ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Граб Мар'ян Лук'янович

УДК 004.4

Розробка онлайн-платформи для гри в настільну гру "Бункер"

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавр

Нормоконтроль

_____Стисло О.В.

(підпис, дата, розшифрування підпису)

Студент

_____Граб М.Л.

(підпис, дата, розшифрування підпису)

Допускається до захисту

Завідувач кафедри

_____ к.т.н., доц. Ващишак С.П.

(підпис, дата, розшифрування підпису)

Керівник роботи

_____ к.т.н., доц. Ващишак С.П.

(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«____» _____ 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Граб Мар'ян Лук'янович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:

Розробка онлайн-платформи для гри в настільну гру "Бункер"

керівник роботи:

Ващишак Сергій Петрович, к.т.н., доцент

затверджена наказом вищого навчального закладу від «12» березня 2024 року № 19/1

2. Термін подання студентом роботи 05.06.2024
3. Вихідні дані роботи: алгоритми та технології для реалізації гри "Бункер"

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)
   1. Аналіз рішень в області в онлайн платформ по настільній грі Бункер
   2. Постановка задачі та розробка вимог до системи
   3. Розробка та реалізація архітектури системи
   4. Тестування функціональності системи

5. Дата видачі завдання 14.03.2024

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз сервісів та рішень в сфері онлайн платформ по настільній грі Бункер	28.03.2024	Виконано
2.	Постановка задач та розробка вимог до системи	15.04.2024	Виконано
3.	Розробка та реалізація архітектури системи	18.04.2024	Виконано
4.	Тестування функціональності системи	09.05.2024	Виконано
5.	Формування висновків	13.05.2024	Виконано
6.	Оформлення пояснювальної записки	15.05.2024	Виконано
7.	Оформлення графічного матеріалу та підготовка до захисту роботи	20.05.2024	Виконано

Студент _____ Граб М.Л.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ващишак С.П.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

23	Генерація характеристик персонажа	84	Восьма та дев’ята частина файлу room_details.html
74	Перша частина файлу index.html	85	Десята частина файлу room_details.html
75	Друга частина файлу index.html	86	Одинадцята частина файлу room_details.html
76	Файл header.html		

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці онлайн-платформи для гри в настільну гру "The Bunker" з використанням мови програмування Python та фреймворку Django. Онлайн-гри займають особливе місце у сучасному світі інтерактивних розваг, пропонуючи можливість гравцям насолоджуватися ігровим процесом у віртуальному середовищі. Розробка онлайн-платформи для гри в "The Bunker" стала актуальною завдяки постійному зростанню популярності цієї настільної гри та підвищенню попиту на можливість грати в неї в онлайн-режимі.

У вступі роботи обґрунтовується актуальність теми, формулюються мета та завдання дослідження. Робота структурована у відповідності до вимог до кваліфікаційних робіт, де розділ 1 присвячений загальнотеоретичному огляду. У цьому розділі проведено аналіз літератури та існуючих досліджень у галузі, виявлені ключові аспекти та невирішені проблеми, які потребують подальшого дослідження.

Дослідницький процес передбачає розробку онлайн-платформи для гри в "The Bunker" з використанням сучасних методів програмування та розробки ігор. Робота включає в себе такі етапи, як уточнення функціональності платформи, вибір технологій та інструментів розробки, розробка архітектури системи, створення та тестування прототипу платформи, а також впровадження для загального користування.

Отримані результати дослідження мають практичне значення для широкого кола гравців, що мають можливість грати в улюблену гру в будь-який час та в будь-якому місці.

КЛЮЧОВІ СЛОВА: ОНЛАЙН-ПЛАТФОРМА, PYTHON, DJANGO, , ВІРТУАЛЬНЕ СЕРЕДОВИЩЕ, РОЗРОБКА ІГОР, ФУНКЦІОНАЛЬНІСТЬ ПЛАТФОРМИ, ТЕХНОЛОГІЇ РОЗРОБКИ, ВПРОВАДЖЕННЯ, , ОНЛАЙН-РЕЖИМ, ПРАКТИЧНЕ ЗНАЧЕННЯ.

SUMMARY

This thesis is devoted to the development of an online platform for playing the board game "The Bunker" using the Python programming language and the Django framework. Online games occupy a special place in today's world of interactive entertainment, offering players the opportunity to enjoy gameplay in a virtual environment. The development of an online platform for playing "The Bunker" has become relevant due to the constant growth in popularity of this board game and increasing demand for the ability to play it online.

The introduction of the paper substantiates the relevance of the topic, formulates the purpose and tasks of the research. The work is structured in accordance with the requirements for qualification papers, where section 1 is devoted to a general theoretical overview. This chapter analyzes the literature and existing research in the field, identifying key aspects and outstanding issues that require further research.

The research process involves the development of an online platform for playing "The Bunker" using modern methods of programming and game development. The work includes such stages as clarifying the functionality of the platform, choosing technologies and development tools, developing the system architecture, creating and testing a prototype of the platform, as well as implementation for public use.

The obtained research results have practical significance for a wide range of players who have the opportunity to play their favorite game at any time and in any place.

KEYWORDS: ONLINE PLATFORM, PYTHON, DJANGO, VIRTUAL ENVIRONMENT, GAME DEVELOPMENT, PLATFORM FUNCTIONALITY, DEVELOPMENT TECHNOLOGIES, IMPLEMENTATION, ONLINE MODE, PRACTICAL SIGNIFICANCE.

ЗМІСТ

ВСТУП

В сучасному світі розвиток технологій та швидкий розповсюдження Інтернету відкривають нескінченні можливості для розвитку різноманітних інтерактивних розваг. Серед них особливе місце займають онлайн-ігри, які стають не лише джерелом розваг, але і способом відпочинку та спілкування для мільйонів користувачів по всьому світу. Цей феномен став відомим як важливий аспект сучасної культури та суспільства, і вимагає постійного вдосконалення та розвитку ігрових платформ, щоб задовольнити постійно зростаючі потреби гравців в інтернеті.

Однією з таких популярних настільних ігор є "Bunker", яка завойовує серця гравців своєю захоплюючою інтригою та стратегічними елементами. Вона є відмінним засобом розваги та розвитку стратегічних навичок для різних верств населення та вікових груп. Проте, у зв'язку зі зростанням популярності інтернет-ігор, виникає потреба у створенні онлайн-платформи для гри в "Bunker", яка б дозволяла гравцям насолоджуватися цією захоплюючою грою навіть у віртуальному просторі, де вони можуть взаємодіяти незалежно від свого наявного місця перебування.

Актуальність розробки онлайн-платформи для гри в "Bunker" очевидна, оскільки вона відкриває нові можливості для розвитку цієї гри та залучення більшого кола гравців. Вона дозволить збільшити її популярність і розповсюдження, а також розширить можливості її використання як засобу розваги та спілкування.

Мета дослідження: визначення можливостей розробки та впровадження онлайн-платформи для гри в "Bunker" з метою забезпечення нових можливостей для гравців та розширення популярності гри

Завдання дослідження, що сформульовані на основі цієї мети можна визначити як:

1. Визначення функціональних вимог до онлайн-платформи в мережі інтернет для гри в "Bunker".

2. Вибір технологій розробки, що найкраще відповідають вимогам проекту “Bunker”.

3. Проектування архітектури системи, зокрема інтерфейсу користувача та бази даних.

4. Розробка прототипу онлайн-платформи для гри в "Bunker" та його тестування на відповідність визначеним вимогам.

Ці завдання дозволять систематично вирішувати різні аспекти розробки та впровадження онлайн-платформи для гри в "Bunker" з метою досягнення загальної мети дослідження

Об'єкт дослідження: процес розробки та впровадження онлайн-платформи для гри "Bunker" у віртуальному середовищі. Система призначена для того, щоб користувачі мали можливість зіграти в настільну гру Бункер на онлайн платформі, а саме веб-сайті.

Предмет дослідження: розробка нового веб-сайту для користувачів, яка спрямована на створення платформи, де користувачі зможуть зручно грати в настільну гру Бункер, реалізовану в режимі онлайн. Дослідження охоплює аналіз наявних конкурентів та аналіз проблем таких проектів відповідно, огляд можливостей цих проектів, визначення вимог до проекту “Bunker”, розробку прототипу веб-сайту та реалізацію проекту.

Методи дослідження: для того щоб реалізувати проект “Bunker” були використані декілька мов програмування та їх фреймворків. За основу взята мова програмування Python, її фреймворк Django для опису бек-енд частини проекту. Мова гіпертекстової розмітки HTML для розміщення структури побудови сторінок, CSS та Bootstrap для стилізації сторінок, JavaScript для кращого відображення компонентів на сторінці та для процесу запитів на сервер та отримання відповідей від нього, Sqlite3 для роботи з базою даних, щоб зберігати та отримувати дані в ній. Як середовище програмування використовувався

PyCharm. Розроблений проект надає можливість покращувати та модифікувати в майбутнього його функціонал, та варіації процесу гри.

Практичне значення одержаних результатів роботи полягають у розробленій, зручній онлайн-платформі для гри в "Bunker", де користувачі можуть насолоджуватися грою разом з іншими користувачами, навіть перебуваючи у різних місцях світу.

Апробація результатів дослідження. Матеріали кваліфікаційної роботи були представлені на XI Всеукраїнського студентського наукового симпозіуму “СПІВДРУЖНІСТЬ НАУК: архітектура, економіка, право”, яка відбулась 16 листопада 2023 року в Університеті Короля Данила.

Структура роботи: розділи – 3. Загальний обсяг основної частини 87 с. Список використаних джерел містить – 20 позицій.

РОЗДІЛ 1. АНАЛІЗ НАЯВНИХ ПРОГРАМНИХ РІШЕНЬ

Пошук аналогів надав невелику кількість результатів. Є тільки 4 існуючих аналоги ідеї по настільній грі “Бункер”. Перш за все можна знайти сайт “Бункер Онлайн”. Це перший і єдиний існуючий веб-сайт де можна зіграти в настільну гру Бункер онлайн з іншими користувачами. Другим аналогом є мобільний додаток “Shelter”, що доступний на Android та IOS. Окрім цього існує фізична настільна гра Бункер. І останній аналог - телеграм бот “Бункер Гра”, що містить в собі тільки генератор тексту.

Згідно пошуків можна побачити, що не існує задовільних програмних рішень, які матимуть відповідний простий інтерфейс, який зможе задовільнити потреби користувача та бути водночас простим. Окрім цього наявні програмні рішення не можуть запропонувати достатньої інтерактивності та автоматизованості програмного продукту.

Отже як висновок можна зазначити, не існує гідних онлайн аналогів на тему настільної гри Бункер згідно декількох причин. Це пов’язано з тим, що не у кожного з наявних рішень я достатня кількість технічних проблем, які створені вже на частині програмного коду. Окрім цього всі аналоги існують тільки у варіації ворожої локалізації. І в жодному з них не можна змінити мову на будь яку іншу. Отже з аналогів взято лише найкращі риси, що допоможе отримати всі переваги цих програмних рішень і навіть покращити їх. Також після аналізу стали зрозумілі основні недоліки, які були виправлені в процесі виконання кваліфікаційної роботи.

1.1 Аналіз мобільного додатку Shelter

З існуючих аналогів представлений мобільний додаток Shelter. Цей додаток надає можливість користувачам грати в аналогічну настільну гру

"Бункер" шляхом підключення до вже створених кімнат, які заздалегідь існують у системі. Після вибору кімнати гравець може обрати доступне місце в бункері, щоб почати гру.

Основні особливості додатку:

1. Створені кімнати: додаток має вже готові кімнати, до яких можуть підключатися гравці (рис 1.1). Це забезпечує швидкий доступ до гри та спрощує процес організації ігрових сесій. Проте необхідно постійно змінювати кімнати та повідомляти іншим гравцям код для приєднання, оскільки кімнати є статичними і не змінюються.

