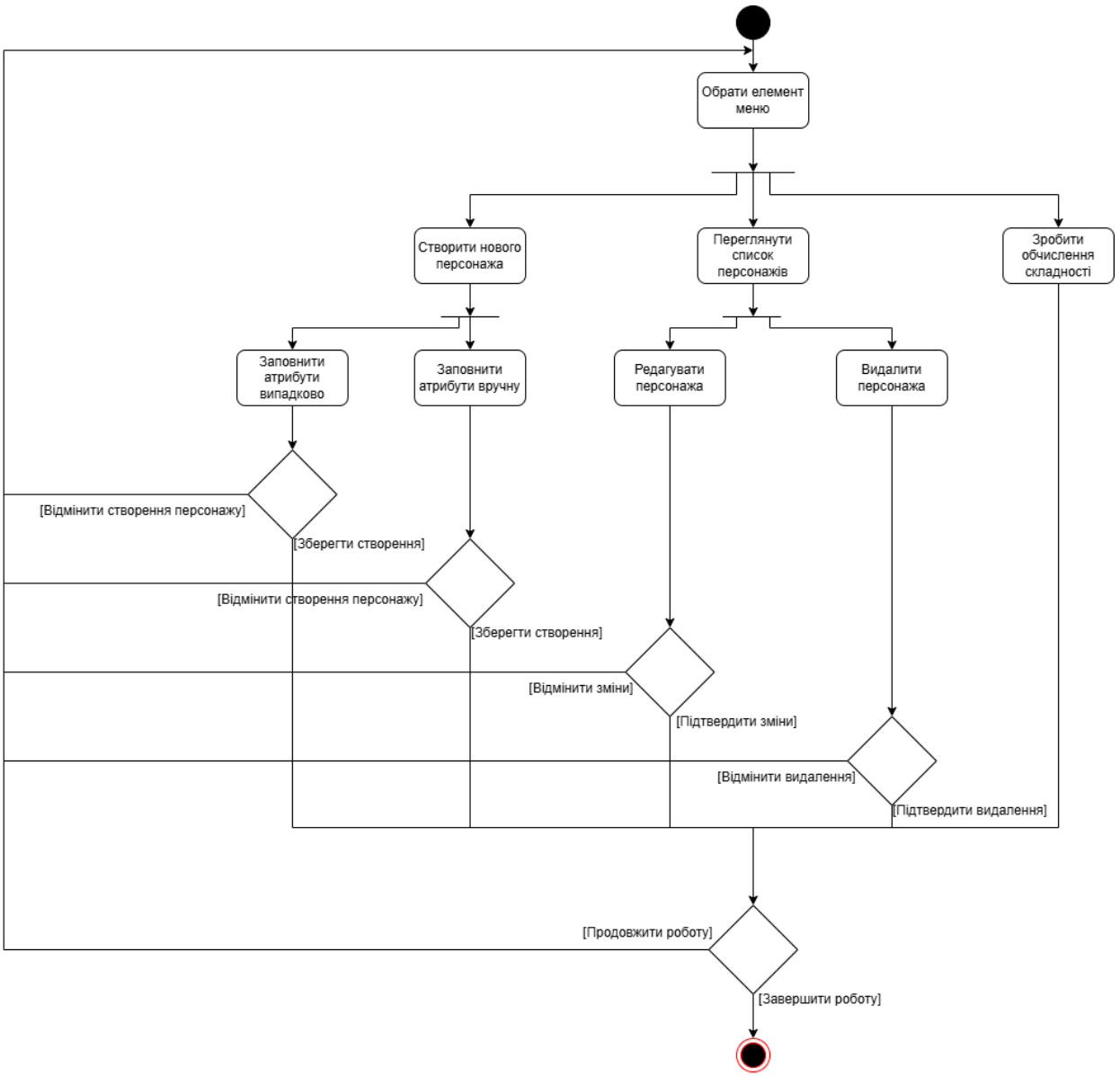

Рисунок 2.6 – Діаграма діяльності системи

Розроблений веб-застосунок структуровано на три основних модулі, які в сукупності формують єдину систему підтримки логічного мислення на основі механік настільної гри Dungeons & Dragons. Кожен з модулів виконує окрему функціональну роль, водночас тісно взаємодіючи з іншими частинами системи.

Перший модуль — модуль створення персонажів. Цей модуль відповідає за генерацію нових ігрових персонажів та реалізує два основні сценарії: ручне заповнення параметрів користувачем або автоматичне — за допомогою генератора випадкових значень. В обох випадках застосунок формує об'єкти персонажів з відповідною структурою (ім’я, клас, рівень, характеристики, опис), дана база даних

виконується у форматі SQL. Модуль використовує вбудовані форми введення, перевірку даних та можливість попереднього перегляду створеного персонажа перед збереженням. У структурі коду цей модуль розміщується у вигляді окремого пакета з контролерами введення, логікою генерації та моделями даних.

Другий модуль — модуль збереження та управління персонажами. Цей модуль формує інтерфейс для перегляду, редагування та видалення вже створених персонажів. Він взаємодіє з базою даних, витягує списки збережених файлів і відображає їх у вигляді таблиці. При виборі окремого персонажа користувач отримує доступ до детального перегляду, редагування характеристик або повного видалення. На рівні реалізації передбачено використання репозиторіїв даних, сервісів для обробки запитів, а також механізмів валідації для коректної роботи операцій. Модуль гарантує цілісність та актуальність інформації про персонажів.

Третій модуль — модуль калькулятора складності. Цей модуль реалізує функцію обчислення рівня складності сценаріїв або боїв, базуючись на параметрах персонажів, що були створені або завантажені раніше. Розрахунок виконується відповідно до формули, яка враховує кількість гравців, їхній середній рівень, рівень ворогів, особливості сутечки. Отримані значення дозволяють оцінити ймовірність успішного проходження або потребу в корекції сценарію. З технічної точки зору, модуль використовує окремий сервіс, що інкапсулює всі обчислення та надає API для отримання результатів у зручному для відображення форматі.

2.6 Висновки

У другому розділі було реалізовано ключові методи та архітектурні елементи програмної системи, що сприяє розвитку логічного мислення на прикладі настільної гри Dungeons & Dragons. Основну увагу приділено автоматизації процесів обробки персонажів, визначення рівня складності ігрової сесії та організації зберігання даних у зручному вигляді.

Метод заповнення атрибутів персонажа дає змогу автоматично сформувати основні характеристики — такі як сила, спритність, інтелект тощо — відповідно до обраного класу, раси чи сценарного контексту. Це спрощує підготовку до гри та

дозволяє уникнути помилок під час ручного введення. У разі потреби значення можуть бути адаптовані під особливі умови конкретного персонажа.

Метод випадкової зміни процесу обробки персонажа являє собою вдосконаленний метод заповнення який реалізує механізм, за якого певні характеристики або етапи формування можуть змінюватися згідно із заздалегідь визначеними ймовірностями. Це створює додаткову варіативність та непередбачуваність, що позитивно впливає на розвиток логічного мислення користувача в умовах гри.

Метод підрахування рівня небезпеки (Challenge Rating) дозволяє автоматично визначати складність створеного персонажа або супротивника. Він враховує основні параметри, такі як кількість життів, атака, захист, магічні здібності тощо, також було вдосконалено метод задопомогою додаткової змінної коефіцієнту випадку. Це дозволяє збалансувати гру та зробити сценарій відповідним до рівня гравців.

Структура бази даних реалізована таким чином, щоб зберігати інформацію про створених персонажів, їхні характеристики, результати розрахунків рівня небезпеки та історію змін. Дані організовано у взаємопов’язані таблиці, що дозволяє здійснювати ефективні запити, фільтрацію та редагування. Також передбачено можливість подальшого розширення бази для зберігання нових типів даних або введення додаткових параметрів персонажів.

Таким чином, у другому розділі було реалізовано основні функціональні складові програмної системи: заповнення атрибутів, варіативну обробку даних, автоматизоване визначення рівня складності та зберігання даних. Це створює основу для подальшого розвитку інтелектуальної гри та підтримує розвиток логічного мислення у користувачів через аналіз, порівняння і прийняття рішень у змодельованих ситуація

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ДЛЯ АВТОМАТИЗАЦІЇ РОБОЧИХ ПРОЦЕСІВ ВЕБЗАСТОСУНКУ

3.1 Вибір мови програмування

Вибір мови програмування для розробки програмних засобів, спрямованих на розвиток логічного мислення через ігрову взаємодію, є вирішальним етапом, що впливає на стабільність, розширюваність і довготривалу підтримку програмного продукту. У контексті створення вебзастосунку, натхненного механікою настільної гри Dungeons & Dragons, особливо важливими є такі характеристики, як надійність, кросплатформеність, безпека та зручність масштабування. Однією з найбільш доцільних мов у цьому випадку є Java, яка поєднує гнучкість з високим рівнем продуктивності.

Java має низку переваг, які забезпечують стабільну роботу інтерактивних ігрових платформ. Завдяки віртуальній машині JVM та підтримці багатопоточності, Java дозволяє ефективно реалізовувати обробку даних у режимі реального часу — наприклад, розрахунок рівня небезпеки в ігрових сценаріях або генерацію випадкових подій у грі. Хоча Python[12] вважається зручним для прототипування та має простий синтаксис, він поступається Java за швидкістю виконання та масштабованістю, що може бути критично важливим у багатокористувацьких ігрових системах.

Порівнюючи Java з іншими мовами, такими як C++[13], слід зазначити, що остання дає більше контролю над системними ресурсами, але водночас ускладнює процес керування пам’яттю та підвищує ризик помилок. C#, хоч і схожий на Java за синтаксисом та підтримкою багатопоточності, зазвичай сильніше прив’язаний до екосистеми Microsoft, що звужує можливості для універсального розгортання. JavaScript із Node.js активно використовується у веброзробці, однак у контексті логічно-орієнтованих ігор з великою кількістю внутрішніх обчислень та обробкою складних структур (наприклад, даних про персонажів чи сценаріїв) його продуктивність може бути недостатньою.

Окремої уваги заслуговує питання надійності — важливого чинника у будь-якому застосунку, що передбачає взаємодію користувача зі складною логікою. Java забезпечує високу надійність завдяки суворій типізації та вбудованій системі обробки винятків, що дозволяє зменшити кількість помилок під час виконання та зручно відстежувати помилки в логіці ігрових механік. У Python динамічна типізація хоч і забезпечує гнучкість, але водночас ускладнює виявлення логічних помилок. У C++ відповідальність за керування пам’яттю повністю лежить на розробнику, що підвищує ймовірність непередбачених збоїв. У свою чергу, C# демонструє хороші результати в контексті надійності, але поступається Java у кросплатформеності.

З міркувань безпеки Java пропонує комплексні вбудовані засоби захисту, зокрема менеджери безпеки, захист виконуваного коду та підтримку сучасних механізмів шифрування. Це є перевагою при створенні застосунків, де користувач може зберігати персоналізовані ігрові дані, історії персонажів або результати логічних завдань. У Python існують додаткові бібліотеки для захисту, але базові можливості мови в цій сфері є менш розвиненими. У C++ розробник має повну свободу реалізації захисту, що робить цей процес трудомістким. C# пропонує сильну систему безпеки, однак тісна інтеграція з Windows обмежує його застосування на інших платформах.

Кросплатформеність є ключовою перевагою Java. Завдяки JVM розроблений застосунок може однаково функціонувати на різних операційних системах без потреби у значних змінах коду. Це є особливо важливим у випадку ігор, які мають на меті охопити широку аудиторію користувачів. Python також підтримує запуск на різних платформах, але його продуктивність у середовищах із високими вимогами до обчислень може бути нестабільною. C++ вимагає ручної компіляції під кожну операційну систему. C# у поєднанні з .NET Core набув більшої універсальності, але все ще часто обмежується Windows-платформам[14].

Ще одним аргументом на користь Java є її потужна екосистема. Існує широкий вибір фреймворків та бібліотек (зокрема Spring та Hibernate), які можна адаптувати для обробки ігрової логіки, керування сесіями, обчислення

характеристик персонажів чи генерації випадкових подій. У Python наявні корисні засоби для прототипування або аналітики, однак у контексті побудови складної, багатомодульної ігрової системи його можливості є обмеженими. C++ вимагає більше часу на налаштування та реалізацію базових речей, що сповільнює розробку. Екосистема C# достатньо багата, але переважно сфокусована на комерційних продуктах у рамках Microsoft. JavaScript зосереджений на вебфронтенді, і хоч має динамічну екосистему, для обчислювально складних ігрових моделей він менш придатний.

З урахуванням усіх вищезгаданих характеристик, Java виглядає найоптимальнішим вибором для реалізації вебзастосунку, спрямованого на розвиток логічного мислення за допомогою гейміфікованої моделі, натхненної Dungeons & Dragons. Порівняльні характеристики основних мов програмування зведено до таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Параметр	Java	Python	C++	C#	JavaScript (Node.js)
Продуктивність	Висока	Середня	Висока	Висока	Низька
Надійність	Висока	Середня	Висока, але складна	Висока	Низька
Безпека	Висока	Середня	Висока, але ручна	Висока	Середня
Кросплатформеність	Висока	Висока	Низька	Середня	Висока
Екосистема	Широка, для фінансів	Широка, для аналітики	Обмежена	Широка, для Windows	Широка, для веб

Таким чином, Java виявляється найкращим вибором для розробки програмних засобів, спрямованих на розвиток логічного мислення на основі настільної гри Dungeons & Dragons, завдяки високій продуктивності, надійності,

безпеці, кросплатформеності та розвиненій екосистемі.

3.2 Вибір середовища розробки

Середовище розробки відіграє ключову роль у створенні програмного забезпечення, оскільки саме воно визначає зручність роботи розробника, продуктивність під час написання коду, а також загальну якість кінцевого продукту. Правильно підібране IDE дозволяє не лише пришвидшити процес розробки, а й зменшити кількість помилок, завдяки вбудованим інструментам автодоповнення, рефакторингу та налагодження.