Рисунок 1.1 – Можливість приєднання до кімнати за допомогою коду

2. Вибір місця в бункері: кожен гравець може обрати доступне місце в бункері заздалегідь визначеної кімнати (рис 1.2).

Це дає можливість уникнути очікування та швидше розпочати гру. Згідно кожного номеру місця розміщені заздалегідь згенеровані картики з характеристиками такими як біологічна інформація, здоров’я, фобія, хоббі, характер, знання, додаткова інформація та багаж. Окрім цього згенеровано

картка дії, яку гравець може використати будь коли. В більшості випадків картки створені для взаємодії з іншими гравця.

Для прикладу гравець може отримати карту дії, що надасть йому можливість обмінятися з іншим гравцем багажем незалежно від того, чи хоче інший гравець цього. Також генерується карта умови, за досягнення якої відбувається якась події по відношенню до гравців, бункера, тощо. Іноді гравець може отримати карту умови, в якій повідомляється, що якщо гравець потрапить в бункер, то він отримає додаткову зброю, що відповідно спростить загалом виживання всіх членів бункеру.

Звичайно, що гравці знаючи цю інформацію можуть маніпулювати іншими грацями, кажучи що у них є у власності така карта умови. А інші гравці не зможуть про це дізнатися до моменту поки не виконається відповідна умова, що прописана в картці. У наведеному раніше прикладі – ця умова, що гравець який має у власності цю карту пройде в бункер.

Рисунок 1.2 – Можливість обрати місце в кімнаті

Заздалегідь згенеровані карти: Після вибору місця в бункері гравець отримує заздалегідь згенеровану карту з характеристиками персонажа та умовами гри.

До переваг у грі можна віднести можливість позначити характеристики які ти вже повідомив іншим гравцям. Це одразу дає тобі розуміння про що вони вже знають і немає сенсу маніпулювати цією інформацією, а про що вони ще не здогадуються. Окрім цього перевагою є наявність в програмі нотатника. В якому ви можете записувати будь яку інформацію, для того, щоб зробити собі нагадування про неї при наступного перегляді картки одного із гравців.

Також перевагою є наявність таймеру, що дозволяє користувачам відслідковувати кількість часу протягом якого гравець може говорити та приводити аргументи чому його слід залишитись в бункері, або чому когось із гравців слід не брати в бункер

Один із найбільших недоліків додатку – використання тільки однієї мови. Це може вплинути на користувацький досвід та зробити додаток менш привабливим для широкого кола користувачів. Оскільки користувачі інших країн не зможуть фізично використовувати цей додаток, тому що, не зможуть отримати мінімальну необхідну інформацію з гри. По причині того, що вся інформація в грі подається в тексті, і відповідно не знаючи мови додатку, це змушує не використовувати його.

Окрім цього як недолік можна виділи те, що кімнати заздалегідь згенеровані та статичні. Тобто користувачі не отримують щоразу нові характеристики та умови, а згідно логіки додатку, вводять код кімнати, та приєднуються до заздалегідь згенерованої кімнати, із картками гравців, що розподілені по місцям. Через це, якщо грати декілька разів в одній кімнаті, то користувачі вже знатимуть у якого місця, які характеристики, що не може гарантувати чесність гри. Також оскільки гра існує у вигляді мобільного додатку, користувачі, що використовують персональні комп’ютери не зможу в неї зіграти, що значно зменшує охоплення гри

Також до недоліків можна виділити нав’язування користувачу платних послуг вже з перших секунд користування додатку. Як тільки користувач відкриває гру він одразу бачить кнопку “Преміум”, що виділяється різними кольорами на відміну від інших, які зображені в більш непомітних кольорах. Після чого при створенні користувач знову бачить що додаткові функції такі як розширений набір карт, та спортивний режим доступні тільки з підпискою преміум. Згідно цього користувач може отримати негативний досвід оскільки одразу розумію що розробники наполягають на тому, щоб він придбав платну підписку на додаток.

Окремо хочеться виділити недолік безпеки та чесності гри в додатку. Оскільки карти заздалегідь згенеровані і кожен з гравців по суті має окрему копію кімнати на своєму пристрої, він легко може натиснути на будь яку характеристику будь якого гравця і таким чином отримати необхідну йому інформацію, що ставить під загрозу цікавість основної механіки гри.

Мобільний додаток Shelter пропонує зручний спосіб грати в настільну гру "Бункер", проте його недоліками є використання однієї мови та невдала ідея із заздалегідь згенерованими кімнатами та картами гравців, що може вплинути на сприйняття та використання додатку користувачами.

1.2 Аналіз сайту Бункер Онлайн

Цей веб-сайт пропонує гравцям можливість грати в настільну гру "Бункер" в реальному часі через інтернет. Гра реалізована у вигляді відкриття характеристик персонажів онлайн та голосування за подальші дії в грі.

Основні переваги даної онлайн-платформи для гри в "Bunker" полягають у наступному: гравці можуть брати участь у грі в реальному часі, що дозволяє їм насолоджуватися процесом гри без затримок; можливість відкриття характеристик персонажів онлайн, що дозволяє гравцям долучитися до ігрового процесу і дізнатися більше про своїх персонажів; можливість голосування за

подальші дії в грі, що робить ігровий процес більш інтерактивним та відкритим для учасників; можливість взаємодії між гравцями та впливу на розвиток подій в грі через голосування та інші механізми; та доступність для учасників гри з будь-якого місця, де є доступ до Інтернету, що робить цей веб-сайт дуже зручним для організації ігрових сесій.

Проте недоліки також присутні. Перш за все – це те, що для гри потрібне стабільне Інтернет-з'єднання, інакше можуть виникати проблеми з підключенням та проведенням гри. Також оскільки гра реалізована через веб-сайт, графічна компонента може бути обмеженою порівняно зі стандартними ігровими додатками. Окрім цього, у сайту не інтуїтивно зрозумілий інтерфейс, що ускладнює залучення більшого кола користувачів до гри (рис 1.3).

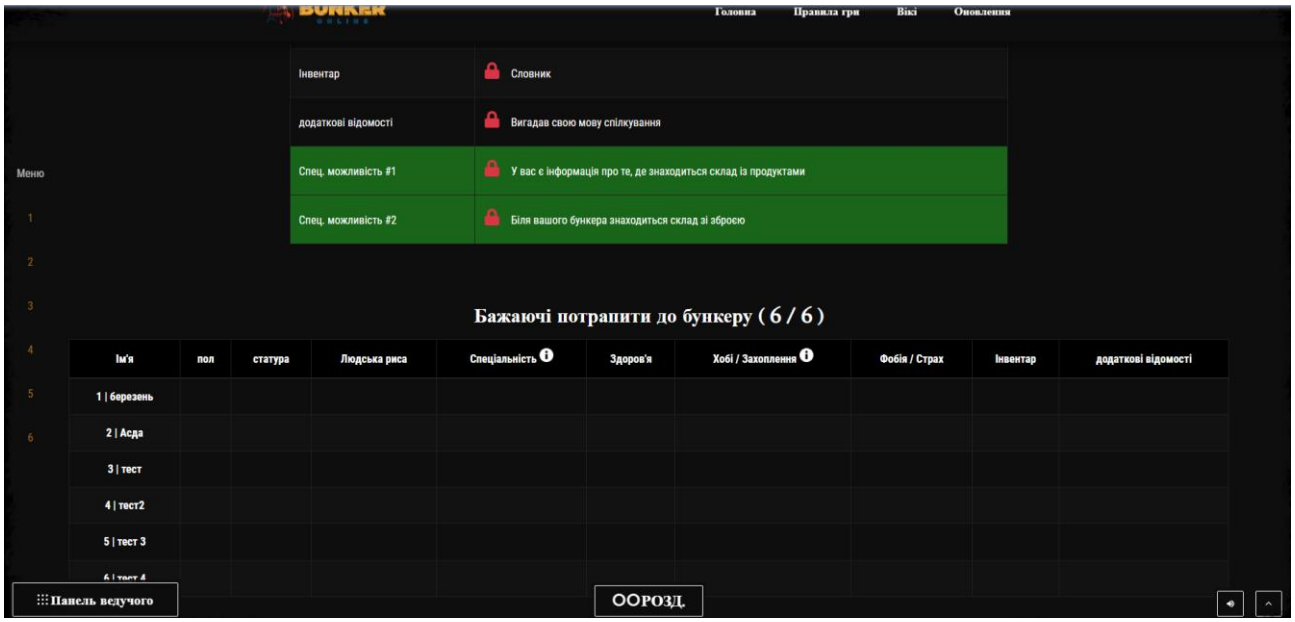


Рисунок 1.3 – Інтерфейс сайту "Бункер Онлайн"

Як недолік також виділяють необхідність контролювати багато процесів гри, власником кімнати. Оскільки деякі важливі дії, такі як голосування, карти дій, зміна професій гравців, закінчення та початок гри, зміна умов бункеру, зміна катаклізму, ініціює власник кімнати, виникає необхідність у більш досвідченому користувачі для цієї ролі. Що відповідно не дає користувачу, який створив

кімнату в повній мірі насолодитися процесом гри, та створює йому додаткові обов'язки. Окрім цього інтерфейс власника кімнати ще більш незрозумілий та складний, що вимагає більшої компетенції від гравця (рис 1.4). Як результат через неправильно влаштовану логіку роботи веб-сайту, у користувачів може відбиватись бажання грати, оскільки потрібно достатньо багато вивчати, для того, щоб просто зіграти в гру.

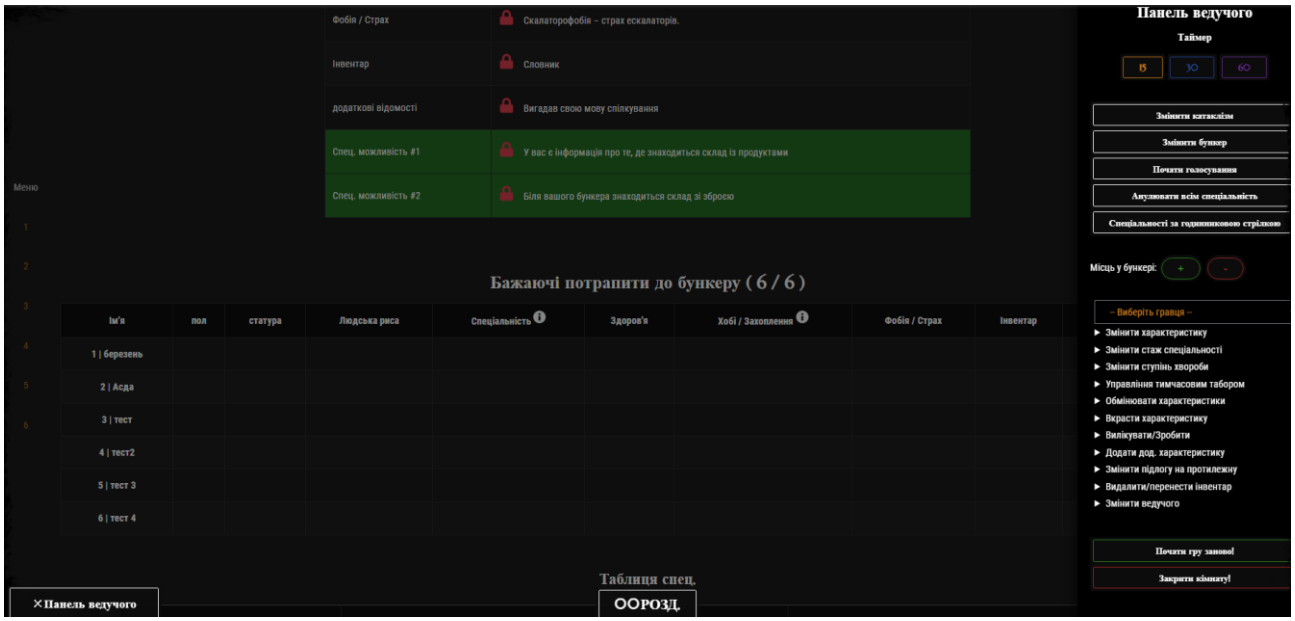


Рисунок 1.4 – Панель керування грою для власника кімнати

Ще один з недоліків необхідність мати щонайменше 6 гравців для початку гри. Тобто сайт вимагає, щоб в сумі було 6 гравців для того, щоб була можливість почати гру. Проте на ньому не існує такого поняття як список з публічними кімнатами куди будь який гравець може доєднатися та зіграти. Як висновок гравцю потрібно самостійно шукати собі користувачів, щоб зіграти. Що очікувано приводить до зменшення залучення користувачів на веб-сайт. Для вирішення цієї проблеми розробники запропонували тільки посилання на їх сервер в програмі Discord, щоб там шукати користувачів для гри. Це рішення не є вдалим оскільки користувачу все одно необхідно знайти на цьому Discord сервері бажаючих зіграти, та вмовити їх доєднатися до тебе.

Враховуючи всю перераховану інформацію, сайт “Бункер Онлайн” є дуже слабким програмним рішенням, що надає лише мінімальний функціонал для користувачів. Проте враховуючи такі недоліки як складний та інтуїтивно не зрозумілий інтерфейс, та погана автоматизація процесу гри. Окрім цього через відсутність можливості комунікувати прямо на сайті, користувачам необхідно доєднатися до Discord серверу і вже там розмовляти з іншими гравцями, що знову ж вимагає від гравців більшої кількості зусиль. Сайт вимагає від користувачів підготовки та знань для того, що користувач міг зіграти. Що очікувано призводить до зменшення кількості гравців, оскільки розбиратися в незручному інтерфейсі та самостійно організовувати процес гри хотітимуть тільки ентузіасти, а не звичайні перехідні гравців, яких більшість.

1.3 Аналіз настільної гри Бункер

Ще один з конкурентів цього проекту є оригінальна настільна гра Бункер. До переваг гри можна одразу віднести візуалізацію, оскільки при купівлі настільної гри користувачі отримують не тільки текстову гру, а ще карточки з тематичним оформленням (рис 1.5). Проте на цьому плюси гри і закінчуються.

Рисунок 1.5 – Приклад фізичних карток настільної гри "Бункер"

Одним і головних недоліків настільної гри Бункер виділяють необхідність придбання її. Оскільки ця гра є фізичною копією, її необхідно придбати у магазинах. Також ще одним важливим недоліком є необхідність фізичної присутності декількох гравців у одному місці для нормальної гри. А ще

враховуючи, що для позитивного досвіду гри рекомендується якнайбільше гравців, це створює необхідність і проблему бути присутнім в одному місці всім гравцям. Якщо ж спробувати зіграти її дистанційно то це зобов’язує одного із гравців займати роль ведучого оскільки потрібно кожному з гравців надіслати текст, або зображення його характеристик. Що як наслідок зменшує загальну кількість користувачів, що будуть приймати активну участь у грі.

Також як недолік виділяють неможливість постійного оновлення та актуалізації функціоналу гри та додаткові варіанти характеристик. Оскільки гра фізична, після придбання у вас залишається тільки стала копія гри з тими самими характеристиками та іншими картами. Як висновок це може призвести до зменшення бажання гравців насолоджуватися грою постійно. Окрім цього з часом фізичні матеріали гри зношуються та пошкоджуються, що призводить до втрати цінності придбаного продукту, та зменшення бажання використовувати гру для задоволення.

Враховуючи представлену інформацію настільна гра "Бункер" має деякі переваги, такі як візуальна привабливість через наявність тематично оформлених карток, які надають грі естетичний аспект.

Однак, не дивлячись на ці плюси, існують значні недоліки, які обмежують її популярність та зручність використання. Серед них можна виділити необхідність придбання фізичної копії гри, що робить її недоступною без витрат для користувачів. Крім того, гравцям необхідно фізично збиратися разом для гри, що ускладнює її організацію.

Неможливість постійного оновлення та актуалізації функціоналу також є важливим недоліком, що може призвести до швидкої втрати інтересу користувачів до гри. У результаті, наявність цих обмежень може значно зменшити бажання гравців користуватися грою, що обмежує її потенціал та ринкові можливості.

1.4 Аналіз телеграм-бота “Бункер Гра”

Один із небагатьох аналогів є телеграм бот “Бункер гра”. Як телеграм бот він пропонує функціонал, який надає можливість згенерувати випадково катастрофу, професія, стать, вік, плодовитість, здоров’я, риса характери, фобія, хобі, додаткова інформація, багаж. Також він випадково генерує дві карти дій, що насправді не є правильним з точки зору основної гри. Оскільки має бути карта дій та карта умов, і відповідно до певної умови, гравці в бункері отримають додаткові бонуси або загрози після вибуття гравця з бункеру.

До переваг можна віднести те, що телеграм бот це доволі просто і оскільки функціональності у нього небагато, інтерфейс користувача закінчується на на десятку кнопок. Згідно цього користувачу легше зорієнтуватися в інтерфейсі.

Окрім цього розробник бота додав свої авторські доповнення до гри. В кінці гри гравці можуть обрати дві опції із доповнень під назвами “Життя в бункері” та “Надія вмирає останньою” де додаються додаткові умови грі. Наприклад після натискання на кнопку життя в бункері гравець отримує випадкову ситуацію, яка пов’язана із життям в бункері, що може додати цікавості грі та збільшити бажання грати у користувачів (рис 1.6).

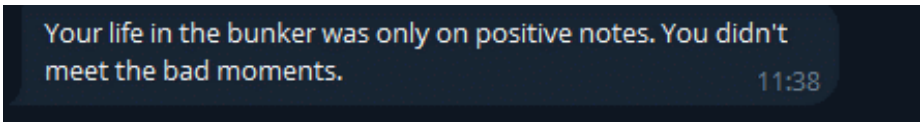


Рисунок 1.6 – Механіка "Життя в бункері"

А ось функція “Надія вмирає останньою” є досить незрозумілою для пересічного користувача. Оскільки в описі немає детального пояснення того, як саме цей етап має відбуватися, користувачі можуть відчувати певну плутанину та невпевненість. Для того щоб функція була зрозумілою та легкою у використанні, необхідно забезпечити повний опис етапів та дій, які повинні

виконуватися. Це дозволить користувачам краще зрозуміти механіку процесу та діяти відповідно до очікувань (рис 1.7).

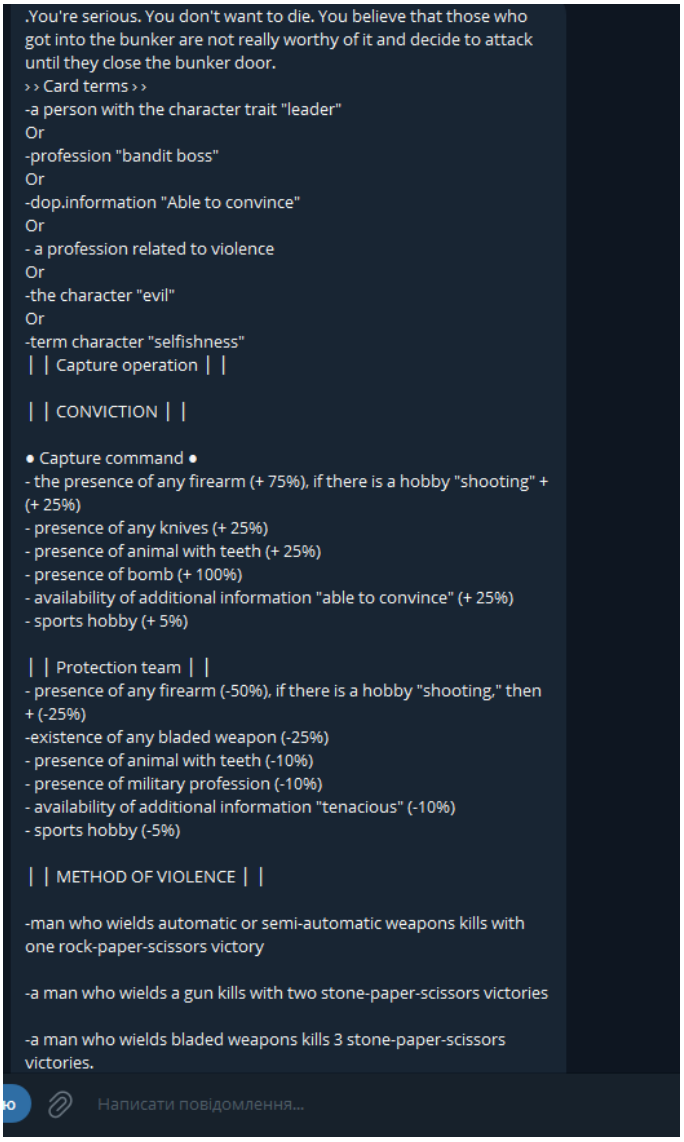


Рисунок 1.7 – Функціонал "Надія вмирає останньою"

Недоліків у аналогу телеграм бота “Бункер Гра” досить багато. Перш за все виділяється невеликий функціонал, що обмежується тільки генерацією характеристик. Оскільки вони генеруються кожна окремо користувачу потрібно витратити багато часу для того, щоб згенерувати собі повну картку (рис 1.8).

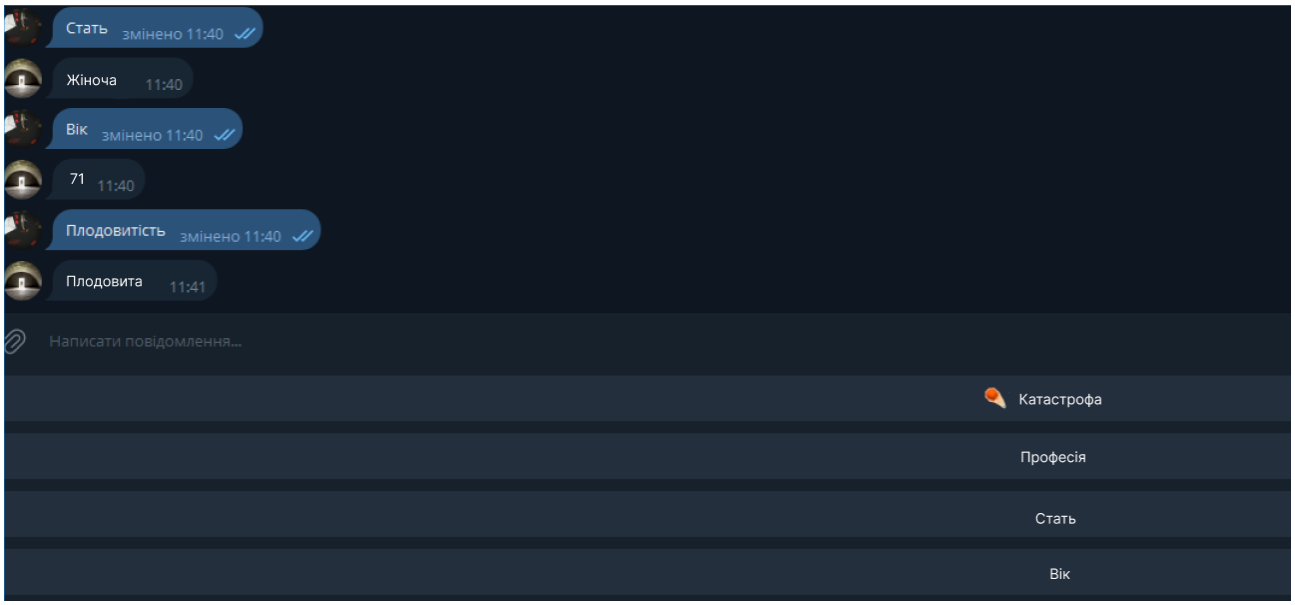


Рисунок 1.8 – Генерація характеристик персонажа

Окрім цього користувач отримує кожну характеристику окремо як наслідок читати ці характеристики зовсім не зручно, оскільки потрібно постійну шукати потрібне повідомлення серед інших.