Серед найпопулярніших середовищ, які підтримують мову програмування Java, можна виокремити такі, як IntelliJ IDEA, Eclipse, NetBeans та Visual Studio Code. Кожне з них має власні переваги, функціональність і цільову аудиторію. У цьому проєкті було прийнято рішення використовувати Visual Studio Code (VS Code) як основне середовище розробки. Такий вибір обумовлений низкою переваг, що роблять цю платформу зручною, ефективною та адаптивною для розробки модульного програмного забезпечення, яке спрямоване на підтримку логічного мислення через механіку настільної гри Dungeons & Dragons.

Visual Studio Code[14] — це кросплатформене, легке та швидкодійне середовище розробки з відкритим кодом, створене компанією Microsoft. На відміну від класичних IDE, таких як IntelliJ IDEA або Eclipse, VS Code базується на концепції редактора коду з розширюваними можливостями. Завдяки потужній екосистемі розширень, це середовище здатне підтримувати повноцінну розробку на Java — зокрема, завдяки Java Extension Pack, який включає підтримку Maven, Spring Boot, автодоповнення, IntelliSense, компіляцію, запуск і налагодження коду.

У порівнянні з більш потужними, але важкими IDE, VS Code споживає набагато менше системних ресурсів, швидше завантажується і працює навіть на менш продуктивних пристроях. Це робить його практичним вибором для студентських або академічних проєктів, які не потребують великої кількості додаткових інструментів. Для розробки модульного Java-застосунку, де кожен модуль реалізує окрему функцію — наприклад, генерацію персонажів, заповнення

характеристик, побудову сценаріїв або обчислення складності бою — саме легкість і швидкість середовища є вагомими перевагами.

Інтерфейс VS Code[15] є лаконічним, інтуїтивно зрозумілим і високофункці ональним. Він дозволяє зосередитися на коді, не перевантажуючи користувача великою кількістю елементів управління. Для досвідчених розробників VS Code відкриває можливість тонкого налаштування під особисті потреби, а для новачків — забезпечує простий поріг входження у світ Java-розробки.

Крім того, Visual Studio Code підтримує інтеграцію з Git, що дозволяє реалізувати контроль версій безпосередньо зі середовища розробки. Також у ньому зручно працювати з REST-запитами, використовувати вбудований термінал, запускати локальні сервери або підключати Docker-контейнери для симуляції реального середовища розгортання. Це актуально для проєктів, які потенційно можна розширити до клієнт-серверної архітектури або інтегрувати з вебінтерфейсом.

У контексті даного проєкту, який базується на розробці логіко-орієнтованого ігрового модуля на Java, VS Code забезпечує достатню функціональність і гнучкість. Середовище дозволяє ефективно організувати структуру проєкту, реалізувати окремі логічні блоки з чітко визначеними зв’язками між ними, і швидко переходити до масштабування застосунку при розширенні функціоналу. Підтримка таких інструментів, як Spring Framework або Hibernate, через плагіни, відкриває можливість подальшої інтеграції з базами даних, серверними частинами або сторонніми сервісами.

Таблиця 3.2 – Порівняльна характеристика середовищ розробки

Характеристика	IntelliJ IDEA	Eclipse	NetBeans	Visual Studio Code
Зручність використання	Інтуїтивний інтерфейс, потужне автодоповнення	Складніший інтерфейс, повільніша робота	Простий інтерфейс, але обмежений	Зручний, але більше для веб-розробки

Продовження таблиці 3.2

Продуктивність	Висока, оптимізована для великих проектів	Середня, може підвисати	Низька на великих проектах	Середня, залежить від плагінів
Підтримка фреймворків	Глибока інтеграція з популярними фреймворками (Spring, Hibernate)	Добра, але менш зручна	Обмежена підтримка	Основна підтримка через плагіни
Рефакторинг коду	Потужні інструменти для рефакторингу	Базові інструменти	Обмежені можливості	Базові інструменти через плагіни
Характеристика	IntelliJ IDEA	Eclipse	NetBeans	Visual Studio Code
Підтримка тестування	Інтеграція з JUnit, TestNG, простота запуску тестів	Схожа, але менш зручна	Базова підтримка	Обмежена підтримка через плагіни
Інтеграція з системами контролю Версій	Потужна інтеграція з Git, SVN	Підтримка, але менш зручна	Базова підтримка	Основна підтримка через плагіни

Visual Studio Code був обраний як основне середовище розробки для проєкту «Розробка програмних засобів для сприяння розвитку логічних здібностей на прикладі настільної гри Dungeons & Dragons» завдяки своїй швидкодії, простоті у використанні, мінімальним вимогам до ресурсів та гнучкості у налаштуванні

середовища. Його розширюваність, зручна інтеграція з сучасними інструментами та підтримка Java-технологій робить його доцільним вибором для реалізації модульного застосунку з великою кількістю логічних та ігрових компонентів

3.3 Розробка модуля для створення персонажів

Одним із важливих модулів у системі є модуль для створення персонажів, який забезпечує користувача можливістю формування базових ігрових сутностей — героїв, що використовуються в рамках логічної гри. Основна функціональність модуля полягає у введенні ключових характеристик персонажа та збереженні їх до відповідної структури в базі даних.

Структура класу CharacterModel включає в себе низку основних характеристик, необхідних для повноцінного представлення персонажа у настільній грі або програмному середовищі, що моделює механіку Dungeons & Dragons. Ці характеристики мають тип цілих чисел (int) і охоплюють такі параметри, як сила (Strength), спритність (Dexterity), інтелект (Intelligence), витривалість (Endurance), харизма (Charisma) та мудрість (Wisdom).

Крім основних числових характеристик, до структури також входять клас персонажа (Class) — рядкове поле, яке визначає роль персонажа в ігровому процесі (наприклад, «воїн», «магістр», «паладин»), та опис (Description) — текстова частина, яка не бере участі у математичних обчисленнях, але дозволяє користувачеві задавати індивідуальні особливості чи передісторію героя. Опис є важливим елементом для збереження до бази даних і подальшого використання в інтерфейсі програми або візуалізації.

Усім персонажам автоматично призначається унікальний ідентифікатор (ID) у форматі Guid, що дозволяє надійно зберігати та шукати записи у базі даних.

Ручне введення параметрів користувачем — дозволяє заповнити всі характеристики вручну на основі власного бачення чи певних правил гри.

Автоматичну генерацію характеристик — передбачає використання спеціального алгоритму з обмеженням допустимих відхилень.

У разі автоматичної генерації застосовується функція мутації, яка базується

на змінюванні базових значень у межах визначеного діапазону. Для кожної характеристики задається початкове значення (наприклад, 10), після чого за допомогою генератора випадкових чисел додається або віднімається випадкова величина в межах від -3 до +3. Такий підхід дозволяє зберігати баланс параметрів, забезпечуючи при цьому варіативність створення персонажів.

Загальний принцип мутації реалізується за допомогою допоміжної функції:
int Mutate(int baseValue) => baseValue + rnd.Next(-maxDelta, maxDelta + 1);

Тут baseValue — базове значення характеристики, а maxDelta — максимально допустиме відхилення, яке може бути змінене або параметризоване в майбутніх версіях.

У режимі ручного введення користувач самостійно заповнює характеристики персонажа, використовуючи відповідні поля у графічному інтерфейсі. У цій частині реалізації кожне з шести числових значень (сила, спритність, інтелект, витривалість, харизма, мудрість) вводиться окремо через текстові поля.

Після того як користувач заповнює всі значення, програма зчитує вміст кожного поля, перетворює введені дані з текстового формату в числовий тип int, і зберігає їх у відповідні змінні.

Крім базових характеристик, користувач також заповнює два додаткових текстових поля: одне для класу персонажа, яке визначає його роль (наприклад, «воїн», «чаклун»), та інше для опису персонажа, що зберігає вільний текст із будь-якою інформацією, яку користувач бажає додати. Цей опис не впливає на розрахунки, але буде внесений до бази даних як частина повного профілю персонажа.

Після зчитування всіх полів створюється новий об’єкт персонажа — тобто викликається конструктор моделі CharacterModel, куди передаються всі зібрані значення як аргументи. У результаті формується повноцінна структура, готова до використання в інших модулях програми або збереження в базі даних.

Нижче наведено діаграму роботи алгоритму для створення персонажів у графічному представлені (рис. 3.1).

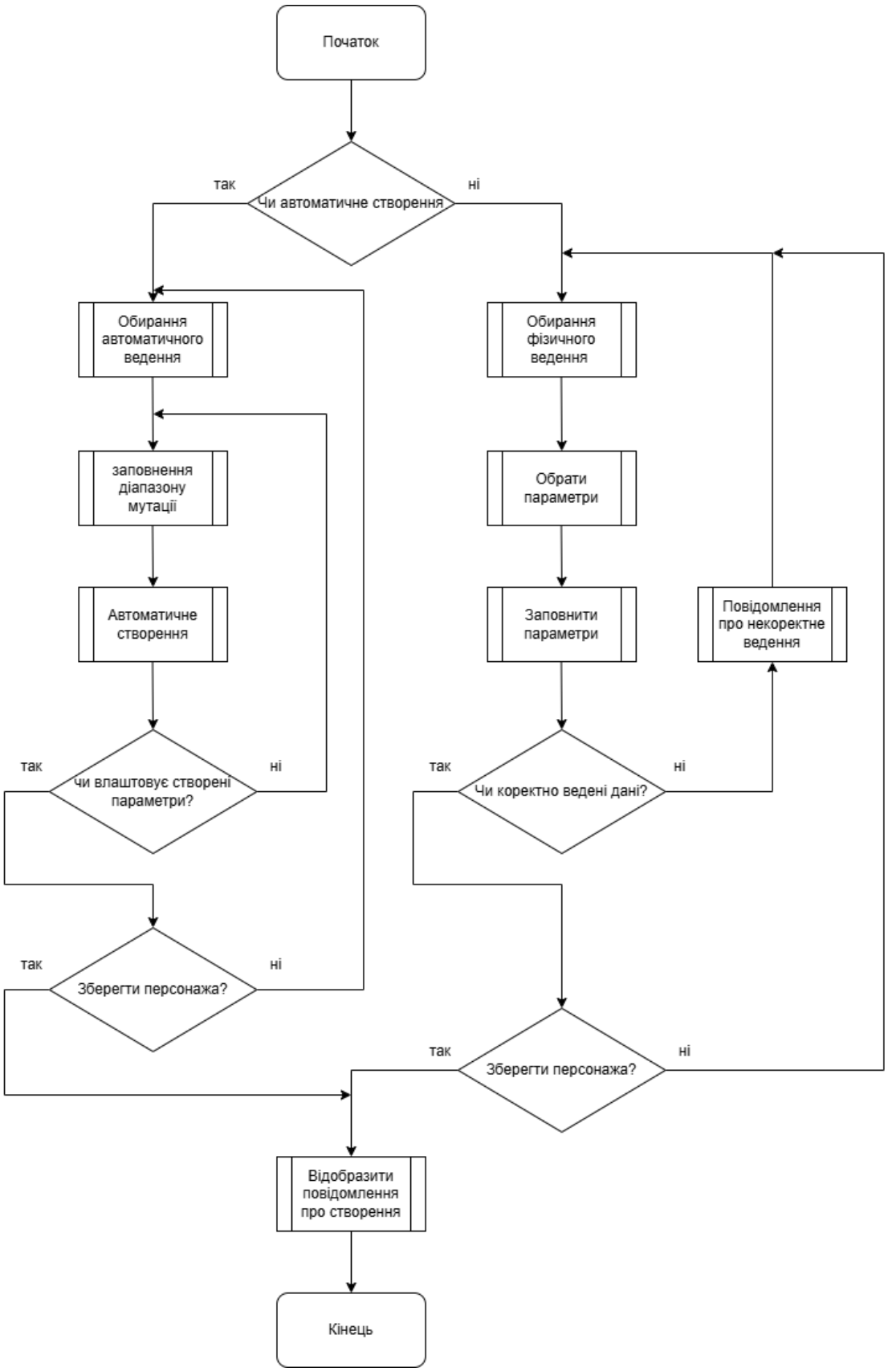

Рисунок 3.1 – Схема алгоритму для створення персонажа

Такий підхід дозволяє повністю контролювати процес створення персонажа

і дає змогу користувачу максимально персоналізувати свого героя згідно з власними вподобаннями.

3.4 Розробка модуля для збереження персонажів

Основною метою модуля є надання користувачеві можливості взаємодіяти зі створеними персонажами у форматі повноцінної бази даних. Створені персонажі мають бути збережені у базі даних (реалізація — SQL, лістинг структури навежено на рис. 3.2), а також доступні для перегляду, редагування та подальшого використання. При цьому редагування реалізується як оновлення даних у вже збережених записах у таблиці бази даних.

```java
public class Character {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private int strength;
    private int dexterity;
    private int intelligence;
    private int endurance;
    private int charisma;
    private int wisdom;

    @Column(name = "class_name")
    private String className;

    @Column(columnDefinition = "TEXT")
    private String description;

    // Getters, Setters, Constructors
}
```

Рисунок 3.2 – Лістинг контролеру для редагування

Користувач може відкрити сторінку з усіма збереженими персонажами, обрати одного з них для редагування та змінити будь-яке з доступних полів — наприклад, ім’я, клас, опис або характеристики, зокрема силу, інтелект, мудрість тощо. Це забезпечує гнучкість і динамічну зміну ігрових умов без потреби створювати нового персонажа з нуля (рис. 3.3).

Ключовим елементом реалізації такого функціоналу є веб-форма редагування, яка завантажує поточні дані з бази у поля форми, а після внесення змін зберігає оновлення через серверну логіку. Веб-застосунок реалізує цей підхід завдяки маршрутам, які обробляють як запити GET для завантаження поточних

значень, так і POST (або PUT) для надсилання оновлень.

```
public class CharacterController {

    @Autowired
    private CharacterRepository characterRepository;

    @PutMapping("/{id}")
    public ResponseEntity<Character> updateCharacter(
        @PathVariable Long id,
        @RequestBody Character updatedCharacter
    ) {
        Optional<Character> optionalCharacter = characterRepository.findById(id);
        if (!optionalCharacter.isPresent()) {
            return ResponseEntity.notFound().build();
        }

        Character existing = optionalCharacter.get();
        existing.setName(updatedCharacter.getName());
        existing.setStrength(updatedCharacter.getStrength());
        existing.setDexterity(updatedCharacter.getDexterity());
        existing.setIntelligence(updatedCharacter.getIntelligence());
        existing.setEndurance(updatedCharacter.getEndurance());
        existing.setCharisma(updatedCharacter.getCharisma());
        existing.setWisdom(updatedCharacter.getWisdom());
        existing.setClassName(updatedCharacter.getClassName());
        existing.setDescription(updatedCharacter.getDescription());

        Character saved = characterRepository.save(existing);
        return ResponseEntity.ok(saved);
    }
```

Рисунок 3.3 – REST редагування

Редагування збережених персонажів у веб-застосунку реалізовано у вигляді REST-сервісу, який дозволяє оновити вже існуючого персонажа за його унікальним ідентифікатором. Основу становить клас Character, який є відображенням таблиці бази даних SQL, де зберігаються всі атрибути персонажа: ім’я, основні характеристики, клас та опис. Всі поля цього класу редаговані, що дозволяє вільно оновлювати значення на нові, введені користувачем через фронтенд.

Контролер отримує нові дані через PUT-запит, і якщо відповідний запис існує в базі, він оновлюється за допомогою JpaRepository. Оновлення відбувається через стандартні Java методи set, які змінюють значення полів у моделі. Після цього результат зберігається у базу, і оновлений об’єкт повертається у відповідь.

Така структура дозволяє інтегруватися з будь-яким сучасним фронтендом (React, Vue, Angular) через REST API і дає змогу легко редагувати дані в базі, дотримуючись принципів MVC-архітектури.

3.5 Розробка модуля для калькулятора складності

У межах програмної системи було реалізовано окремий модуль, відповідальний за обчислення рівня складності бойових зіткнень (Encounter Difficulty Rating, скорочено — EDR), що дозволяє оцінити співвідношення між бойовою силою противників і потенціалом гравців. Цей функціонал є важливим для формування збалансованих ігрових ситуацій та побудований на використанні формалізованої математичної моделі[16].

Суть роботи модуля полягає у збиранні необхідних параметрів про ворогів (рівень складності, тип, кількість) та гравців (рівень, кількість), обчисленні індивідуальних вагових коефіцієнтів для кожного ворога та, відповідно, підрахунку загального EDR за формулою.

Обчислення EDR здійснюється на основі даних, отриманих із бази даних, де зберігається інформація про створених персонажів. Відповідно, модуль взаємодіє з системою збережених даних, витягуючи з неї лише ті записи, які мають стосунок до поточного зіткнення. Детальний приклад такого витягування даних представлено на рис. 3.4. У ньому реалізовано механізм формування SQL-запиту з урахуванням переданого списку ідентифікаторів персонажів. Після встановлення з’єднання з MySQL-сервером виконується запит до таблиці з персонажами, а результати зберігаються у вигляді списку словників. Такий підхід забезпечує як гнучкість у роботі з даними, так і захист від можливих SQL-ін’єкцій.

```
characters_data = []
try:
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor(dictionary=True)
    if character_ids:
        placeholders = ', '.join(['%s'] * len(character_ids))
        query = f"SELECT * FROM characters WHERE id IN ({placeholders})"
        cursor.execute(query, tuple(character_ids))
        characters_data = cursor.fetchall()
except mysql.connector.Error as err:
    print(f"Database error: {err}")
finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
return characters_data
```

Рисунок 3.4 – функція отримання інформації з бази даних

Таким чином, модуль калькулятора складності не лише реалізує саму формулу обчислення, а й інтегрується з базою даних, дозволяючи обробляти бойові ситуації з урахуванням реальних характеристик учасників, що вже збережені в системі.

3.6 Висновки

На основі проведеного аналізу у третьому розділі було здійснено обґрунтований вибір середовища розробки для реалізації програмного засобу, спрямованого на розвиток логічного мислення через механіки настільної гри Dungeons & Dragons. Серед доступних рішень особливу увагу було приділено Visual Studio Code, IntelliJ IDEA, Eclipse та NetBeans.

було створено UML-діаграму діяльності, яка наочно ілюструє послідовність дій користувача у вебсистемі: вибір функцій, введення даних, генерація персонажів, редагування та перегляд списку збережених об’єктів. Це допомогло чітко окреслити логіку взаємодії користувача з системою.

Розроблено модуль для створення персонажів із можливістю ручного введення характеристик або автоматичної генерації з варіаціями. Визначена структура даних персонажа з атрибутами і унікальним ідентифікатором. Реалізовано алгоритм мутації для генерації варіативних характеристик у допустимих межах. Забезпечено зручний інтерфейс для введення параметрів.

Створено модуль, який надає можливість переглядати, редагувати і видаляти збережених персонажів із бази даних. Реалізовано веб-інтерфейс для редагування, який працює через REST-сервіс із підтримкою операцій GET та PUT. Забезпечено цілісність даних та інтеграцію з фронтенд-фреймворками.

Розроблено окремий сервіс для обчислення рівня складності бойових сценаріїв на основі параметрів персонажів і ворогів. Модуль використовує визначені формули для оцінки ймовірності успішного проходження та допомагає коригувати сценарії. Реалізація забезпечує простий API для інтеграції з іншими частинами системи.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення є критично важливим, а подекуди й вирішальним етапом у процесі розробки будь-якого сучасного застосунку, незалежно від його масштабу, складності чи призначення. Саме завдяки цьому етапу стає можливою системна перевірка правильності реалізації функціональності, виявлення різноманітних помилок, логічних суперечностей або технічних недоліків, а також підтвердження того, що створений програмний продукт дійсно відповідає всім визначеним вимогам, що були сформульовані під час етапу проєктування та специфікації. Іншими словами, тестування дозволяє не лише виявити слабкі місця у логіці роботи застосунку, а й гарантує, що його функціональні можливості реалізовані відповідно до технічного завдання та очікувань кінцевого користувача[17].

Під поняттям «тестування» в контексті розробки програмного забезпечення мається на увазі не просто перевірка роботи окремих елементів інтерфейсу або дій користувача, а комплексний процес оцінки якості функціонування програмної системи. Головною метою цього процесу є виявлення будь-яких невідповідностей між фактичною поведінкою системи під час виконання і тим результатом, який був запланований або передбачений технічною документацією. Проведення такого аналізу дозволяє покращити надійність програмного продукту, зменшити ймовірність помилок у майбутньому, оптимізувати роботу окремих функціональних блоків та забезпечити узгоджену і стабільну взаємодію між усіма модулями та підсистемами, з яких складається повноцінна програмна архітектура.

Початковий етап тестування розпочинається з підготовчих дій, серед яких першочергове місце займає встановлення тестової версії програми на обрану апаратну платформу. У нашому випадку, тестування проводиться на персональному комп’ютері, що функціонує під управлінням операційної системи Windows, яка виступає як основна цільова платформа для роботи розробленого застосунку. Паралельно з цим необхідно переконатися в наявності всього

комплекту документації, яка потрібна для ефективного тестування. До такого комплекту обов’язково входять технічні специфікації проєкту, перелік і опис вимог до функціоналу, інструкції до інтерфейсу, а також детальний план тестування, в якому описуються всі передбачені сценарії, очікувані результати та порядок їх перевірки.

Тестування зазвичай складається з кількох взаємопов’язаних етапів і охоплює різні типи перевірок, кожна з яких має свою мету та методологію. Першим з таких етапів є функціональне тестування, яке є основою всього процесу перевірки. Метою функціонального тестування є визначення того, наскільки правильно реалізовано основні функції системи, які є критичними для її повсякденного використання. Сюди входить перевірка операцій введення даних користувачем, обробки цих даних системою, а також виведення відповідної інформації у вигляді результатів або повідомлень. Особлива увага приділяється тому, як система реагує на введення некоректних або помилкових значень — наприклад, букв у числові поля, пусті рядки там, де обов’язково має бути заповнено значення, або вихідні значення за межами допустимого діапазону. Усі ці дії співвідносяться із заздалегідь визначеними сценаріями, і позитивний результат свідчить про те, що програма поводиться так, як це передбачено технічним завданням[18].

Другим етапом тестування є оцінка відмовостійкості програмного продукту. Цей тип тестування зосереджується на перевірці здатності системи стабільно функціонувати в умовах виникнення збоїв, помилок або непередбачуваних ситуацій, які можуть виникнути як з вини користувача, так і з боку зовнішніх чинників. У процесі відмовостійкого тестування моделюються ситуації введення неправдивих або некоректних даних, навмисного порушення логіки дій, втрати підключення до мережі, збою у взаємодії з зовнішніми модулями або базами даних. Метою цього тестування є перевірка того, наскільки ефективно система може не лише виявити такі ситуації, а й правильно на них відреагувати — наприклад, шляхом виведення попереджувального повідомлення, блокування помилкових дій або автоматичного відновлення працездатності після збоїв.

Третім важливим етапом тестування є перевірка продуктивності програмного забезпечення. Цей вид тестування дозволяє визначити швидкість виконання основних операцій у системі, час відгуку інтерфейсу на дії користувача, а також загальну ефективність роботи програми в умовах інтенсивного навантаження. Зокрема, перевіряється, наскільки швидко система обробляє запити, зберігає інформацію, перемикається між вікнами або режимами роботи, а також наскільки стабільно працює під час багаторазового або паралельного використання. Результати цього тестування дозволяють розробникам переконатися, що застосунок здатен працювати в умовах реального середовища експлуатації, де навантаження можуть бути суттєвими.

Крім того, окремо проводиться тестування сумісності, яке передбачає перевірку коректності роботи програмного продукту на різних конфігураціях апаратного забезпечення, версіях операційної системи та, за потреби, у взаємодії з іншими програмами. Такий вид тестування дозволяє визначити, чи не виникають конфлікти між застосунком та сторонніми бібліотеками, драйверами, системними утилітами тощо. В результаті цього тесту можна переконатися в тому, що програмний продукт дійсно адаптований до широкого спектру умов його використання, що підвищує його універсальність і знижує ймовірність технічних збоїв після встановлення користувачем.

На рисунку 4.1 продемонстровано приклад інтерфейсу програми, в якому виводиться повідомлення про помилку під час спроби входу до системи із використанням недійсних або некоректних облікових даних. Така функціональність є не лише стандартною вимогою до будь-якого сучасного застосунку, але й важливим елементом безпеки. Вона дозволяє запобігти несанкціонованому доступу до персональних даних користувача, сприяє покращенню зручності використання і забезпечує загальну надійність програми в частині автентифікації.

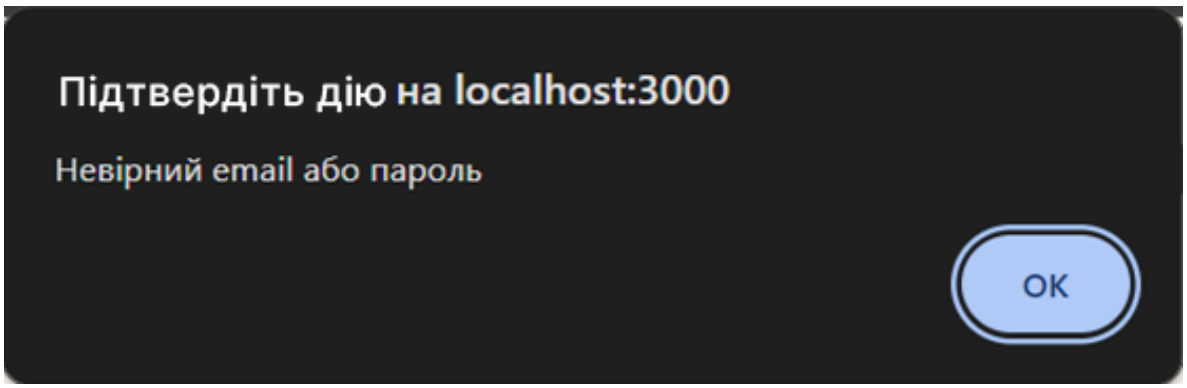

Рисунок 4.1 – Невдалий вхід у застосунок

У межах другого етапу тестування було досліджено поведінку модуля системи калькулятора складності у ситуаціях, коли користувач свідомо або випадково намагається ввести значення, що виходять за межі допустимого діапазону. Зокрема, перевірялася реакція системи на спробу зменшити кількість ворогів або їх рівень нижче встановленого мінімального порогу, що критично впливає на коректність розрахунків і загальну стабільність роботи застосунку.

Цей сценарій має принципове значення, оскільки правильне функціонування калькулятора складності забезпечує дотримання ігрового балансу, особливо під час підготовки до бою з участю гравців різного рівня. У рамках перевірки було змодельовано ситуацію, коли користувач вводить значення, менші за мінімально допустимі — наприклад, спроба встановити кількість ворогів на рівні -1. Така дія автоматично блокується на рівні інтерфейсу: елемент управління не дозволяє фізично ввести від’ємне число, а у разі обхідної спроби (наприклад, вставлення значення вручну) система зчитує це як некоректний ввід та не проводить розрахунок.

На рисунку 4.2 представлено інтерфейс, у якому наочно демонструється обмеження: зниження рівня або кількості ворогів нижче допустимого порогу неможливе, а сама система виводить відповідне повідомлення про те, що розрахунок за таких умов не може бути виконаний. Така реалізація гарантує стабільну логіку роботи калькулятора та унеможливлює введення абсурдних параметрів, які могли б порушити внутрішню гармонію ігрового процесу.