Також оскільки інтерфейс взаємодії користувача сильно спрощений, гравець не отримує ніякої візуальної атмосфери як наприклад з фізичною версією гри. Ще одним недоліком є відсутність будь якої комунікації з іншими гравцями за допомогою телеграм боту. Тобто використовуючи телеграм бот користувачі мають бути присутніми фізично один біля одного, або бути на зв’язку в голосових та текстових каналах.

Вагомим недоліком є відсутність будь-якої візуалізації характеристик інших можливих членів бункеру. Тобто гравець змушений десь собі записувати характеристики інших гравців окремо. Оскільки телеграм бот не дозволяє такого зробити прямо в ньому.

Окрім цього в телеграм боті не реалізовано механіку з голосування за гравців та відповідно немає можливості вигнати гравця за допомогою голосування. І оскільки ця механіка є основною в ідеї настільної гри “Бункер”,

то користувачі зовсім не отримують необхідного функціоналу від телеграм боту “Бункер Гра”.

Останнім та не менш важливим недоліком – є відсутність можливості знайти користувачів для гри. Немає жодного каналу або посилання на чат де перебувають користувачі, що зацікавлені в грі.

Після проаналізованої інформації можна з впевненістю сказати, що телеграм бот "Бункер Гра" є одним із небагатьох аналогів настільної гри "Бункер", але має кілька суттєвих недоліків. Перевагами є простий інтерфейс користувача та наявність додаткових авторських доповнень до гри, які розширюють можливості гравців. Однак недоліки також досить значні. До них належить обмежений функціонал, який ускладнює процес гри та вимагає великої кількості часу для отримання повноцінної картки характеристик. Крім того, відсутність візуалізації та можливості взаємодії з іншими гравцями позбавляє гру від атмосфери та можливості спілкування. Не реалізовані ключові механіки, такі як голосування та викидання гравців, додатково ускладнюють процес гри. В цілому, хоча телеграм бот "Бункер Гра" може бути цікавим доповненням до настільної гри, він не забезпечує повноцінного геймплею та не відображає усі аспекти та можливості оригінальної гри.

1.5. Постановка задачі

Список задач, які враховуватимуть всі недоліки попередньо оглянутих аналогів, та їх переваг відповідно:

1. Автоматизований процес гри передбачає автоматизацію основних процесів гри, щоб користувачам було простіше отримувати задоволення від гри. Це включає зручний доступ до списку кімнат, де відбуваються ігри, та можливість легко доєднатися до них.

2. Генерування унікальних умов для гри важливе для створення цікавих ігрових ситуацій. Це включає генерацію унікальних карт з характеристиками

персонажів, катастроф під час початку нової гри та кінця гри в залежності від катастрофи, яка надається на початку гри.

3. Забезпечення чесності гри є важливою складовою гри. Це означає, що кожен гравець може відкривати тільки свої характеристики, а також має можливість голосувати тільки один раз під час певного ходу.

4. Зручність початку гри для користувача полягає в створенні максимально простого процесу початку гри. Це може включати швидкий та зрозумілий процес реєстрації, простий доступ до списку доступних ігор та можливість швидко приєднатися до гри.

Для того щоб користувачі отримали максимально кількість позитивних емоцій від користування продуктом потрібно зроби весь функціонал простим для використання та розуміння.

Перш за все потрібно зробити інтуїтивно-зрозумілий інтерфейс, щоб користувачі одразу могли зрозуміти як швидко почати грати, оскільки це дуже важливо для утримання нових гравців на продукті.

Висновок до розділу 1

На основі аналізу в розділі 1, можна вирішити, що розробка онлайн-платформи для гри в настільну гру "Бункер" є доцільною, оскільки існуючі конкуренти мають свої недоліки, які можна вирішити через розробку нового рішення. Така платформа може надати зручний спосіб грати в гру, збільшити доступність та інтерактивність гри, а також забезпечити постійне оновлення та розвиток ігрового процесу.

РОЗДІЛ 2. ОБГРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ

2.1 Загальна характеристика проблеми

Проблема з обмеженістю існуючих рішень для гри в настільну гру "Бункер" виникає через низку факторів, що впливають на користувальницький досвід та доступність гри. Початковою причиною є мовні бар'єри, які зустрічаються у деяких ігрових додатках, таких як мобільний додаток Shelter, який використовує лише російську мову. Це може викликати несприятливі умови для гравців, які не володіють цією мовою, обмежуючи їх можливість насолоджуватися грою та взаємодіяти з нею.

Додаток Shelter також має інші недоліки, такі як статичність кімнат і карт, що може призвести до зменшення цікавості гри з плином часу. Крім того, використання готових кімнат і обмеженість у розширенні функціоналу гри можуть обмежувати креативність та можливості гравців.

З іншого боку, веб-сайт Бункер Онлайн пропонує інтерактивний досвід гри в реальному часі, але також має свої недоліки, такі як необхідність стабільного Інтернет-з'єднання для участі у грі та складний інтерфейс, що ускладнює процес гри для користувачів.

У зв'язку з цим, розробка власної онлайн-платформи для гри в настільну гру "Бункер" з використанням мови програмування Python та фреймворку Django є доцільним кроком.

2.2 Предметні риси, що впливають на розвиток проблеми

У розвитку онлайн-платформи для гри в настільну гру "Бункер" ключову роль відіграють предметні риси, що впливають на специфіку самої гри та

потреби гравців. Однією з найважливіших предметних рис є потреба у створенні платформи, яка була б доступною та зручною для гравців у будь-який час і з будь-якого місця. Це вимагає використання технологій, які забезпечують можливість грати в гру через Інтернет, без обмежень щодо географічного розташування гравців.

Ще однією ключовою предметною рисою є потреба спростити процес організації гри та взаємодії між гравцями. Це означає розробку зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам легко створювати кімнати для гри, приєднуватися до них та взаємодіяти під час гри без зайвих перешкод або складнощів.

Покращення інтерактивності гри та зроблення її більш захопливою та цікавою для гравців також є важливою предметною рисою. Це включає в себе використання технологій, які забезпечують можливість взаємодії між гравцями, таку як голосування за подальший розвиток подій у грі або спільне прийняття рішень. Також важливо створити естетичний та привабливий дизайн, який поглибить занурення гравців у гру та стимулюватиме їхню активність.

Таким чином, розробка онлайн-платформи для гри в настільну гру "Бункер" потребує уважного врахування та відповідного відповідного реагування на ці предметні риси, щоб забезпечити найкращий досвід для гравців та зробити гру привабливою та доступною для широкої аудиторії.

2.3 Проектування програмного продукту

Для проектування загального функціоналу програмного продукту був використаний функціонал Use case UML діаграми,яку можна переглянути в додатку И. В якій був спроектований основний запланований функціонал, для реалізації програмного продукту. В ньому описані основні актори, та дії, які їм доступні. Окрім цього описані можливості власника кімнати: створити кімнату, змінити налаштування гри, почати гру, писати в чат, отримати карти з

характеристиками персонажа, показати характеристики інших гравцям, використання карт дій, та голосування за гравців для того, щоб вони залишили ігрову сесію.

Для гравців описаний такі основні доступні дії: Доєднатися до вже створеної кімнати, шукати кімнату по назві, доєднатися до кімнати за допомогою посилання, або за допомогою списку кімнат. Також у користувача є можливість надіслати скаргу на іншого гравця якщо він порушує правила гри, порушує правила спілкування, або перешкоджає процесу гри. Окрім цього звичайно звичайний гравець має можливість показати свої характеристики іншим гравцям, використати карту дії, та проголосувати за те, щоб інший гравець покинув гру.

Після того, як залишиться необхідна кількість користувачів для того, щоб гра завершилася, гравці отримують повідомлення із результатом гри. Де в залежності від того, чи є у них необхідне спорядження або професії вони або виживають, або отримають погане закінчення гри.

Використання Use Case UML-діаграми може бути дуже корисним для проектування розробки онлайн-платформи для гри в настільну гру "Бункер" з наступних причин:

1. Визначення функціональності системи: Use Case діаграми дозволяють чітко визначити, які функції має виконувати система. Для гри "Бункер" це може включати створення кімнат для гри, вибір персонажів, взаємодію з іншими гравцями тощо.

2. Ідентифікація акторів: Use Case діаграми допомагають ідентифікувати всіх основних акторів, які взаємодіють з системою. Для випадку онлайн-платформи для гри в настільну гру "Бункер” це гравці та власники кімнат(в яких проходитимуть ігри).

3. Уточнення потреб користувачів: Ці діаграми дозволяють збільшити розуміння потреб користувачів і визначити їх очікування від системи. Наприклад, використання Use Case може допомогти з'ясувати, які функції гри важливі для гравців, які аспекти їм цікаві, які можливості вони хочуть мати тощо.

4. Запобігання непорозумінням: Use Case діаграми дозволяють сторонам проектування і розробки зрозуміти, як система повинна працювати, та уникнути непорозумінь щодо її функціональності.

5. Орієнтація на користувача: Використання Use Case діаграм допомагає зорієнтувати розробників на потреби користувачів, що сприяє створенню більш користувацько-орієнтованої системи.

Отже, використання Use Case UML-діаграми для проектування розробки онлайн-платформи для гри в настільну гру "Бункер" допоможе чітко визначити функціональні вимоги системи, ідентифікувати акторів, уточнити потреби користувачів, уникнути непорозумінь та забезпечити орієнтацію на користувача.

2.4 Вибір мови та технологій програмування

Вибір мови програмування та технологій для розроблення програмного продукту є важливим етапом, оскільки від цього залежить якість, швидкість розробки, продуктивність та майбутні можливості продукту. Далі були описані причини згідно з якими були обрані мови програмування та технології для розробки проекту.

2.4.1 Причини використання Python

Python - це високорівнева та інтерпретована мова програмування, яка набула широкої популярності завдяки своїй простоті, читабельності та ефективності . Вона була розроблена Гвідо ван Россумом і перший раз випущена у 1991 році.

Однією з основних особливостей Python є його простота використання. У нього є обширна та зрозуміла документація [14]. Мова має чистий і зрозумілий синтаксис, що робить її ідеальним вибором для початківців у програмуванні.

Python пропонує простий спосіб вирішення складних завдань, не потребуючи великої кількості коду.

Ще однією важливою рисою Python є його читабельність. Код на Python легко читати і зрозуміти, що сприяє збереженню і підтримці великих проектів. Це дозволяє командам програмістів працювати ефективно, спільно працюючи над проектами.

Python також відомий своєю масивною екосистемою бібліотек і фреймворків, які дозволяють розробникам виконувати різноманітні завдання. Наприклад, Django та Flask використовуються для розробки веб-додатків, TensorFlow і PyTorch - для розробки і штучного інтелекту та машинного навчання, а Matplotlib і Seaborn - для візуалізації даних. Саме Django і був використаний для реалізації проекту.

Крім того, Python підтримує крос-платформенність, що означає, що код, написаний на Python, може працювати на будь-якій платформі без змін. Це робить його універсальним інструментом для розробки програмного забезпечення на різних платформах.

2.4.2 Причини використання Django

Django - це високорівневий фреймворк для розробки веб-додатків на мові програмування Python, який надає розробникам зручні інструменти для швидкої і ефективної розробки веб-застосунків.

Однією з головних переваг Django є його вбудовані функції, які значно спрощують розробку веб-додатків. Окрім цього за допомогою простої документації, легко розібратися як його використовувати [2]. Наприклад, Django має вбудовану адміністративну панель, яка дозволяє легко створювати, оновлювати та видаляти дані з бази даних без необхідності написання власного адміністративного інтерфейсу. Також Django має вбудовану систему

аутентифікації, яка дозволяє легко налаштовувати авторизацію та ідентифікацію користувачів у вашому додатку.

Ще однією важливою особливістю Django є його система маршрутизації URL-адрес, яка дозволяє з легкістю визначати, які дії виконувати при отриманні певного URL-адресу. Це спрощує організацію маршрутів у вашому додатку та робить його структуру більш зрозумілою та організованою.

Додатково, Django має потужну систему шаблонів, яка дозволяє вам використовувати шаблони HTML для відображення динамічного контенту у вашому веб-додатку. Це робить розробку фронтенду більш зручною та організованою для розробника.

Нарешті, Django має високий рівень безпеки, що дозволяє розробникам будувати безпечні веб-додатки з мінімальними загрозами для безпеки. Фреймворк має вбудовані заходи безпеки, такі як захист від Cross-Site Scripting (XSS), захист від SQL-ін'єкцій та інші, які допомагають запобігати потенційним загрозам безпеки для вашого додатку.

Узагальнюючи, Django - це потужний і простий у використанні фреймворк, який дозволяє розробникам швидко та ефективно створювати безпечні веб-додатки на Python. Його вбудовані функції та висока безпека роблять його ідеальним вибором для розробки веб-додатків будь-якого розміру і складності.

2.4.3 Причини використання HTML

HTML (HyperText Markup Language), CSS (Cascading Style Sheets) і JavaScript є основними мовами для розробки фронтенду веб-додатків, які використовуються для створення інтерфейсу користувача, його оформлення та забезпечення взаємодії з користувачем.

Також HTML є стандартною мовою розмітки, яка використовується для створення структури веб-сторінок. Вона визначає різні елементи сторінки, такі як заголовки, абзаци, списки, посилання, зображення та інші. HTML дозволяє

організувати інформацію на сторінці та встановлювати її структуру. Також вона має дуже зрозуміло та зручну документацію [6].

Окрім цього вона є мовою розмітки, що використовується для створення веб-сторінок. Вона дозволяє розміщувати різноманітний контент, такий як текст, зображення, відео та інші елементи, на веб-сторінках. HTML допомагає організувати цей контент і визначати його структуру, що робить веб-сторінки зрозумілими для браузерів та користувачів.

Основні концепції HTML включають:

1. Структура сторінки: HTML визначає структуру веб-сторінки, включаючи заголовки, підзаголовки, абзаци, списки та інші елементи.
2. Семантика: HTML дозволяє використовувати семантичні теги для позначення різних частин сторінки, що допомагає пошуковим системам та іншим інструментам краще розуміти контент.
3. Вкладеність: HTML дозволяє вкладати один елемент в інший, що дозволяє створювати складніші структури та розміщувати контент у відповідних блоках сторінки.
4. Форми: HTML має спеціальні елементи для створення форм, які дозволяють користувачам взаємодіяти з веб-сторінками, наприклад, для відправки даних на сервер або для введення користувачем інформації.
5. Мультимедіа: HTML дозволяє вставляти мультимедійний контент, такий як зображення, відео та аудіо, на веб-сторінки, що робить їх більш привабливими та інтерактивними.

Тож HTML є важливою складовою веб-розробки і є основою для будь-якого веб-сайту чи веб-додатка.

2.4.4 Причини використання CSS

CSS відповідає за візуальне оформлення веб-сторінок, таке як кольори, шрифти, розміри та розташування елементів. Вона дозволяє розміщувати стилі

на сторінці, що забезпечує її привабливість та користувацьку доступність. CSS також дозволяє створювати анімацію та ефекти для покращення користувацького досвіду.

CSS (Cascading Style Sheets) - це мова стилів, яка використовується для визначення вигляду та оформлення веб-сторінок. Вона дозволяє розміщувати стилі на веб-сторінці, щоб визначити різні аспекти візуального вигляду, такі як кольори, шрифти, розміри, відступи, рамки, фонові зображення та багато іншого.

Основні концепції CSS включають:

1. Селектори: CSS використовує селектори для вибору елементів на веб-сторінці, до яких будуть застосовані стилі. Це може бути конкретний елемент HTML, клас, ідентифікатор або навіть псевдоклас або псевдоелемент.

2. Властивості та значення: Кожна CSS-властивість має значення, яке визначає, як буде відображатися вибраний елемент. Наприклад, властивість "color" визначає колір тексту, а "font-size" - розмір шрифту.

3. Каскадність і спадкування: Каскадність в CSS визначає порядок застосування стилів до елементів, що дозволяє використовувати кілька різних наборів стилів для одного елементу. Спадкування дозволяє властивостям батьківського елемента передаватися дочірнім елементам, що спрощує оформлення сторінки.

4. Боксова модель: CSS використовує боксову модель для визначення розмірів та розташування блочних елементів на сторінці. Кожен елемент має внутрішній контент, відступи, рамку та поле, які можна керувати за допомогою мови CSS.

5. Адаптивність і резиновий дизайн: CSS дозволяє створювати адаптивні та гнучкі дизайни, що забезпечує підтримку різних пристроїв та розмірів екрану.

6. Анімація і переходи: CSS дозволяє створювати анімацію та переходи для створення рухомих та інтерактивних ефектів на веб-сторінці.

CSS є важливою складовою веб-розробки, оскільки вона дозволяє розміщувати стилі на веб-сторінці, що забезпечує привабливий та доступний для користувача вигляд веб-сторінки.

2.4.5 Причини використання JavaScript

JavaScript - це мова програмування, яка широко використовується для розробки веб-додатків і надає можливість створення інтерактивної та динамічної веб-сторінки. Вона забезпечує можливість взаємодії з користувачем без перезавантаження сторінки, що робить веб-додатки більш зручними та ефективними для користування [17].

JavaScript використовується для різноманітних завдань, таких як валідація введених даних у форми, анімація елементів сторінки, динамічне створення та зміна вмісту сторінки, робота з кукісами та локальним сховищем, а також взаємодія з сервером за допомогою AJAX-запитів.

Однією з ключових особливостей JavaScript є його можливість реагувати на події, такі як натискання кнопок, переміщення курсора миші, а також введення тексту користувачем. Це дозволяє створювати інтерактивні елементи інтерфейсу, які реагують на дії користувача миттєво та змінюються відповідно до контексту.

JavaScript має велику кількість різноманітних бібліотек і фреймворків, таких як React, Angular, Vue.js, які спрощують розробку веб-додатків і надають додаткові можливості для роботи з інтерфейсом та даними.

В цілому, JavaScript є незамінним інструментом для розробки сучасних веб-додатків і забезпечує широкі можливості для створення динамічних та інтерактивних веб-сторінок. Від розширює можливості при створенні проекту. Та допомагає зробити його якісним.

2.4.6 Причини використання Bootstrap

Bootstrap - це інструмент, який значно спрощує процес розробки веб-інтерфейсів, забезпечуючи розробникам готові компоненти та стилі для швидкої і ефективної роботи. Він містить широкий спектр елементів, таких як кнопки, форми, меню, картки, таблиці, навігаційні панелі та інші, що дозволяє створювати стильні та сучасні веб-інтерфейси без необхідності писати велику кількість коду з нуля.

Однією з ключових переваг Bootstrap є його адаптивність. Це означає, що веб-додатки, розроблені з використанням цього фреймворку, оптимізовані для різних пристроїв та розмірів екранів. Незалежно від того, чи відкривається додаток на комп'ютері, планшеті або смартфоні, він буде відображатися коректно та зручно для користувача.

Ще однією важливою перевагою Bootstrap є його простота використання. Фреймворк має добре документовану інструкцію та багато прикладів, які допомагають розробникам швидко засвоїти його функціонал та почати працювати над проектами. Це особливо корисно для початківців, які тільки вчаться робити веб-розробку.

Крім того, Bootstrap постійно оновлюється та підтримується великою спільнотою розробників. Це означає, що він завжди актуальний і відповідає сучасним вимогам ринку веб-розробки. Багато проектів використовують Bootstrap, що сприяє його поширенню та популярності серед розробників.

У підсумку, Bootstrap є потужним інструментом для швидкої та ефективної розробки адаптивних та стильних веб-інтерфейсів. Його готові компоненти, адаптивність та простота використання роблять його незамінним для багатьох веб-розробників у всьому світі.

2.4.7 Причини використання SQLite3

Розширення наявного технологічного стеку базою даних SQLite3 виявляється стратегічним рішенням з декількох причин. По-перше, SQLite3 славиться своєю легкістю використання, вона вбудована та не потребує окремого сервера або складної конфігурації [18]. Це особливо підходить для проекту, оскільки було створено середнього розміру веб-додаток із базовими потребами у зберіганні даних.

Друга причина - швидкість. SQLite3 працює ефективно, що робить його ідеальним вибором для веб-додатків з помірним обсягом даних. Він не потребує окремого процесу сервера, що дозволяє зменшити час доступу до даних і підвищити загальну продуктивність.

Третя перевага - портативність. SQLite3 база даних може бути легко перенесена між різними платформами, що робить її ідеальною для розробки крос-платформених веб-додатків, які можуть працювати на різних пристроях та операційних системах.

Крім того, SQLite3 легко інтегрується з фреймворком Django, який ми обрали для розробки нашого веб-додатку. Django має вбудовану підтримку для SQLite3, що робить його легким у використанні та інтеграції, і дозволяє швидко створити зв'язок з базою даних та працювати з даними за допомогою Django ORM.

Не остання, але дуже важлива перевага полягає у відсутності необхідності в складних конфігураціях. SQLite3 не потребує складних налаштувань чи установки, що полегшує розгортання та управління базою даних.

Отже, використання SQLite3 в проекті є логічним та обґрунтованим вибором, оскільки вона надає нам швидкість, простоту використання, інтеграцію з Django та портативність, що сприятиме успішному розвитку та ефективній роботі нашого веб-додатку.

Підтримка спільноти та документація: Python, Django та зазначені вище технології мають велику активну спільноту розробників, що забезпечує швидке вирішення проблем та підтримку у випадку потреби. Крім того, для цих технологій існує обширна та доступна документація, яка допоможе розробникам у розв'язанні будь-яких технічних питань.

2.4.8 Причини вибору PyCharm, як інтегрованого середовища

PyCharm - це інтегроване середовище розробки (IDE) для мови програмування Python. Воно розроблене компанією JetBrains, яка спеціалізується на створенні програмних продуктів для розробки програмного забезпечення. PyCharm надає розробникам зручні та потужні інструменти для написання, відлагодження та тестування програм на Python.

Створення PyCharm було спрямоване на забезпечення продуктивної роботи розробників Python, шляхом надання їм інтегрованого середовища з широким набором функцій і інструментів. Інтерфейс PyCharm є добре організованим і інтуїтивно зрозумілим, що дозволяє швидко орієнтуватися в ньому та зосередитися на розробці програмного забезпечення.

Підтримка віртуальних середовищ, автоматичне завершення коду, вбудовані інструменти аналізу та відлагодження, підтримка різних фреймворків і бібліотек, інтеграція з системами керування версіями - лише кілька з функцій, які роблять PyCharm потужним інструментом для розробки на Python.

1. Комплексність та функціональність: PyCharm - це повнофункціональне інтегроване середовище розробки, яке містить в собі все необхідне для зручної роботи з Python. Воно об'єднує в собі редактор коду, вбудовану систему керування версіями, відлагоджування, рефакторинг, підтримку віртуальних середовищ та багато іншого.

2. Інтуїтивний інтерфейс користувача: Інтерфейс PyCharm є добре організованим і інтуїтивно зрозумілим. Різноманітні функції та інструменти

легко доступні через різні панелі та меню, що робить робочий процес розробки більш продуктивним.

3. Підтримка різних інструментів та технологій: PyCharm підтримує роботу з різними версіями Python, включаючи Python 2 та Python 3, а також інтегрується з різними фреймворками та бібліотеками, що використовуються в розробці Python.