Таким чином, перевірка цього модуля підтвердила відповідність застосунку вимогам до надійного оброблення граничних значень та запобігання помилкам, що можуть виникати під час інтерактивної роботи з числовими даними.

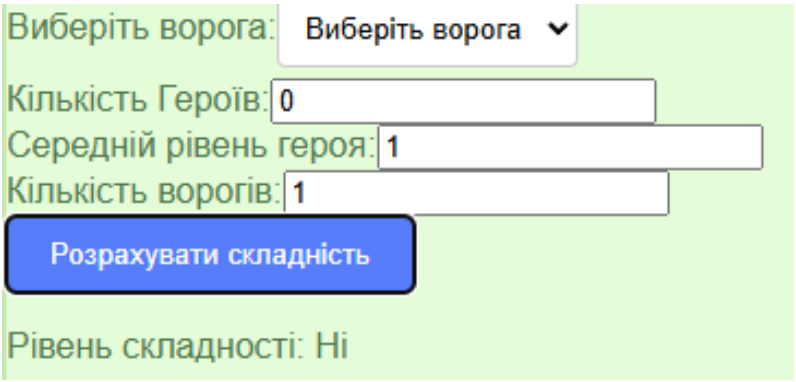

Рисунок 4.2 – повідомлення про неможливість розрахунку складності

У рамках виконання третього тестового завдання особливу увагу було приділено перевірці реакції системи на введення користувачем некоректних або неприйнятних даних під час створення персонажів. Такий підхід дозволив оцінити ефективність реалізованих обмежень та валідації, які покликані запобігти формуванню нереалістичних чи технічно некоректних характеристик у грі.

Зокрема, під час перевірки була свідомо введена неправильна інформація у всі доступні поля — як числові, так і текстові. У процесі цього було протестовано реакцію системи на спроби додати символи замість чисел у відповідні поля атрибутів, а також на введення даних, які перевищують дозволені межі або суперечать внутрішній логіці механіки гри. Система спрацювала відповідно до очікувань: вона автоматично заблокувала введення недопустимих символів у числових полях та надала користувачеві чітке попередження з поясненням помилки. Також було перевірено правильність повідомлень про обмеження, коли користувач намагається вийти за межі допустимого обсягу ресурсів або дій у межах одного сценарію.

На рисунку 4.3 наведено приклад повідомлення, що демонструє перевищення допустимого значення ресурсу, а також блокування можливості додавання нової події. Це підтверджує, що система здатна не лише виявляти помилки, а й активно запобігати подальшим некоректним діям, зберігаючи при цьому стабільність своєї

роботи.

Таке функціонування свідчить про високий рівень адаптивності програмного продукту, його відповідність принципам UX-дизайну та орієнтацію на користувача, що особливо важливо в умовах швидкого створення контенту для ігрових сесій.

Рисунок 4.3 – Спроба використати некоректні значення

Окрему увагу в рамках тестування було приділено аналізу можливих ситуацій, які можуть спричинити порушення стабільності або коректності роботи вебзастосунку. З цією метою було реалізовано серію перевірок, спрямованих на виявлення вразливих місць у логіці взаємодії між користувачем і системою, а також на забезпечення стійкої роботи при нестандартних або помилкових вхідних даних. Це включало перевірку на повторне введення, пропущені поля, надмірне навантаження на окремі модулі тощо.

Крім того, проведено всебічне тестування на надійність функціонування вебсайту в цілому, зокрема в умовах підвищеного навантаження або при одночасному використанні кількома користувачами. Також було досліджено

характер і стабільність зв’язку із серверною частиною: перевірялися типи запитів, коректність обробки відповіді від сервера, затримки передачі даних, а також відновлення після тимчасових збоїв з'єднання.

Завдяки цьому вдалося забезпечити високий рівень надійності та стабільності роботи системи, що є критично важливим фактором для інтерактивного вебзастосунку з динамічною архітектурою та багатофункціональним інтерфейсом.

4.2 Розробка інструкції користувача

Керівництво користувача містить інформацію щодо технічних вимог, необхідних для запуску програмного продукту. Відомості про мінімальну та рекомендовану конфігурацію системи наведено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5-10400/AMD Ryzen 5 3600	Intel Core i3-4160/AMD A6-9500
Оперативна пам'ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	1 ГБ	500 МБ
Відеокарта	NVIDIA GeForce GTX 1050/AMD Radeon RX 560	Inter HD Graphics 4400/AMD Radeon R3
Операційна система	Windows 10	Windows 7
Монітор	Роздільна здатність не менше 1920x1080 пікселів або вище	Роздільна здатність не менше 1366x768 пікселів

Оскільки розроблений застосунок функціонує у вебсередовищі, тобто запускається безпосередньо у браузері користувача, основна частина обчислювальних процесів виконується на клієнтській стороні з використанням мови JavaScript. Це дозволяє уникнути необхідності встановлення додаткового програмного забезпечення або драйверів, адже для повноцінної роботи достатньо лише мати сучасний інтернет-браузер — наприклад, Google Chrome, Mozilla Firefox чи Microsoft Edge. Таким чином, застосунок є надзвичайно доступним і невибагливим до технічних характеристик пристрою, що особливо зручно для

широкого кола користувачів, зокрема тих, хто не має потужного обладнання або спеціальних навичок у сфері IT.

Однією з ключових переваг вебзастосунку є його миттєва готовність до роботи після запуску. На відміну від традиційних програмних рішень, що потребують встановлення, завантаження або налаштування додаткових компонентів, цей застосунок функціонує без затримок — користувачеві достатньо відкрити посилання в браузері. Вся необхідна інфраструктура вже інтегрована у вебінтерфейс, тож усі компоненти системи активні одразу після завантаження сторінки. Це є особливо важливим у специфічному контексті створення ігрового контенту для настільних сесій — ситуацій, де час є критичним ресурсом. Відкривши застосунок, користувач одразу може приступити до виконання потрібних завдань: від генерації нових персонажів до створення або симуляції потенційних бойових сцен, які можуть виникнути в рамках ігрового процесу. Завдяки цьому, система забезпечує негайний доступ до основного функціоналу, що, у свою чергу, суттєво економить час, знижує рівень підготовчих операцій і спрощує робочий процес у цілому.

Після проходження авторизації або первинної реєстрації, користувач потрапляє до спеціально адаптованої персоналізованої панелі управління, яка є основною точкою взаємодії з усіма можливостями застосунку. Панель містить зручне, логічно структуроване навігаційне меню, яке дозволяє без труднощів орієнтуватися між функціональними модулями системи. Цей інтерфейс враховує потреби як досвідчених користувачів, так і новачків, які вперше працюють з подібними інструментами. Серед основних модулів платформи варто виділити наступні компоненти: «Головна», «Створення персонажа», «Збережені персонажі», «Калькулятор складності» та «База даних істот». Перемикання між цими модулями здійснюється через верхню панель навігації або за допомогою інтерактивних елементів на головному екрані, які виконують роль швидких дій, значно прискорюючи процес переходу до потрібного розділу.

У модулі «Головна» користувач має змогу ознайомитися з узагальненою інформацією щодо самого застосунку, а також з основними напрямами його

використання. Тут подано відомості про структуру платформи, основний функціонал, відомості про розробника, мету розробки, а також призначення програмного продукту в цілому. Додатково, користувачам доступна коротка довідка, у якій лаконічно, але змістовно пояснено, як взаємодіяти з різними модулями. Такий підхід дає змогу швидко зорієнтуватися у системі, що є особливо важливим для нових користувачів, які лише починають працювати з платформою. Це забезпечує плавне занурення у процес та зменшує бар'єр входу до активного використання всіх інструментів.

Модуль «Створення персонажа» дозволяє користувачеві реалізувати повноцінний процес формування нового ігрового героя згідно з класичними принципами настільної гри Dungeons & Dragons. Інтерфейс модуля побудований поетапно: кожен крок відповідає певному аспекту створення персонажа. Користувач послідовно заповнює дані, які стосуються ключових атрибутів, зокрема таких, як ім'я, клас, рівень, характеристики, походження, володіння зброєю, здібності та інше. Кожен параметр супроводжується підказками або прикладами, що дозволяє краще зрозуміти його значення навіть без попередньої підготовки. Після завершення всіх етапів, створеного персонажа можна зберегти у відповідному розділі, де він буде доступний для подальшого перегляду, редагування або, за потреби, повного видалення.

У модулі «Збережені персонажі» користувач має змогу керувати всіма створеними раніше ігровими героями. Список персонажів подано у вигляді зрозумілої таблиці або карток із базовою інформацією, що дозволяє швидко знайти потрібного. При відкритті конкретного запису користувач може внести зміни до будь-якого параметра персонажа — змінити рівень, відредагувати характеристики, змінити ім'я чи спорядження тощо. За потреби, є можливість повністю видалити запис, якщо персонаж більше не використовується або потребує повного переформатування. Такий підхід забезпечує гнучкість і зручність у роботі з контентом, дозволяє оперативно адаптувати персонажів під різні сценарії та ситуації в межах ігрової кампанії.

Розділ «Калькулятор складності» являє собою інструмент, призначений

передусім для майстрів гри, які займаються підготовкою або моделюванням бойових сцен. Він дозволяє ввести вихідні параметри потенційних ворогів — такі як їхній рівень, кількість, типи характеристик та інші важливі дані. На основі введеної інформації система автоматично розраховує рівень загрози, який ці супротивники становлять для гравців. Це дає змогу оперативно оцінити складність майбутнього бою та за потреби змінити баланс — наприклад, зменшити кількість ворогів, додати союзників або змінити обстановку. Таким чином, цей модуль суттєво спрощує процес планування та балансування ігрових ситуацій, роблячи гру цікавою та викликовою, але справедливою.

Модуль «База даних істот» реалізовано у вигляді зручного пошукового інтерфейсу, який дозволяє швидко отримати доступ до докладної інформації про різноманітних монстрів та істот. Пошук відбувається за назвою істоти, а результати містять ключові параметри — включаючи характеристики, здібності, рівень небезпеки, особливості поведінки та опис зовнішності. Така функціональність особливо цінна під час підготовки до ігрової сесії або у процесі її проведення в реальному часі, коли потрібно швидко знайти відповідного супротивника або істоту для взаємодії з гравцями.

Загалом, інтерфейс вебзастосунку побудований таким чином, щоб бути максимально інтуїтивно зрозумілим. Усі елементи розміщені логічно, наявні численні текстові й візуальні підказки, що суттєво спрощує опанування функціоналу навіть для користувачів без технічного або ігрового досвіду. Завдяки цьому забезпечується комфортне та безбар’єрне використання системи, підвищується якість взаємодії, зменшується ймовірність помилок під час заповнення даних або налаштування параметрів, а загальний користувацький досвід стає позитивним і послідовним.

4.3 Тестування в реальних умовах

Для оцінки дієвості створеного вебзастосунку було проведено експериментальне дослідження, у якому взяли участь реальні користувачі — майстри та гравці настільної рольової гри Dungeons & Dragons. Метою дослідження

було порівняння часу, необхідного для підготовки до ігрової сесії, за умови використання різних підходів. Учасникам запропонували пройти процес підготовки до простої ігрової сесії, скориставшись двома методами: першим був традиційний підхід, до якого вони вже звикли, а другим — новий підхід із використанням створеного вебзастосунку.

Традиційні методи, які часто включають роботу з паперовими носіями [19], використання застарілих або складних у навігації інтерфейсів сторонніх програм, а також ручне створення або опрацювання характеристик персонажів і монстрів, зазвичай потребували значної кількості часу. Згідно з отриманими результатами, середній час, витрачений на підготовку за цим підходом, становив приблизно 1,5 години (90 хвилин).

Натомість новий підхід, заснований на використанні вебзастосунку, розробленого в межах даної роботи, продемонстрував певні переваги. Завдяки простому, інтуїтивно зрозумілому інтерфейсу та структурованій архітектурі, учасники змогли швидше орієнтуватися в інструментах, що були їм доступні. У результаті середній час підготовки скоротився приблизно на 8%, що відповідає економії у 8 хвилин. Таким чином, підготовка до сесії за допомогою вебзастосунку тривала в середньому 82 хвилини (1 година 22 хвилини).

Отримані дані підтверджують, що запропоноване програмне рішення справді сприяє покращенню організації ігрового процесу, зменшуючи навантаження на користувачів під час підготовки. Особливо це стосується таких задач, як створення та редагування ігрових персонажів і монстрів, що зазвичай потребують багато часу при ручному виконанні.

На рисунку 4.4 представлено порівняльну візуалізацію середнього часу, необхідного для підготовки до ігрової сесії за традиційним підходом та за допомогою розробленого вебзастосунку. Графік наочно демонструє позитивну динаміку та дозволяє зробити висновки щодо практичної користі впровадженого рішення..