4. Вбудовані інструменти аналізу коду та автоматичної перевірки стилю: PyCharm має вбудовані інструменти аналізу коду, які допомагають виявляти помилки, оптимізувати код та вдосконалювати його стиль. Це допомагає підтримувати високу якість коду та полегшує процес розробки.

5. Широкі можливості налаштування: PyCharm надає широкі можливості налаштування, що дозволяє налаштовувати середовище роботи під власні потреби та вимоги проекту.

6. Активна спільнота користувачів та підтримка: PyCharm має велику активну спільноту користувачів, що дозволяє швидко знаходити відповіді на питання та вирішувати проблеми, а також отримувати оновлення та підтримку від розробників.

7. Інтеграція з іншими інструментами розробки: PyCharm може легко інтегруватися з іншими інструментами розробки, такими як системи керування версіями, сервери баз даних, віртуальні середовища та інші, що робить його універсальним інструментом для розробників Python

Вибір PyCharm як інтегрованого середовища розробки для програмування на мові Python обгрунтований його численними перевагами. Розроблене компанією JetBrains, PyCharm надає розробникам потужні та зручні інструменти для написання, відлагодження та тестування програм.

Комплексність та функціональність PyCharm забезпечують повноцінну роботу з Python, включаючи редагування коду, відлагодження, рефакторинг та керування версіями. Інтуїтивний інтерфейс користувача робить робочий процес більш продуктивним та ефективним.

Узагальнюючи, вибір PyCharm є обґрунтованим з точки зору зручності, функціональності та підтримки, що робить його ідеальним інструментом для розробки програмного забезпечення на мові Python

Висновок до розділу 2

У результаті обґрунтування вибору технологій програмування можна зробити висновок, що використання Python, PyCharm, HTML, CSS, Bootstrap, JavaScript та SQLite3 є оптимальним варіантом для реалізації поставлених завдань у рамках даного проекту. Вони забезпечують потрібний рівень продуктивності, зручність у розробці та підтримку, що є ключовими для успішної реалізації програмного продукту.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ МАТЕРІАЛ ДОСЛІЖЕННЯ

3.1 Модель для розробки проекту

Для реалізації кваліфікаційної роботи вибрано Python та його фреймворк Django. Окрім цього згідно розділу 2, для реалізації візуальної частини проекту використовувалисб JavaScipt, HTML, CSS, Bootstrap. Django використовує модель розробки, відому як "Model-View-Controller" (MVC), або у випадку Django, можна сказати, "Model-View-Template" (MVT) [12]. Ця модель розділяє різні аспекти веб-розробки на три компоненти:

1. Модель (Model): Це представлення даних, що використовуються в додатку. Моделі Django визначають структуру даних і логіку доступу до них. Вони зазвичай відображаються в базі даних.

2. Представлення (View): Це компонент, який відповідає за відображення даних на сторінці. У Django представленням може бути функція або метод класу, яка приймає запит та повертає відповідь.

3. Шаблон (Template): Шаблони Django використовуються для відображення HTML-сторінок з використанням даних, отриманих з представлення. Вони дозволяють вставляти дані та логіку в HTML-код.

Ця архітектурна модель робить код більш організованим, що дозволило розділити логіку додатку на логічно пов'язані компоненти.

3.2 Структура проекту

Папка Bunker є основним пакетом проекту на Django і містить основний код з налаштуваннями проекту, та маршрутами. У ній зберігаються такі фалйи:

– _init_.py: файл містить код ініціалізації для пакета Bunker;

– asgi.py: налаштовує асинхронний серверний інтерфейс WSGI (ASGI) для роботи проекту;

– settings.py: містить всі налаштування проекту Django, такі як з'єднання з базою даних, шаблони та маршрутизація URL;

– urls.py: визначає маршрутизацію URL для проекту;

– wsgi.py: налаштовує сервер WSGI для проекту.

У папці myapp зберігається основний код веб-сайту, де було прописано весь основний функціонал. Він описаний в таких файлах:

– migrations: папка містить файли міграції для моделей Django;

– _init_.py: містить код ініціалізації для пакета myapp;

– admin.py: налаштовує адміністративний інтерфейс Django для моделей;

– apps.py: реєструє ваш пакет myapp у проекті Django;

– forms.py: містить форми Django для проекту;

– models.py: містить моделі Django для проекту;

– tests.py: містить тестові файли для коду;

– utils.py: містить утилітні функції для коду;

– views.py: містить представлення Django для проекту.

Папка templates зберігає в собі шаблони з Fron-end частиною проекту. В більшості у цій папці зберігаються файли з HTML,CSS та JavaScipt кодом. В ній зберігаються такі файли та папки:

– registration: містить шаблони для реєстрації користувачів;

– character_info.html: використовується для відображення форми створення персонажа;

– header.html: використовується для відображення заголовка сторінки;

– index.html: використовується для відображення головної сторінки сайту;

– room_detail.html: використовується для відображення правильної інформації про кімнату;

– db.sqlite3: файл бази даних SQLite, який використовується проектом.

manage.py: використовується для керування проектом Django, наприклад, для створення міграцій та запуску сервера розробки.

Проект має два основні компоненти: "Bunker" і "myapp". Папка "Bunker" є основним пакетом проекту, який містить код, спільний для всіх додатків. Це включає різноманітні утиліти, загальні налаштування, конфігураційні файли та базові класи або функції, що використовуються в різних частинах проекту. Цей пакет служить ядром, на якому будуються інші частини проекту, забезпечуючи єдине місце для загального функціоналу та дозволяючи уникнути більшості дублювання коду.

Папка "myapp" — це додаток, що містить код для веб-сайту. У ньому знаходяться моделі, представлення, шаблони та інші файли, необхідні для роботи веб-сайту. Моделі визначають структуру даних та взаємодію з базою даних, представлення відповідають за обробку запитів користувачів і повернення відповідних відповідей, а шаблони визначають вигляд веб-сторінок, які будуть відображені користувачам. Інші файли можуть включати статичні ресурси (зображення, CSS, JavaScript), файли конфігурації та тести для забезпечення якості коду.

Шаблони використовуються для генерації HTML-коду для веб-сайту. Вони містять HTML-код, а також спеціальні теги Django, які дозволяють вставляти динамічні дані.

Моделі використовуються для представлення даних у вашій базі даних. Вони визначають структуру таблиць та полів у базі даних.

Представлення використовуються для обробки запитів HTTP та генерування відповідей. Вони повертають HTML-код, JSON або інші типи даних.

База даних використовується для зберігання даних проекту, які пізніше використовувалися для відображення інформації користувачу.

Маршрутизація URL використовується для зіставлення URL-адрес з представленнями. Коли користувач вводить URL-адресу у своєму браузері,

Django використовує маршрутизацію URL, щоб знайти відповідне представлення для обробки запиту.

3.3 Розробка функціоналу проекту

3.3.1 Моделі

Основна частина розробки проекту знаходиться у декількох основних файлах. Перш за все це файл models.py, views.py, urls.py та папка templates з основними html файлами, що відповідають за відображення інформації для користувачів на сторінці. У файлі models.py зберігається код моделей які існують в проекті. Цільний файл з усім кодом можна переглянути в додатку А.

```
from django.db import models
from django.contrib.auth.models import User
from django.conf import settings
from .utils import BIO_OPTIONS, HEALTH_OPTIONS, PHOBIA_OPTIONS,
HOBBY_OPTIONS, KNOWLEDGE_OPTIONS, \
    ADDITIONAL_INFO_OPTIONS, LUGGAGE_OPTIONS
from django.db.models.signals import pre_delete
from django.dispatch import receiver
```

Перш за все на початку файлу прописані основні залежності файлу та імпортовано їх. Цей фрагмент коду визначає моделі даних Django для додатка, який, здається, створює або керує ігровою платформою.

В першому рядку відбувається імпорт базового класу моделей Django. В наступному рядку коду прописано імпорт моделі користувача Django для роботи з автентифікацією. Після цього вже в наступному рядку імпорт налаштувань Django. Далі прописано імпорт наборів опцій для характеристик персонажів з іншого модуля, для ініціалізації цих наборів характеристик для використання для процесу гри. Після цього прописано імпорт сигналів Django. В рядку 7 описано

імпорт сигналу, який відправляється перед видаленням об'єкта з бази даних. Окрім цього вже в наступному рядку описано імпорт декоратора, який дозволяє підключити функцію-приймач до сигналу.

```
class Room(models.Model):
    name = models.CharField(max_length=100)
    max_players = models.IntegerField()
    password = models.CharField(max_length=100, blank=True, null=True)
    creator = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='created_rooms')
    current_player = models.ForeignKey(User, on_delete=models.SET_NULL,
null=True, blank=True,
```

У першому рядку вказується, що це клас "Room", який є підкласом "models.Model". Це означає, що він успадковує всі функції та можливості моделі Django і може бути використаний для взаємодії з базою даних. Починаючи з другого рядка, ініціалізуються основні поля моделі "Room", які визначають різні аспекти кімнати гри. Кожне поле має вказане призначення та обмеження. Є такі поля:

– Поле "name" використовується для зберігання назви кімнати гри. Його тип - "CharField" з максимальною довжиною 100 символів.

– Поле "max_players" визначає максимальну кількість гравців, які можуть перебувати в кімнаті одночасно. Для цього поля використовується "IntegerField".

– Поле "password" використовується для зберігання паролю для входу в кімнату. Воно також може бути порожнім ("blank=True") і допускає значення "NULL" ("null=True").

– Поле "creator" є ForeignKey, яке встановлює зв'язок з моделлю "User", представляючи користувача, який створив кімнату гри. Параметр "on_delete=models.CASCADE" вказує, що при видаленні користувача, всі кімнати, які він створив, також будуть видалені.

– Поле "current_player" також є ForeignKey до моделі "User", вказуючи на поточного гравця у кімнаті. Це поле може бути пустим ("null=True"), що дозволяє кімнаті не мати поточного гравця.

```
related_name='current_player_rooms')
    game_started = models.BooleanField(default=False)
    current_turn_player = models.ForeignKey(User,
on_delete=models.SET_NULL, null=True, blank=True,

related_name='current_turn_player_rooms')
    turn_ended = models.BooleanField(default=False)
    voting_started = models.BooleanField(default=False)

    def str(self):
        return self.name
```

Поля `game_started`, `turn_ended` та `voting_started` представляють різні стани гри і вказують, чи розпочалася гра, чи завершився поточний хід, чи почалось голосування в кімнаті. Кожне поле моделі `Room` відображає певний аспект кімнати гри і визначає його характеристики та можливості.

```
@receiver(pre_delete, sender=Room)
def delete_character_cards(sender, instance, **kwargs):
    places = instance.place_set.all()
    character_cards = CharacterCard.objects.filter(place__room=instance)
    character_cards.delete()
```

Ця частина коду визначає функцію "delete_character_cards", яка викликається перед видаленням об'єкта типу "Room". Декоратор "@receiver(pre_delete, sender=Room)" встановлює отримувача сигналу, що викликається перед видаленням об'єкта типу "Room", приймаючи два аргументи: тип сигналу ("pre_delete") і відправника сигналу ("sender=Room"). Функція "delete_character_cards" оголошується з трьома аргументами: "sender"

(відправник сигналу), "instance" (конкретний об'єкт "Room", який буде видалено) і "**kwargs" (додаткові аргументи). Спершу отримуються всі об'єкти "Place", пов'язані з цим об'єктом "Room" через зв'язок One-to-Many за допомогою "instance.place_set.all()". Потім отримуються всі об'єкти "CharacterCard", пов'язані з об'єктом "Room" через об'єкти "Place" за допомогою "CharacterCard.objects.filter(place__room=instance)". Нарешті, видаляються всі знайдені об'єкти "CharacterCard" за допомогою методу "character_cards.delete()". Отже, перед видаленням об'єкта "Room" функція виконує наступні дії: отримує всі пов'язані об'єкти "Place", отримує всі пов'язані об'єкти "CharacterCard" та видаляє всі знайдені об'єкти "CharacterCard".

```
class CharacterCard(models.Model):
    player        =        models.OneToOneField(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE)
    bio = models.TextField(blank=True)
    health = models.CharField(blank=True, max_length=100)
    phobia = models.CharField(blank=True, max_length=100)
    hobby = models.CharField(blank=True, max_length=100)
    knowledge = models.CharField(blank=True, max_length=100)
    additional_info = models.TextField(blank=True)
    luggage = models.TextField(blank=True)
```

Цей фрагмент коду визначає модель "CharacterCard", яка представляє інформацію про персонажа гравця в грі. Поле "player" типу "OneToOneField" вказує на користувача, що володіє цим персонажем, забезпечуючи зв'язок один-до-одного між персонажем і користувачем. Поля "bio", "health", "phobia", "hobby", "knowledge", "additional_info" і "luggage" зберігають інформацію про біографію, стан здоров'я, фобії, хобі, знання, додаткову інформацію та багаж персонажа відповідно. Ці поля представлені різними типами полів, такими як "TextField" або "CharField", залежно від природи інформації.

```
    bio_hidden = models.BooleanField(default=True)
```

```
    health_hidden = models.BooleanField(default=True)
    phobia_hidden = models.BooleanField(default=True)
    hobby_hidden = models.BooleanField(default=True)
    knowledge_hidden = models.BooleanField(default=True)
    additional_info_hidden = models.BooleanField(default=True)
    luggage_hidden = models.BooleanField(default=True)
```

Поле bio_hidden та інші: це поля типу BooleanField вказують, чи прихована від інших користувачів певна інформація про персонажа. За замовчуванням всі ці дані приховані. Це необхідно оскільки логіка гри в тому, щоб поступово, щоходу відкривати певну кількість характеристик для інших гравців і на основі відкритих характеристик дискутувати, за право залишитися в бункері. Кожному з цих полів присвоєно власне значення за замовчуванням True, що вказує на те, що інформація за замовчуванням прихована.

```
    BIO_OPTIONS = BIO_OPTIONS
    HEALTH_OPTIONS = HEALTH_OPTIONS
    PHOBIA_OPTIONS = PHOBIA_OPTIONS
    HOBBY_OPTIONS = HOBBY_OPTIONS
    KNOWLEDGE_OPTIONS = KNOWLEDGE_OPTIONS
    ADDITIONAL_INFO_OPTIONS = ADDITIONAL_INFO_OPTIONS
    LUGGAGE_OPTIONS = LUGGAGE_OPTIONS
```

Поле BIO_OPTIONS та інші: це константи визначають можливі варіанти для кожного поля. Їх призначення може бути використане для надання варіантів користувачеві під час створення персонажа. Наприклад, `BIO_OPTIONS` може містити список доступних біологічні характеристик, які можуть бути використані. За логікою гри, ці характеристики випадково генеруються, коли починається гра.

```
class Place(models.Model):
    room = models.ForeignKey('Room', on_delete=models.CASCADE)
    player_name = models.CharField(max_length=100, blank=True, null=True)
```

```
    character_card = models.ForeignKey('CharacterCard',
on_delete=models.SET_NULL, null=True, blank=True)
    turn_finished = models.BooleanField(default=False)
    can_end_turn = models.BooleanField(default=True)
    voted = models.BooleanField(default=False)
    is_kicked = models.BooleanField(default=False)
```

Цей фрагмент коду визначає модель Place, яка представляє місце в кімнаті гри, де може знаходитись персонаж гравця. В цьому класі знаходяться такі поля:

– room: поле типу ForeignKey, яке вказує на кімнату, до якої відноситься це місце. Отримання кімнати, до якої відноситься це місце, може бути здійснено за допомогою атрибута room.

– player_name: поле типу CharField, яке зберігає ім'я гравця, що займає це місце. Це поле може бути порожнім, оскільки не всі місця можуть бути зайняті.

– character_card: поле типу ForeignKey, яке вказує на карту персонажа, яка пов'язана з цим місцем. Це поле може бути порожнім (null=True), оскільки не всі місця можуть мати пов'язану карту персонажа.

– turn_finished: поле типу BooleanField, яке вказує, чи завершено хід для гравця, який займає це місце.

– can_end_turn: поле типу BooleanField, яке вказує, чи може гравець, який займає це місце, завершити свій хід.

– voted: поле типу BooleanField, яке вказує, чи проголосував гравець, який займає це місце.

– is_kicked: поле типу BooleanField, яке вказує, чи був вилучений гравець, який займав це місце.

Метод __str__ використовується для представлення об'єкта моделі у зрозумілому форматі. В цьому випадку, він повертає рядок, що вказує на те, що це місце для конкретної карти персонажа за ім'ям гравця, який займає це місце.

```
class Vote(models.Model):
    voter = models.ForeignKey(User, on_delete=models.CASCADE
    target_player = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='received_votes')
    room = models.ForeignKey(Room, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    def str(self):
        return f'Character Card for {self.player_name}'
    def str(self):
        return      f"Vote      from      {self.voter.username}      to
{self.target_player.username} in room {self.room.id}"
```

Цей фрагмент коду визначає модель "Vote", яка представляє голос, відданий гравцем в грі. Де є такі поля:

– "voter": поле типу "ForeignKey", яке вказує на користувача, який віддає голос. Воно пов'язане з моделлю "User".

– "target_player": поле типу "ForeignKey", яке вказує на гравця, на якого віддається голос. Це також поле типу "User", яке пов'язане з моделлю "User".

– "room": поле типу "ForeignKey", яке вказує на кімнату, де відбувається голосування. Воно пов'язане з моделлю "Room".

– "created_at": поле типу "DateTimeField", яке вказує на дату та час створення голосу. Цей атрибут встановлює значення поля автоматично при створенні об'єкту за допомогою параметра "auto_now_add=True".

Метод "__str__" використовується для представлення об'єкта моделі у зрозумілому форматі. У цьому випадку, він повертає рядок, який містить інформацію про голос - від кого до кого він надійшов і в якій кімнаті відбувається голосування проти гравців.

Таким чином за допомогою методу “Vote” гравці можу голосувати проти інших гравців в кімнаті, з метою їхнього вигнання з гри.

3.3.2 Веб-види проекту

У файлі views.py зберігається код веб-видів проекту, які обробляються HTTP-запити та повертаються відповіді. Далі було описано основну частину коду. За необхідності повноцінний файл можна знайти в додатку Б.

```
from django.http import HttpResponse
from django.contrib.auth.forms import UserCreationForm,
AuthenticationForm
from django.contrib.auth import login, authenticate
from django.shortcuts import redirect
from .forms import RoomForm
from django.http import JsonResponse
from django.shortcuts import render, get_object_or_404
from django.views.decorators.http import require_http_methods
from .models import Place, CharacterCard
from django.contrib.auth.models import User
```

Початок файлу "views.py" містить імпорти необхідних модулів та функцій для обробки запитів та відображення веб-сторінок. Ось відповідні імпорти зі значеннями:

– "from django.http import HttpResponse": імпорт класу "HttpResponse" з модулю "django.http". "HttpResponse" використовується для створення відповідей на HTTP-запити.

– "from django.contrib.auth.forms import UserCreationForm, AuthenticationForm": імпорт форм для реєстрації користувача ("UserCreationForm") та автентифікації ("AuthenticationForm") з модулю "django.contrib.auth.forms".

– "from django.contrib.auth import login, authenticate": імпорт функцій "login" і "authenticate" з модулю "django.contrib.auth". Вони використовуються

для автентифікації користувачів та збереження інформації про автентифікованого користувача.

– "from django.shortcuts import redirect": імпорт функції "redirect" з модулю "django.shortcuts". Вона використовується для перенаправлення користувача на іншу сторінку.

– "from .forms import RoomForm": імпорт форми "RoomForm" з поточного пакету (папки).

– "from django.http import JsonResponse": імпорт функції "JsonResponse" з модулю "django.http". "JsonResponse" використовується для відправлення відповіді у форматі JSON.

– "from django.shortcuts import render, get_object_or_404": Імпорт функцій "render" і "get_object_or_404" з модулю "django.shortcuts". "render" використовується для відображення веб-сторінок, а "get_object_or_404" - для отримання об'єкта з бази даних або відображення сторінки 404, якщо об'єкт не знайдено.

– "from django.views.decorators.http import require_http_methods": Імпорт декоратора "require_http_methods" з модулю "django.views.decorators.http". Цей декоратор використовується для обмеження доступу до певних HTTP-методів (GET, POST і т. д.).

– "from .models import Place, CharacterCard": імпорт моделей "Place" і "CharacterCard" з поточного пакету.

– "from django.contrib.auth.models import User": імпорт моделі "User" з "django.contrib.auth.models".

"from .models import Room, Vote": імпорт моделей "Room" і "Vote" з поточного пакету.

```
from django.contrib.auth.decorators import login_required
from .models import Room, Vote
import random
from django.db import transaction
```

```
from collections import Counter
from django.db.models import Count
from django.core.serializers.json import DjangoJSONEncoder
```

Цей фрагмент коду імпортує необхідні модулі та функції для роботи з моделями та додатковими утилітами у файлі "views.py":

– login_required: декоратор для обмеження доступу до представлень лише для автентифікованих користувачів.

– Room, Vote: моделі, які використовуються у програмі.

– random, transaction, Counter: бібліотеки для генерації випадкових чисел, роботи з транзакціями бази даних та підрахунку елементів.

– Count: функція для підрахунку об'єктів у запитах до бази даних.

– DjangoJSONEncoder: клас для серіалізації даних у формат JSON.

– Імпорт списків опцій з модуля "utils" для використання в інших частинах програмного коду.

```
from .utils import (
    BIO_OPTIONS,
    HEALTH_OPTIONS,
    PHOBIA_OPTIONS,
    KNOWLEDGE_OPTIONS,
    HOBBY_OPTIONS,
    ADDITIONAL_INFO_OPTIONS,
    LUGGAGE_OPTIONS,
)
```

Цей фрагмент коду імпортує кілька констант із модуля `utils` для використання в іншій частині коду. Ось що вони означають:

– BIO_OPTIONS містить варіанти біографічних даних для персонажів гри.

– HEALTH_OPTIONS містить варіанти стану здоров'я для персонажів гри.

– PHOBIA_OPTIONS містить варіанти фобій, які можуть бути присутні у персонажів гри.

– KNOWLEDGE_OPTIONS містить варіанти знань або навичок, якими володіють персонажі гри.

– HOBBY_OPTIONS містить варіанти хобі або захоплень персонажів гри.

– ADDITIONAL_INFO_OPTIONS містить додаткову інформацію або опції, які можуть бути використані для деталізації персонажів гри.

– LUGGAGE_OPTIONS містить варіанти багажу, який персонажі можуть мати при собі.

Ці імпорти необхідні для виконання різноманітних операцій, таких як автентифікація, обробка форм, взаємодія з базою даних та інші дії, що відбуваються в переглядах.

```
def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
    else:
        form = UserCreationForm()
    return render(request, 'registration/register.html', {'form': form})
```

Функція register призначена для обробки запитів на реєстрацію користувача. Функція має такі особливості:

1. Функція отримує об'єкт запиту (request), який містить інформацію, що надходить від користувача.

2. Перевіряється метод запиту. Якщо метод - POST, це означає, що користувач відправив дані форми реєстрації.

3. Якщо метод - POST, створюється екземпляр форми UserCreationForm, використовуючи дані, що надійшли в запиті (request.POST).

4. Перевіряється, чи дані у формі є вірними (проходять валідацію). Якщо так, тобто метод is_valid() повертає True, дані форми зберігаються (новий

користувач додається до бази даних), і користувач перенаправляється на сторінку автентифікації login.

5. Якщо метод - не POST, це означає, що користувач просто відкриває сторінку реєстрації, тому створюється порожній екземпляр форми UserCreationForm.