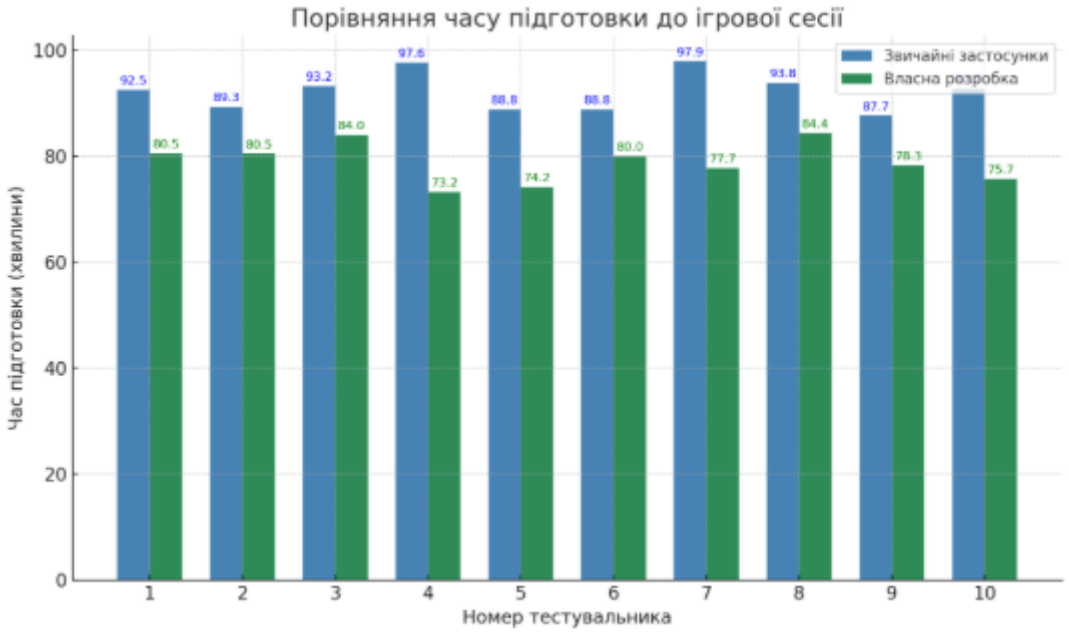

Рисунок 4.4 – Результати тестування

Отже, виходячи з отриманих результатів, можна зробити висновок, що розроблене програмне забезпечення справді сприяє зменшенню часу, необхідного для підготовки до ігрової сесії. Згідно з аналізом, цей показник скоротився приблизно на 8%, що є помітним досягненням. Для майстрів і гравців, які прагнуть максимально зручно та організовано проводити ігровий процес, така економія часу має практичну цінність. Вона дозволяє зосередитись безпосередньо на сюжеті, імпровізації та взаємодії між учасниками, замість витрачання зайвого часу на технічну або підготовчу частину. Таким чином, впровадження цієї розробки покращує загальний досвід гри та робить процес планування більш динамічним і доступним.

4.4 Висновки

У підсумку можна зазначити, що було проведено повноцінне тестування функціональних модулів розробленого програмного забезпечення, зокрема тих, що відповідають за створення персонажів і розрахунок рівня складності супротивників. Особливу увагу було приділено перевірці стійкості системи до некоректних дій користувача, тестуванню обробки вхідних даних різних форматів і виявленню можливих вразливостей. Також розроблено детальну інструкцію

користувача, яка допомагає швидко почати роботу з вебзастосунком і ефективно використовувати всі його можливості.

ВИСНОВКИ

У рамках бакалаврської кваліфікаційної роботи було розроблено програмні модулі, орієнтовані на допомогу рішення логічних задач на прикладі настільної гри Dangeons & Dragons. Розробку здійснено у середовищі Visual Studio Code з дотриманням вимог, визначених у методичних вказівках [20].

У процесі виконання кваліфікаційної роботи було здійснено аналітичний огляд актуального стану проблеми, а також проведено аналіз існуючих аналогів програмного забезпечення. Виявлені недоліки сторонніх рішень стали підґрунтям для формування власного бачення системи. Отримані результати дозволили чітко сформулювати основні цілі та завдання проєкту, що визначили подальший напрям розробки.

Під час аналізу технологій розробки було обґрунтовано вибір мов програмування JavaScript, а саме бібліотеку React, для серверу – платформа Node.js, для бази даних – MySQL, а саме MySQLWorkbench.

У бакалаврській кваліфікаційній роботі було виконано наступне:

- проаналізувано існуючі рішення та визначено їхні обмеження щодо потреб майстрів та гравців настільної гри;

- створено графічний інтерфейс користувача з логічною структурою, вкладками та кнопками швидкого доступу до функцій;

- розроблено модулі для автоматизації підрахункових процесів та автоматизації роботи зі створенням персонажів;

- всі модулі були інтегровані в єдину систему;

- розроблено модулі клієнтської частини, що передбачають виконування певних розрахункових задач;

- створено базу даних для зберігання інформації про користувачів, події та ресурси, з використанням MySQL;

- проведено тестування роботи системи та оцінено її зручність для ефективного виконання поставлених задач та було проведено експерементальну перевірку в реальних умовах.

Тестування довело працездатність веб-застосунку та його відповідність поставленим задачам. Практичне застосування розробленого модуля продемонструвало скорочення часу на виконання базових операцій, якістановлять близько 10% робочого процесу, що дозволяє економити до 8% загального часу роботи. Отже, мету бакалаврської кваліфікаційної роботи досягнуто.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Байдалюк В. О. Розробка програмних модулів комп’ютерної системи для розвитку логічних здібностей на прикладі настільної гри. Матеріали Міжнародної науково-практичної конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2025) [Електронний ресурс] Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/24796/20558.

2. Wizards of the Coast. Dungeon Master's Guide. Renton: Wizards of the Coast LLC, 2024. 15 с.

3. Wizards of the Coast. Dungeon Master's Guide. Renton: Wizards of the Coast LLC, 2024. 80 с.

4. Obsidian [Електронний ресурс] – Режим доступу: https://obsidian.md/

5. Dungeonscrawl.com [Електронний ресурс] – Режим доступу до ресурсу: https://www.dungeonscrawl.com/

6. Donjon.bin.sh [Електронний ресурс] – Режим доступу до ресурсу: Donjon.bin.sh

7. Kobold Fight Club [Електронний ресурс] – Режим доступу до ресурсу: https://koboldplus.club/

8. JavaFX [Електронний ресурс] – Режим доступу: https://www.kievoit.ippo.edu.ua

9. SQL [Електронний ресурс] – Режим доступу: https://freehost.com.ua

10. Threat level Wizards of the Coast. Dungeon Master's Guide. Renton: Wizards of the Coast LLC, 2024. 99 с.

11. Spring Boot [Електронний ресурс] – Режим доступу: https://azure.micrisoft.com

12. Python [Електронний ресурс] – Режим доступу: https://docs.pyton.org

13. C++ [Електронний ресурс] – Режим доступу: https://acode.com.ua/

14. Кросплатформеність [Електронний ресурс] – Режим доступу: https://fit.knu.ua

15. VS Code [Електронний ресурс] – Режим доступу: https://vscode.dev

16. Методи створення рівня складності [Електронний ресурс] – Режим доступу: https://foxminded.ua

17. Weinberg G. Perfect Software And Other Illusions About Testing. "Addison-Wesley Professional". 2011. 224 p.

18. Тестування [Електронний ресурс] – Режим доступу: https://mate.academy

19. Wizards of the Coast. Dungeon Master's Guide. Renton: Wizards of the Coast LLC, 2024. 11 c.

20. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад. Романюк О. Н., Черноволик Г. О. – Вінниця: ВНТУ, 2022. – 52 с.

ДОДАТКИ

61

Додаток А – Технічне завдання

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

ЗАТВЕРДЖУЮ
д.т.н., проф. Романюк О.Н.
«25» березня 2025 р.

Технічне завдання
На бакалаврську кваліфікаційну роботу «Розробка програмних засобів для сприйняття розвитку логічних здібностей на прикладі настільної гри Dangeons & Dragons» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., О. М. Ткаченко
«24» березня 2025 р.
Виконав:
студент гр.5ПІ-21б, В. О. Байдалюк
«24» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмних засобів для сприйняття розвитку логічних здібностей на прикладі настільної гри Dangeons & Dragons».

Галузь застосування – навчання та розвиток.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «21» березня 2025 р. ректора ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки

Метою роботи є процес пришвидшення створення ігових сесій, за рахунок використання програми, що дає можливість автоматизувати процеси та спростити інтерфейс. Завдяки такій системі кожен користувач може швидко створити необхідних персонажів, та провести перевірку, що в свою чергу дозволяє покращити свої навички в аналізі.

4. Технічні вимоги

Вихідні дані до роботи: середовище розробки Visual studio Code, мова розробки – Java cript, операційна система – Windows.

5. Конструктивні вимоги.

Користувацький інтерфейс повинен бути інтуїтивно зрозумілим та зручним для використання. Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до БКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії і етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1.	Аналітичний огляд вебзастосунків для гравців та аналіз допоміжних інструментів	25.03.2025 – 07.04.25
2.	Реалізація методів створення та обробки персонажів, розрахунку рівня небезпеки та обробка архітектури програмної системи	08.04.25 – 22.04.25
3.	Розробка інтерактивного вебзастосунку обробки та розрахунку створених персонажів	02.05.25 – 01.05.25
4.	Тестування програми	02.05.25 – 14.05. 25

9. Порядок контролю і приймання

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

Додаток Б – ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмних засобів для сприяння розвитку логічних здібностей на прикладі настільної гри Dangeons & Dragons

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКИ, 5ПІ-21Б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 3.3 %

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(-на)

Керівник _____ к.т.н., доц. каф. ПЗ Ткаченко О.М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Байдалюк В. О.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

Лістинг App.js:

```
// src/App.js
import React from "react";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";

import NavBar from "./components/NavBar/NavBar";
import InformationPage from "./pages/InformationPage";
import CreateCharacterPage from "./pages/CreateCharacterPage";
import SavedCharactersPage from "./pages/SavedCharactersPage";
import DifficultyCalculatorPage from "./pages/DifficultyCalculatorPage";
import CreatureDatabasePage from "./pages/CreatureDatabasePage";
import HomePage from "./pages/HomePage";


const App = () => {
return (
<Router>
<NavBar />
<Routes>
<Route path="/" exact element={<HomePage />} />
<Route path="/create-character" element={<CreateCharacterPage />} />
<Route path="/saved-characters" element={<SavedCharactersPage />} />
<Route path="/difficulty-calculator" element={<DifficultyCalculatorPage />} />
<Route path="/information" element={<InformationPage />} />
<Route path="/creature-database" element={<CreatureDatabasePage />} />
</Routes>
</Router>
);
};
```

Лістинг index

```
import React from 'react';
```

```
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import store from './redux/store';
import App from './App';
import './styles/main.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

Лістинг main.css:

```
body {
  background-color: #F3F6F9;
  font-family: sans-serif;
}

body, h1, h2, h3, h4, h5, h6 {
  color: #2B2B2B;
}



/* Accent Elements */
button,
.active-link {
  background-color: #567DFA;
  color: white;
  border: none;
  padding: 10px 20px;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s ease;
```

```
}

button:hover {
  background-color: #3c56b5;
}

/* Content Blocks */
.block-1 {
  background-color: #CDEDFE;
  border: 2px solid #9AD0EC;
  box-shadow: 0 4px 8px rgba(59, 110, 143, 0.2);
  color:#3B6E8F;
}

.block-2 {
  background-color: #F7E2C4;
  border: 2px solid #EAC9A3;
  box-shadow: 0 4px 8px rgba(138, 98, 62, 0.2);
  color:#8A623E;
}

.block-3 {
  background-color: #E4FCD8;
  border: 2px solid #C7E8B4;
  box-shadow: 0 4px 8px rgba(79, 120, 74, 0.2);
  color:#4F784A;
}

.block-4 {
  background-color: #F9D9E2;
  border: 2px solid #F1B6C6;
  box-shadow: 0 4px 8px rgba(161, 76, 97, 0.2);
  color:#A14C61;
}
```

```
/* Info Block */
.info-block {
  background-color: #F9D9E2; /* Light background */
  border: 2px solid #F1B6C6; /* Border color */
  box-shadow: 0 4px 8px rgba(161, 76, 97, 0.2); /* Shadow */
  color: #A14C61; /* Text color */
  padding: 10px; /* Padding */
  margin: 10px 0; /* Margin */
  border-radius: 5px;
}

/* Navigation Bar */
.navbar {
  display: flex;
  gap: 20px;
  margin-bottom: 20px;
  padding: 10px;
  background-color: #f0f0f0;
}

/* Forms */
input[type="text"],
select {
  padding: 8px;
  margin: 5px 0;
  border: 1px solid #ccc;
  border-radius: 4px;
}
```

Лістинг action.js:

```
export const ADD_CHARACTER = 'ADD_CHARACTER';
export const DELETE_CHARACTER = 'DELETE_CHARACTER';
export const UPDATE_CHARACTER = 'UPDATE_CHARACTER';

// Action Creators
export const addCharacter = (character) => ({
```

```
  type: ADD_CHARACTER,
  payload: character,
});

export const deleteCharacter = (id) => ({
  type: DELETE_CHARACTER,
  payload: id,
});
export const updateCharacter = (id, updatedCharacter) => ({
  type: UPDATE_CHARACTER,
  payload: { id, updatedCharacter },
});
```

Лістинг reducers:

```
const initialState = {
  characters: [],
};

const generateUniqueId = () => {
  return Math.random().toString(36).substring(2, 9);
};

const rootReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'ADD_CHARACTER':
      return {
        ...state,
        characters: [...state.characters, action.payload],
      };
    case 'DELETE_CHARACTER':
      return {
        ...state,
        characters: state.characters.filter(character => character.id !== action.payload)
      }
        case "UPDATE_CHARACTER":
```

```
      return {
        ...state,
        characters: state.characters.map(character => {
          if (character.id === action.payload.id) {
            return { ...character, ...action.payload.updatedData };
          }
          return character;
        })
      };
    case "ADD_CHARACTER":
      return {
        ...state,
        characters: [...state.characters, { ...action.payload, id: generateUniqueId() }]
      };

    default:
      return state;
  }
};

export default rootReducer;
```

Лістинг store:

```
import { configureStore } from '@reduxjs/toolkit'
import rootReducer from './reducers'

const store = configureStore({
  reducer: rootReducer,
})

export default store
```

Лістинг dataService:

```
/**
 * Gets the characters from localStorage.
 * @returns {Array<Object>} The array of characters.
 */
export const getCharacters = () => {
  try {
    const characters = localStorage.getItem('characters');
    return characters ? JSON.parse(characters) : [];
  } catch (error) {
    console.error('Error getting characters:', error);
    return [];
  }
};

/**
 * Saves a character to localStorage.
 * @param {Object} character - The character to save.
 */
export const saveCharacter = (character) => {
  try {
    const characters = getCharacters();
    localStorage.setItem('characters', JSON.stringify([...characters, character]));
    console.log('Character saved successfully!');
  } catch (error) {
    console.error('Error saving character:', error);
  }
};

/**
 * Updates a character in localStorage.
 * @param {string} id - The ID of the character to update.
 * @param {Object} updatedCharacter - The updated character data.
 * @returns {boolean} - True if updated, false if not found
 */
export const updateCharacter = (id, updatedCharacter) => {
```

```
  try {
    const characters = getCharacters();
    const index = characters.findIndex((char) => char.id === id);
    if (index !== -1) {
      characters[index] = { ...characters[index], ...updatedCharacter };
      localStorage.setItem('characters', JSON.stringify(characters));
      return true;
    }
    return false;
  } catch (error) {
    console.error('Error updating character:', error);
    return false;
  }
};

/**
 * Deletes a character from localStorage.
 * @param {string} id - The ID of the character to delete.
 * @returns {boolean} True if deleted, false if not found
 */
export const deleteCharacter = (id) => {
  try {
    const characters = getCharacters();
    const filteredCharacters = characters.filter((char) => char.id !== id);
    if (characters.length > filteredCharacters.length) {
      localStorage.setItem('characters', JSON.stringify(filteredCharacters));
      return true;
    }
    return false;
  } catch (error) {
    console.error('Error deleting character:', error);
    return false;
  }
};
```

```
/**
 * Gets the creatures from localStorage.
 * @returns {Array<Object>} The array of creatures.
 */
export const getCreatures = () => {
  try {
    const creatures = localStorage.getItem('creatures');
    return creatures ? JSON.parse(creatures) : [];
  } catch (error) {
    console.error('Error getting creatures:', error);
    return [];
  }
};
```

Лістинг NavBar.js:

```
// src/components/NavBar/NavBar.js
import React from 'react';
import { NavLink } from 'react-router-dom';
import '../../styles/main.css';
import './NavBar.css';

const NavBar = () => {
    return (
        <nav className="nav-bar">
            <NavLink to="/" className={({ isActive }) => isActive ? 'nav-link active-nav-link' : 'nav-link'}>Home</NavLink>
            <NavLink to="/create-character" className={({ isActive }) => isActive ? 'nav-link active-nav-link' : 'nav-link'}>Create Character</NavLink>
            <NavLink to="/saved-characters" className={({ isActive }) => isActive ? 'nav-link active-nav-link' : 'nav-link'}>Saved Characters</NavLink>
            <NavLink to="/difficulty-calculator" className={({ isActive }) => isActive ? 'nav-link active-nav-link' : 'nav-link'}>Difficulty Calculator</NavLink>
            <NavLink to="/creature-database" className={({ isActive }) => isActive ? 'nav-link active-nav-link' : 'nav-link'}>Creature Database</NavLink>
            <NavLink to="/information" className={({ isActive }) => isActive ? 'nav-link active-
```

```
nav-link' : 'nav-link'}>Інформація</NavLink>
            </nav>
        );
    };

    export default NavBar;
```

Лістинг CreateCharacter.js:

```
// src/components/CreateCharacter/CreateCharacter.js
import React, { useState } from 'react';
import { useForm } from 'react-hook-form';
import { useDispatch } from 'react-redux';
import * as yup from 'yup';
import { saveCharacter } from '../../api/dataService';
import { yupResolver } from '@hookform/resolvers/yup';
import { addCharacter } from '../../redux/actions';
import './CreateCharacter.css';

const schema = yup.object().shape({
  id: yup.string(),
  name: yup.string().required('Name is required'),
  characterClass: yup.string().required('Class is required'),
  ac: yup.number().required('Armor Class is required').positive().integer(),
  strength: yup.number().required('Strength is required').positive().integer(),
  dexterity: yup.number().required('Dexterity is required').positive().integer(),
  constitution: yup.number().required('Constitution is required').positive().integer(),
  intelligence: yup.number().required('Intelligence is required').positive().integer(),
  wisdom: yup.number().required('Wisdom is required').positive().integer(),
  charisma: yup.number().required('Charisma is required').positive().integer(),
  speed: yup.number().required('Speed is required').positive().integer(),
  hp: yup.number().required('Hit Points are required').positive().integer(),
});

function CreateCharacter() {
  const { register, handleSubmit, formState: { errors }, reset } = useForm({
```

```
  resolver: yupResolver(schema),
 });
 const [successMessage, setSuccessMessage] = useState('');
 const dispatch = useDispatch();

 const onSubmit = async (data) => {
  const newCharacter = { ...data, id: Date.now().toString() };
  try {
   await saveCharacter(newCharacter);
   dispatch(addCharacter(newCharacter));
   setSuccessMessage('Character created successfully!');
   setTimeout(() => setSuccessMessage(''), 3000); // Clear after 3 seconds
   reset();
  } catch (error) {
   console.error("Failed to save character", error);
  }
 };

 return (
  <div className="block-2-container">
   <h2>Create New Character</h2>
   <form onSubmit={handleSubmit(onSubmit)} className="create-character-form">
    <input type="text" name="name" {...register("name")} placeholder="Name"
className="form-input" />
    {errors.name && <p className="form-error">{errors.name.message}</p>}

    <select name="characterClass" {...register("characterClass")} className="form-select">
     <option value="">Select Class</option>
     <option value="warrior">Warrior</option>
     <option value="mage">Mage</option>
     <option value="примара">Примара</option>
     <option value="демон">Демон</option>
     <option value="створіння">Створіння</option>
     <option value="тварина">Тварина</option>
     <option value="голем">Голем</option>
```

```
            </select>
            {errors.characterClass && <p className="form-
error">{errors.characterClass.message}</p>}

            <input type="number" name="ac" {...register("ac")} placeholder="Armor Class (AC)"
className="form-input" />
            {errors.ac && <p className="form-error">{errors.ac.message}</p>}

            <input type="number" name="strength" {...register("strength")} placeholder="Strength"
className="form-input" />
            {errors.strength && <p className="form-error">{errors.strength.message}</p>}

            <input type="number" name="dexterity" {...register("dexterity")}
placeholder="Dexterity" className="form-input" />
            {errors.dexterity && <p className="form-error">{errors.dexterity.message}</p>}

            <input type="number" name="constitution" {...register("constitution")}
placeholder="Constitution" className="form-input" />
            {errors.constitution && <p className="form-error">{errors.constitution.message}</p>}

            <input type="number" name="intelligence" {...register("intelligence")}
placeholder="Intelligence" className="form-input" />
            {errors.intelligence && <p className="form-error">{errors.intelligence.message}</p>}

            <input type="number" name="wisdom" {...register("wisdom")} placeholder="Wisdom"
className="form-input" />
            {errors.wisdom && <p className="form-error">{errors.wisdom.message}</p>}

            <input type="number" name="charisma" {...register("charisma")}
placeholder="Charisma" className="form-input" />
            {errors.charisma && <p className="form-error">{errors.charisma.message}</p>}

            <input type="number" name="speed" {...register("speed")} placeholder="Speed"
className="form-input" />
            {errors.speed && <p className="form-error">{errors.speed.message}</p>}
```

```
            <input type="number" name="hp" {...register("hp")} placeholder="Hit Points (HP)"
classNameName="form-input" />
            {errors.hp && <p classNameName="form-error">{errors.hp.message}</p>}

            <button type="submit" classNameName="form-button">Create Character</button>
          </form>
          {successMessage && (
            <div classNameName="success-message">
              {successMessage}
            </div>
          )}
        </div>
      );
    }
    export default CreateCharacter;
```

Лістинг SavedCharacters.js:

```
    import React, { useEffect, useState } from 'react';
    import { useSelector, useDispatch } from 'react-redux';
    import { deleteCharacter, updateCharacter, setCharacters } from '../../redux/actions';
    import './SavedCharacters.css';
    import { getCharacters, updateCharacter as updateCharacterApi} from '../../api/dataService';
    import { useForm } from 'react-hook-form';


    const SavedCharacters = () => {
      const characters = useSelector((state) => state.characters);
      const [editingCharacterId, setEditingCharacterId] = useState(null);
      const dispatch = useDispatch();
      const { register, handleSubmit, reset, formState: { errors } } = useForm();


       useEffect(() => {
          const loadCharacters = async () => {
```

```
        try {
            const fetchedCharacters = await getCharacters();
            dispatch(setCharacters(fetchedCharacters));
        } catch (error) {
            console.error("Failed to load characters", error);
        }
    };
    loadCharacters();
  }, [dispatch]);

const deleteCharacterHandler = (id) => {
  dispatch(deleteCharacter(id));
};

const editCharacterHandler = (id) => {
  setEditingCharacterId(id);
};

const editingCharacter = characters.find(char => char.id === editingCharacterId);

const onSubmit = async (data) => {
  try {
    await updateCharacterApi(editingCharacterId, data);
    dispatch(updateCharacter(editingCharacterId, data));
    setEditingCharacterId(null);
  } catch (error) {
    console.error("Failed to update character", error);
  }
};

if (characters.length === 0) {
  return <div><h2>Saved Characters</h2><p>No characters saved yet.</p></div>
}
```

```
        return (
          <div className="saved-characters-container block-1">
            <h2 className="saved-characters-heading">Saved Characters</h2>
              {editingCharacterId && editingCharacter ? (
                <form className="edit-character-form" onSubmit={handleSubmit(onSubmit)}>
                  <input className="edit-character-input" type="text" {...register("name")}
defaultValue={editingCharacter.name} placeholder="Name" />
                  <select className="edit-character-select" {...register("class")}
defaultValue={editingCharacter.class}>
                    <option value="">Select Class</option>
                    <option value="warrior">Warrior</option>
                    <option value="mage">Mage</option>
                  </select>

                  <button className="edit-character-button accent-button" type="submit">Update
Character</button>
                  <button className="edit-character-button secondary-button" onClick={() =>
setEditingCharacterId(null)}>Cancel</button>
                </form>
              ) : (
                <div className="character-list">
                  {characters.map((character, index) => (
                    <div key={character.id} className="character-card card block-1-light">
                      <h3 className="character-name">{character.name}</h3>
                      <p className="character-class">Class: {character.class}</p>
                      <p>AC: {character.AC}</p>
                                <p>Strength: {character.Strength}</p>
                                <p>Dexterity: {character.Dexterity}</p>
                                <p>Constitution: {character.Constitution}</p>
                                <p>Intelligence: {character.Intelligence}</p>
                                <p>Wisdom: {character.Wisdom}</p>
                                <p>Charisma: {character.Charisma}</p>
                                <p>Speed: {character.Speed}</p>
                                <p>HP: {character.HP}</p>
                      <div className="card-buttons">
```

```
                                <button
                                    className="edit-button accent-button"
                                    onClick={() => editCharacterHandler(character.id)}
                                >Edit</button>
                    <button className="delete-button secondary-button" onClick={() =>
deleteCharacterHandler(character.id)}>Delete</button>
                  </div>
                 </div>
               ))}
              </div>
            )}
          </div>
        );
      };

      export default SavedCharacters;
```

Лістинг CreatureDatabase.js:

```
      import React, { useState, useEffect } from 'react';
      import { getCreatures } from '../../api/dataService';

      function CreatureDatabase() {
        const [filter, setFilter] = useState('');
        const [creatures, setCreatures] = useState([]);
        const [filteredCreatures, setFilteredCreatures] = useState([]);

        useEffect(() => {
          const fetchCreatures = async () => {
            try {
              const allCreatures = await getCreatures();
              setCreatures(allCreatures);
            } catch (error) {
              console.error("Failed to fetch creatures", error);
            }
          };
```

```
    fetchCreatures();
  }, []);

  useEffect(() => {
    const filtered = creatures.filter(creature =>
      creature.name.toLowerCase().includes(filter.toLowerCase())
    );
    setFilteredCreatures(filtered);
  }, [filter, creatures]);

  return (
    <div className="creature-database-container block-4">
      <h2 className='accent-text'>Creature Database</h2>
      <input
        className="creature-filter-input"
        type="text"
        value={filter}
        onChange={(e) => setFilter(e.target.value)}
        placeholder="Search creatures"
      />
      <div className="creature-list">
      {filteredCreatures.map((creature) => (
          <div key={creature.id} className="creature-card">
            <h3>{creature.name}</h3><p>{creature.description}</p>
            </div>
        ))}
      </div>
    </div>
  );
}

export default CreatureDatabase;
// This is an empty file for the CreatureDatabase component.
```

Лістинг package.json:

```
{
  "name": "my-app",
  "version": "0.1.0",
  "dependencies": {
    "@hookform/resolvers": "^5.0.1",
    "@reduxjs/toolkit": "^2.7.0",
    "react-hook-form": "^7.56.1",
    "react-redux": "^9.2.0",
    "react-router-dom": "^7.5.3",
    "yup": "^1.6.1"
  },
  "scripts": {
    "start": "react-scripts start"
  },
  "devDependencies": {
    "react-scripts": "^5.0.1"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Лістинг app:

```
. // src/App.js
import React from "react";
```

```
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";

import NavBar from "./components/NavBar/NavBar";
import InformationPage from "./pages/InformationPage";
import CreateCharacterPage from "./pages/CreateCharacterPage";
import SavedCharactersPage from "./pages/SavedCharactersPage";
import DifficultyCalculatorPage from "./pages/DifficultyCalculatorPage";
import CreatureDatabasePage from "./pages/CreatureDatabasePage";
import HomePage from "./pages/HomePage";


const App = () => {
   return (
      <Router>
         <NavBar />
         <Routes>
      <Route path="/" exact element={<HomePage />} />
      <Route path="/create-character" element={<CreateCharacterPage />} />
      <Route path="/saved-characters" element={<SavedCharactersPage />} />
      <Route path="/difficulty-calculator" element={<DifficultyCalculatorPage />} />
      <Route path="/information" element={<InformationPage />} />
      <Route path="/creature-database" element={<CreatureDatabasePage />} />
     </Routes>
      </Router>
   );
};

export default App;
```

Лістинг informationPage.js:

```
// src/pages/InformationPage.js
import React from 'react';
import Home from '../components/Home/Home';

function InformationPage() {
  return (
```

```
      <div className="page-container block-1">
        <Home />
        <p className="info-block">Ознайомтеся з інформацією про розробку та цілі даного проєкту.</p>
      </div>
    );
  }

  export default InformationPage;
```

Лістинг SavedCharacters.css:

```
/* src/components/SavedCharacters/SavedCharacters.css */

.saved-characters-container {
  background-color: #CDEDFE; /* Main background color from Block 1 */
  padding: 20px;
  border-radius: 8px;
}

.character-list {
  list-style: none;
  padding: 0;
}

.character-item {
  background-color: #9AD0EC; /* Background for individual character items */
  border: 1px solid #3B6E8F; /* Border color from Block 1 */
  color: #3B6E8F; /* Text color from Block 1 */
  margin-bottom: 10px;
  padding: 15px;
  border-radius: 5px;
}

.character-item h3 {
  color: #3B6E8F; /* Heading color within item */
```

```
    margin-top: 0;
}

.character-item p {
    margin-bottom: 5px;
    font-size: 0.9em;
}

.character-item p strong {
    color: #2B2B2B; /* Use text color for labels for better contrast */
}

/* Add styles for buttons if any in this component */
.saved-characters-container button {
    background-color: #567DFA; /* Accent color for buttons */
    color: white;
    border: none;
    padding: 8px 15px;
    border-radius: 5px;
    cursor: pointer;
    margin-right: 5px;
}

.saved-characters-container button:hover {
    opacity: 0.9;
}
```

Лістинг createCharacters.css:

```
import { useEffect, useState } from "react";
/* src/components/CreateCharacter/CreateCharacter.css */

.create-character-container {
    background-color: #F7E2C4; /* Light background for the container */
    padding: 20px;
    border-radius: 8px;
```

```
    max-width: 600px;
    margin: 20px auto;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    color: #2B2B2B; /* Default text color from main palette */
}

.create-character-container h2 {
    color: #8A623E; /* Darker accent for headings */
    margin-bottom: 20px;
    text-align: center;
}

.create-character-form label {
    display: block;
    margin-bottom: 8px;
    font-weight: bold;
    color: #8A623E; /* Darker accent for labels */
}

.create-character-form input[type="text"],
.create-character-form input[type="number"],
.create-character-form select {
    width: 100%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #8A623E; /* Border color */
    border-radius: 4px;
    background-color: #EAC9A3; /* Background for input fields */
    color: #2B2B2B; /* Text color */
    box-sizing: border-box; /* Include padding and border in element's total width and height */
}

.create-character-form button {
    display: block;
    width: 100%;
```

```
    padding: 12px;
    background-color: #567DFA; /* Accent color for buttons */
    color: white;
    border: none;
    border-radius: 4px;
    font-size: 16px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

.create-character-form button:hover {
    background-color: #3b5cd3; /* Slightly darker shade on hover */
}

.success-message {
    margin-top: 20px;
    padding: 10px;
    background-color: #E4FCD8; /* Color from Block 3 - potentially for success, as green */
    color: #4F784A; /* Darker green for text */
    border: 1px solid #4F784A;
    border-radius: 4px;
    text-align: center;
}
```

Додаток Г – Ілюстративний матеріал

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

Бакалаврська дипломна робота на тему:
«Розробка програмних засобів для сприяння розвитку логічних здібностей на прикладі настільної гри Dangeons & Dragons.»

Автор: ст. гр. 5ПІ-21б Байдалюк В. О.
Науковий керівник: к.т.н., доц. каф. ПЗ Ткаченко О.М.

м.Вінниця - 2025

Рисунок Г.1 – Титульний слайд

Актуальність теми

Dungeons & Dragons — популярна настільна гра, що розвиває логіку, стратегічне мислення та навички прийняття рішень. Її активно використовують не лише для дозвілля, а й у навчальних цілях. Водночас підготовка сесії вимагає значних витрат часу: розрахунок рівня небезпеки, генерація персонажів і супротивників, перевірка балансу. Існуючі цифрові інструменти не завжди зручні — обмежений функціонал, складне редагування, орієнтація на англомовну аудиторію. Це створює потребу в україномовному вебзастосунку, який автоматизує ключові етапи підготовки та дозволяє зберігати і повторно використовувати створені матеріали.

Рисунок Г.2 – Актуальність теми

Мета, об’єкт та предмет дослідження

- Метою бакалаврської кваліфікаційної роботи є зменшення часу створення ігрової сесії настільної гри Dangeons & Dragons за рахунок автоматизаціїї процесу створення персонажів.
- Об'єкт дослідження - є процес розробки модулів комп'ютерної системи для організації часу та менеджменту для людей які беруть участь у розробці ігрової партії.
- Предметом дослідження є методи та програмні засоби, що забезпечують ефективне створення атрибутів та виконання розрахунків з урахуванням індивідуальних потреб користувачів.

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

Задачі дослідження

- проаналізувано існуючі рішення та визначено їхні обмеження щодо потреб майстрів та гравців настільної гри;
- створено графічний інтерфейс користувача з логічною структурою, вкладками та кнопками швидкого доступу до функцій;
- розроблено модулі для автоматизації підрахункових процесів та автоматизації роботи зі створенням персонажів;
- всі модулі були інтегровані в єдину систему;
- розроблено модулі клієнтської частини, що передбачають виконування певних розрахункових задач;
- створено базу даних для зберігання інформації про користувачів, події та ресурси, з використанням MySQL;
- проведено тестування роботи системи та оцінено її зручність для ефективного виконання поставлених задач та було проведено експерементальну перевірку в реальних умовах.

Рисунок Г.4 – Задачі дослідження

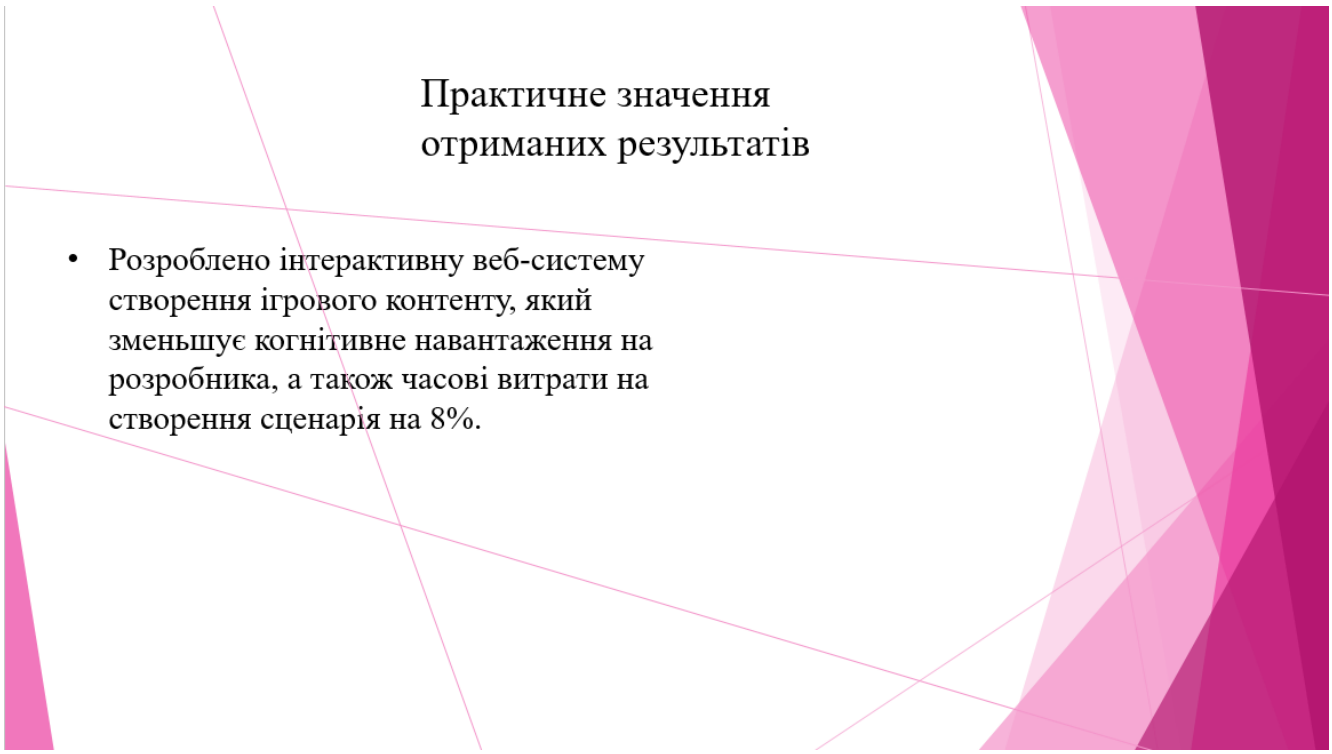

Рисунок Г.5 – Практичне значення отриманих результатів

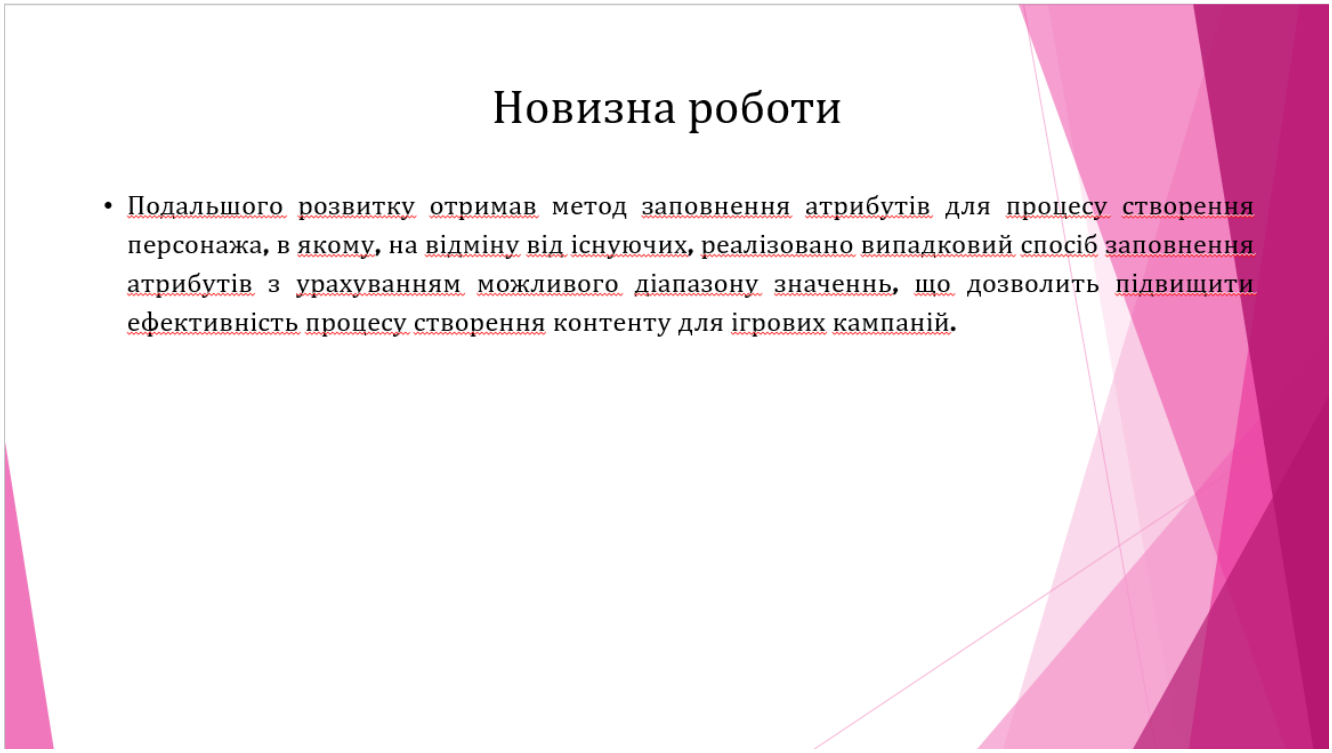

Рисунок Г.6 – Новизна роботи

Порівняльний аналіз з аналогами

Критерії	Kobold Fight Club	Donjon	bin.sh	Dungeon Scrawl
Система для створення нових персонажів	-	+	+	+
Встроєна можливість опису	-	-	+	-
Калькулятор для підрахування рівня небезпеки	+	-	-	-
автономність	-	-	-	-
Простий інтерфейс	+	+	-	-

Рисунок Д.7 – Порівняльний аналіз з аналогами

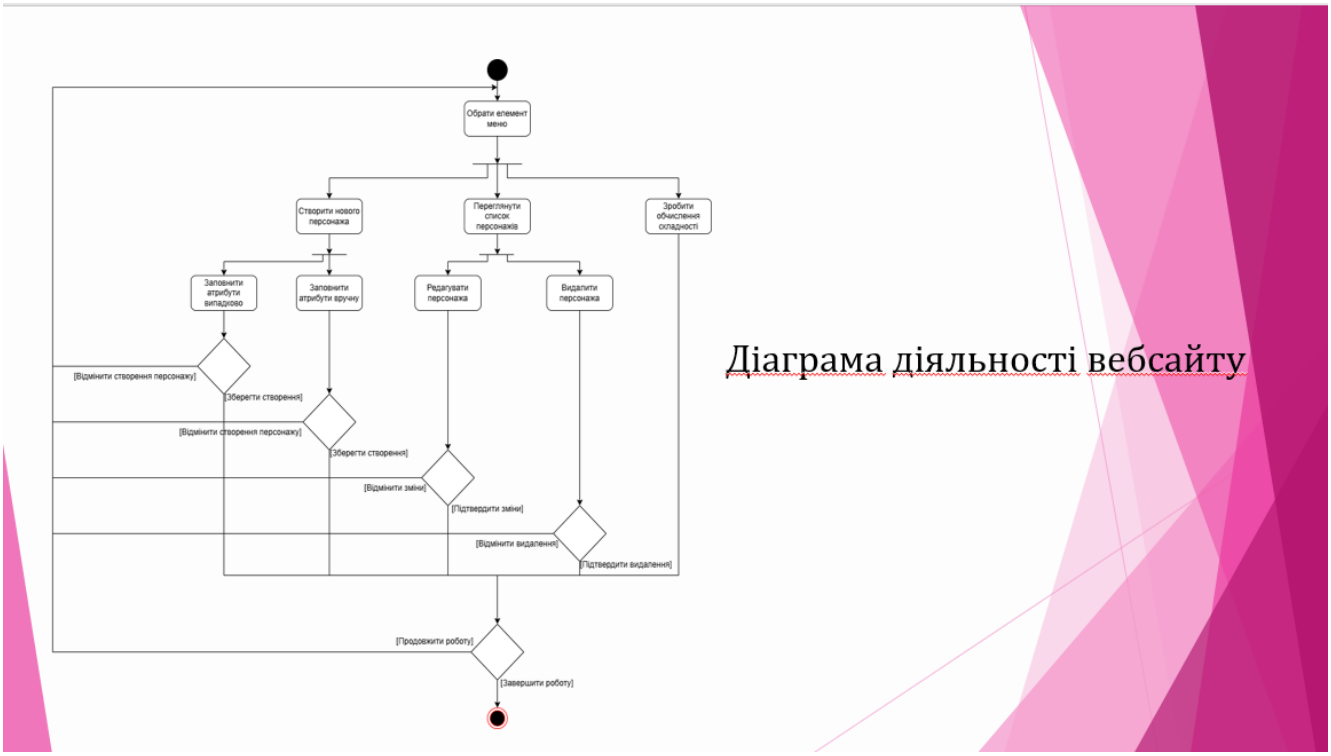

Рисунок Г.8 – UML-діаграма діяльності вебсайту

Блок-схема алгоритму роботи програмного модуля

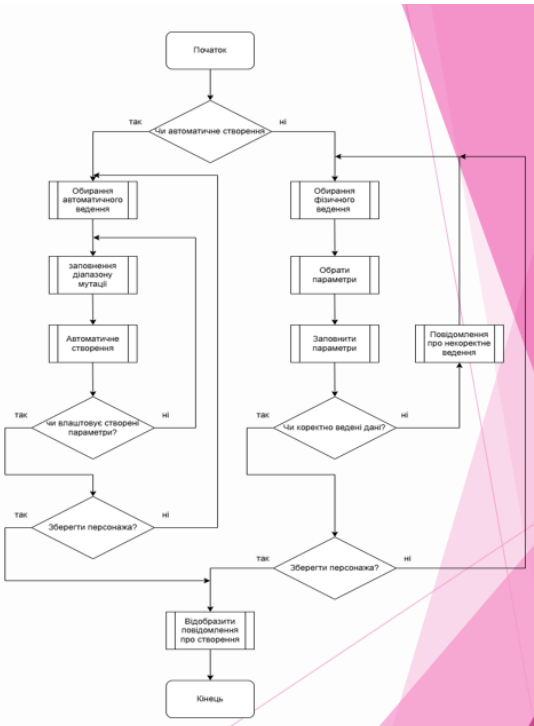

Рисунок Г.9 – Блок-схема алгоритму роботи програмного модуля

Удосконалений метод випадкової зміни

Одним із важливих модулів у системі є модуль для створення персонажів, який забезпечує користувача можливістю формування базових ігрових сутностей — героїв, що використовуються в рамках логічної гри. Основна функціональність модуля полягає у введенні ключових характеристик персонажа та збереженні їх до відповідної структури в базі даних.

Структура класу CharacterModel включає в себе низку основних характеристик, необхідних для повноцінного представлення персонажа у настільній грі або програмному середовищі, що моделює механіку Dungeons & Dragons. Ці характеристики мають тип цілих чисел (int) і охоплюють такі параметри, як сила (Strength), спритність (Dexterity), інтелект (Intelligence), витривалість (Endurance), харизма (Charisma) та мудрість (Wisdom).

Рисунок г.10 – Удосконалений метод випадкової зміни

Розробка модуля для збереження персонажів

Редагування збережених персонажів у веб-застосунку реалізовано у вигляді REST-сервісу, який дозволяє оновити вже існуючого персонажа за його унікальним ідентифікатором. Основу становить клас Character, який є відображенням таблиці бази даних SQL, де зберігаються всі атрибути персонажа: ім'я, основні характеристики, клас та опис. Всі поля цього класу редаговані, що дозволяє вільно оновлювати значення на нові, введені користувачем через фронтенд.

Контролер отримує нові дані через PUT-запит, і якщо відповідний запис існує в базі, він оновлюється за допомогою JpaRepository. Оновлення відбувається через стандартні Java методи set, які змінюють значення полів у моделі. Після цього результат зберігається у базу, і оновлений об'єкт повертається у відповідь.

Така структура дозволяє інтегруватися з будь-яким сучасним фронтендом (React, Vue, Angular) через REST API і дає змогу легко редагувати дані в базі, дотримуючись принципів MVC-архітектури.

Рисунок Г.11 – Розробка модуля для збереження персонажів

Розробка модуля для калькулятора складності

У межах програмної системи було реалізовано окремий модуль, відповідальний за обчислення рівня складності бойових зіткнень (Encounter Difficulty Rating, скорочено — EDR), що дозволяє оцінити співвідношення між бойовою силою противників і потенціалом гравців. Цей функціонал є важливим для формування збалансованих ігрових ситуацій та побудований на використанні формалізованої математичної моделі[16].

Суть роботи модуля полягає у збиранні необхідних параметрів про ворогів (рівень складності, тип, кількість) та гравців (рівень, кількість), обчисленні індивідуальних вагових коефіцієнтів для кожного ворога та, відповідно, підрахунку загального EDR за формулою.

Обчислення EDR здійснюється на основі даних, отриманих із бази даних, де зберігається інформація про створених персонажів. Відповідно, модуль взаємодіє з системою збережених даних, витягуючи з неї лише ті записи, які мають стосунок до поточного зіткнення. Детальний приклад такого витягування даних представлено на рис. 3.4. У ньому реалізовано механізм формування SQL-запиту з урахуванням переданого списку ідентифікаторів персонажів. Після встановлення з'єднання з MySQL-сервером виконується запит до таблиці з персонажами, а результати зберігаються у вигляді списку словників. Такий підхід забезпечує як гнучкість у роботі з даними, так і захист від можливих SQL-ін'єкцій.

```
characters_data = []
try:
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor(dictionary=True)
    if character_ids:
        placeholders = ', '.join(['%s'] * len(character_ids))
        query = f"SELECT * FROM characters WHERE id IN ({placeholders})"
        cursor.execute(query, tuple(character_ids))
        characters_data = cursor.fetchall()
except mysql.connector.Error as err:
    print(f"Database error: {err}")
finally:
    if 'conn' in locals() and conn.is_connected():
        cursor.close()
        conn.close()
return characters_data
```

Рисунок Г.12 – Розробка модуля для калькулятора складності

Розробка інтерфейсу вебдодатку

Вікно створення персонажа має всі необхідні підказки для швидкого заповнення назви атрибутів та опису персонажа, такоє кнопка створення реалізовує перенесення інформації про створенного персонажа в вікно «створенні персонажі»
Вікно створених персонажів має список усіх створених персонажів для перегляду, редагування та видалення персонажів, які були створенні попередньо.

Рисунок Г.13 – Розробка інтерфейсу вебдодатку

Тестування в реальних умовах

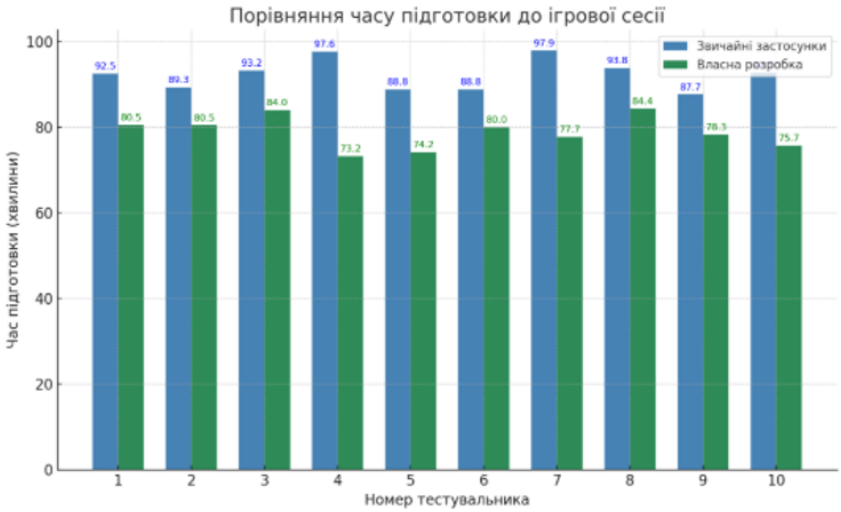

Рисунок Г.14 – Тестування в реальних умовах

Апробація та публікації результатів роботи

АПРОБАЦІЯ

Результати роботи було представлено на Міжнародній науково-практичній інтернет-конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2025) (Вінниця, 2025 р.).

ПУБЛІКАЦІЇ

Байдалюк В. О. Розробка програмних модулів комп'ютерної системи для розвитку логічних здібностей на прикладі настільної гри. Матеріали Міжнародної науково-практичної конференції Молодь в науці: дослідження, проблеми, перспективи. 2025р.

Рисунок Г.15 – Апробація та публікації результатів роботи

Висновки

У процесі виконання кваліфікаційної роботи було здійснено аналітичний огляд актуального стану проблеми, а також проведено аналіз існуючих аналогів програмного забезпечення. Виявлені недоліки сторонніх рішень стали підґрунтям для формування власного бачення системи. Отримані результати дозволили чітко сформулювати основні цілі та завдання проєкту, що визначили подальший напрям розробки. Тестування довело працездатність веб-застосунку та його відповідність поставленим задачам. Практичне застосування розробленого модуля продемонструвало скорочення часу на виконання базових операцій, якістановлять близько 10% робочого процесу, що дозволяє економити до 8% загального часу роботи. Отже, мету бакалаврської кваліфікаційної роботи досягнуто.

Рисунок Г.16 – Висновки

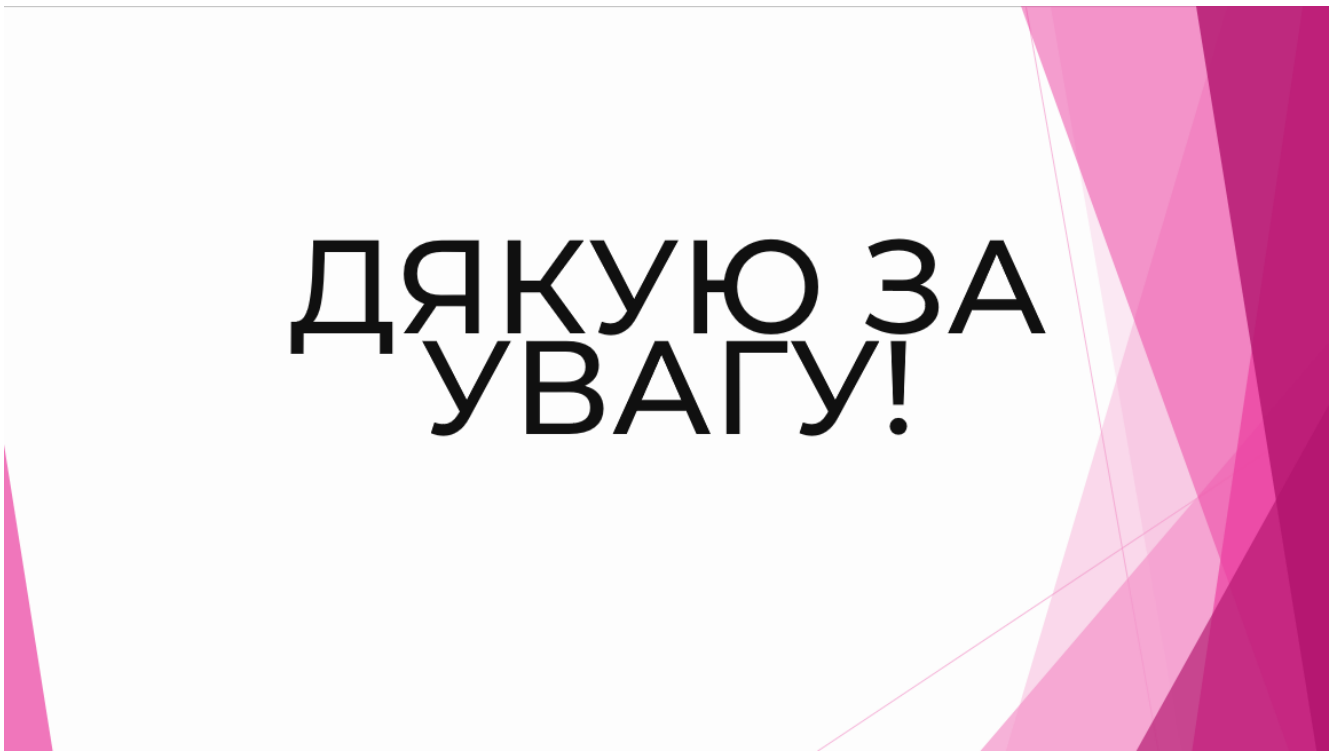

Рисунок Г.17 – Кінцевий слайд