6. Метод використовує функцію render для відображення сторінки з шаблоном registration/register.html, передаючи в нього створену форму. Це дозволяє користувачеві заповнити та відправити форму для реєстрації.

```
def user_login(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                return redirect('index')
    else:
        form = AuthenticationForm()
    return render(request, 'registration/login.html', {'form': form})
```

Ця функція "user_login" відповідає за обробку запитів на аутентифікацію користувача. Метод працює таким чином:

1. Функція отримує об'єкт запиту ("request"), який містить дані, що надійшли від користувача.

2. Перевіряється метод запиту. Якщо метод - POST, це означає, що користувач відправив дані форми для входу.

3. Якщо метод - POST, створюється екземпляр форми "AuthenticationForm", використовуючи дані, що надійшли в запиті ("request.POST") на сервер.

4. Перевіряється, чи дані у формі є вірними (проходять валідацію). Якщо так, тобто метод "is_valid()" повертає "True", витягуються дані про користувача (ім'я користувача та пароль).

5. Здійснюється спроба автентифікації користувача за допомогою отриманих даних. Якщо користувач з таким ім'ям та паролем існує, відбувається вхід користувача в систему за допомогою функції "login", і користувач перенаправляється на головну сторінку ("index").

6. Якщо метод запиту не є POST, створюється порожній екземпляр форми "AuthenticationForm".

7. Функція використовує функцію "render" для відображення сторінки з шаблоном "registration/login.html", передаючи в нього створену форму. Це дозволяє користувачеві заповнити та відправити форму для входу.

```
def index(request):
    rooms = Room.objects.all()
    if request.method == 'POST':
        form = RoomForm(request.POST)
        if form.is_valid():
            room = form.save(commit=False)
            room.creator = request.user
            room.save()
            return redirect('index')
```

Ця функція "index" відповідає за відображення головної сторінки веб-додатку. Ось пояснення її роботи:

1. Функція отримує об'єкт запиту ("request"), який містить дані, що надійшли від користувача.

2. Отримуються всі кімнати з бази даних за допомогою запиту "Room.objects.all()" в коді.

3. Перевіряється метод запиту. Якщо метод - POST, це означає, що користувач відправив дані форми для створення нової кімнати.

4. Якщо метод - POST, створюється екземпляр форми "RoomForm" з даних, що надійшли в запиті ("request.POST").

5. Перевіряється, чи дані у формі є вірними (проходять валідацію). Якщо так, тобто метод "is_valid()" повертає "True", створюється нова кімната на основі валідних даних. Користувач, який створює кімнату, встановлюється як поточний користувач для гри ("request.user").

6. Форма не зберігається в базі даних автоматично, тому викликається метод "save(commit=False)", щоб створити екземпляр моделі "Room", але ще не зберегти його.

7. Після того як створена кімната, відбувається перенаправлення користувача на головну сторінку.

```
    else:
        form = RoomForm()
    context = {
        'rooms': rooms,
        'form': form,
    }
    return render(request, 'index.html', context)
```

В наданій частині коду описано невеликий функціонал, що допомагає завершити логіку попереднього коду. Згідно логіки цей код виконує наступну функцію:

1. Якщо метод запиту не є POST, створюється порожній екземпляр форми "RoomForm" в грі.

2. Дані про кімнати та створену форму передаються в контекст шаблону.

3. Функція використовує функцію "render" для відображення сторінки з шаблоном "index.html", передаючи в нього контекст. Це дозволяє відображати інформацію про кімнати та форму на головній сторінці.

```
def room_detail(request, room_id):
    room = get_object_or_404(Room, id=room_id)
```

```
    places = room.place_set.all()
    if not places:
        for i in range(room.max_players):
            place = Place.objects.create(room=room)
            places = room.place_set.all()
    players = [place.player_name for place in places if
place.player_name]
    player_users = User.objects.filter(username__in=players)
    return render(request, 'room_detail.html',
                  {'room': room, 'places': places, 'room_id': room_id,
'players': player_users})
```

Функція “room_detail” відповідає за відображення деталей конкретної кімнати. Також вона має таку логіку:

1. Функція отримує об'єкт запиту ("request") та ідентифікатор кімнати ("room_id"), за яким потрібно знайти відповідну кімнату.

2. Використовуючи допоміжну функцію "get_object_or_404", отримується об'єкт кімнати з бази даних за заданим ідентифікатором. Якщо кімната не знайдена, повертається сторінка з помилкою 404.

3. Потім отримуються або створюються місця для цієї кімнати. Якщо місця ще не існують (наприклад, коли кімната щойно створена), то створюються місця для гравців у кількості, яка відповідає максимальній кількості гравців, які можуть перебувати в кімнаті.

4. Отримується список гравців у кімнаті через модель "Place".

5. За допомогою отриманих імен гравців формується список об'єктів "User", які представляють гравців у кімнаті.

6. Функція використовує функцію "render" для відображення сторінки з шаблоном "room_detail.html", передаючи в нього інформацію про кімнату, список місць, ідентифікатор кімнати та гравців у кімнаті.

```
def take_place(request, room_id, place_id):
    room = get_object_or_404(Room, id=room_id)
```

```
    existing_place =
room.place_set.filter(player_name=request.user.username).first()
    if existing_place:
        existing_place.player_name = None
        existing_place.save()
    place = get_object_or_404(Place, id=place_id, room=room)
    if place.player_name:
        return JsonResponse({'error': 'Place already taken'})
    place.player_name = request.user.username
    place.save()
    return JsonResponse({'player_name': request.user.username})
```

Функція `take_place` відповідає за процес резервування місця для користувача у конкретній кімнаті. Вона працює таким чином:

1. Функція отримує об'єкт запиту (`request`), ідентифікатор кімнати (`room_id`) та ідентифікатор місця (`place_id`), за яким потрібно знайти відповідні об'єкти.

2. Використовуючи допоміжну функцію `get_object_or_404`, отримується об'єкт кімнати з бази даних за заданим ідентифікатором. Якщо кімната не знайдена, повертається сторінка з помилкою 404.

3. Перевіряється, чи користувач вже має місце в цій кімнаті. Якщо так, його поточне місце звільняється.

4. Отримується об'єкт місця за заданим ідентифікатором у вказаній кімнаті.

5. Перевіряється, чи місце вільне. Якщо ні, повертається відповідь JSON з помилкою, що місце вже зайняте.

6. Якщо місце вільне, користувач займає його, ім'я користувача зберігається у полі `player_name` об'єкту місця, а зміни зберігаються до правильної бази даних .

7. Повертається відповідь JSON з ім'ям користувача, яке зайняло місце.

Декоратор `@require_http_methods(["DELETE"])` вказує, що ця функція оброблятиме тільки HTTP-запити DELETE.

```
def delete_room(request, room_id):
    room = get_object_or_404(Room, id=room_id)
    if request.user == room.creator:
        room.delete()
        return HttpResponse(status=204)
    else:
        return HttpResponse(status=403)
```

Ця функція `delete_room` відповідає за видалення кімнати з системи. Метод працює таким чином:

1. Декоратор “@require_http_methods(["DELETE"])” вказує, що ця функція може бути викликана лише методом DELETE HTTP-запиту.

2. Функція отримує об'єкт запиту (“request”) та ідентифікатор кімнати (“room_id”), за яким потрібно здійснити видалення.

3. Використовуючи допоміжну функцію “get_object_or_404”, отримується об'єкт кімнати з бази даних за заданим ідентифікатором. Якщо кімната не знайдена, повертається сторінка з помилкою 404.

4. Перевіряється, чи поточний користувач є творцем кімнати. Якщо так, кімната видаляється методом “delete()”.

5. Якщо користувач не є творцем кімнати, повертається відповідь з HTTP-кодом 403, що означає заборону доступу.

6. У випадку успішного видалення, повертається відповідь з HTTP-кодом 204, що означає успішне виконання операції без повернення якого-небудь контенту.

```
@login_required
def start_game(request, room_id):
    room = get_object_or_404(Room, id=room_id)
    places = room.place_set.all()
    place = places.filter(player_name=request.user.username).first()
    if not place:
```

```
        return HttpResponse(status=403)
    if room.game_started:
        return HttpResponse("The game has already started")
    room.game_started = True
    room.current_turn_player = request.user
    room.save()
```

Наступна функція названа `start_game` відповідає за запуск гри у вказаній кімнаті. Ось пояснення її роботи:

1. Декоратор `@login_required` вказує, що ця функція може бути викликана тільки для аутентифікованих користувачів.

2. Функція отримує об'єкт запиту (`request`) та ідентифікатор кімнати (`room_id`), за яким потрібно здійснити запуск гри.

3. Використовуючи допоміжну функцію `get_object_or_404`, отримується об'єкт кімнати з бази даних за заданим ідентифікатором. Якщо кімната не знайдена, повертається сторінка з помилкою 404.

4. Отримується список всіх місць у кімнаті.

5. Перевіряється, чи поточний користувач вже зайняв місце у цій кімнаті. Якщо ні, повертається відповідь з HTTP-кодом 403, що означає заборону доступу до місця.

6. Перевіряється, чи гра вже почалася. Якщо так, повертається повідомлення про те, що гра вже розпочалася.

7. Якщо гра ще не розпочалася, встановлюється прапорець `game_started` в `True`, щоб позначити початок гри.

8. Поточним гравцем (гравцем, який ініціював запит) призначається користувач, який здійснив запит.

9. Зміни зберігаються до бази даних.

```
    with transaction.atomic():
        for place in places:
            if place.player_name:
```

```
                character_card = CharacterCard.objects.create(
                    player=User.objects.get(username=place.player_name),
                    bio=random.choice(BIO_OPTIONS) if BIO_OPTIONS else
'',
                    health=random.choice(HEALTH_OPTIONS) if
HEALTH_OPTIONS else '',
                    phobia=random.choice(PHOBIA_OPTIONS) if
PHOBIA_OPTIONS else '',
                    hobby=random.choice(HOBBY_OPTIONS) if HOBBY_OPTIONS
else '',
                    knowledge=random.choice(KNOWLEDGE_OPTIONS) if
KNOWLEDGE_OPTIONS else '',

additional_info=random.choice(ADDITIONAL_INFO_OPTIONS) if
ADDITIONAL_INFO_OPTIONS else '',
                    luggage=random.choice(LUGGAGE_OPTIONS) if
LUGGAGE_OPTIONS else '',
                )
                place.character_card = character_card
                place.save()
    return JsonResponse({'message': 'Гра успішно розпочалася'})
```

В коді вище описано таку логіку:

1. За допомогою транзакції роздаються характеристики гравцям. Для кожного місця створюється новий об'єкт `CharacterCard` з випадковими характеристиками зі списку доступних варіантів. Потім цей об'єкт присвоюється об'єкту місця.

2. Повертається відповідь JSON, що підтверджує успішний початок гри.

```
def endTurn(request, room_id):
    room = get_object_or_404(Room, id=room_id)
    voting_id_room = room_id
    if not room.game_started:
```

```
        return JsonResponse({'error': 'Гра ще не розпочалася'},
status=400)
    place = Place.objects.filter(room=room,
player_name=request.user.username).first()
    if not place:
        return JsonResponse({'error': 'Ви не перебуваєте в цій кімнаті'},
status=400)
```

На перший погляд проста функція `endTurn` відповідає за завершення ходу гравця в кімнаті. Проте вона має важливу функцію для процесу гри, а саме вона робить наступні дії:

1. Отримується об'єкт кімнати за ідентифікатором `room_id` за допомогою допоміжної функції `get_object_or_404`. Якщо кімната не знайдена, повертається сторінка з помилкою 404.

2. Перевіряється, чи гра вже почалася. Якщо гра не розпочалася, повертається відповідь JSON з помилкою та статусом 400.

3. Отримується місце поточного гравця у кімнаті.

4. Перевіряється, чи гравець перебуває в цій кімнаті. Якщо ні, повертається відповідь JSON з помилкою та статусом 400.

```
    if place.turn_finished:
        return JsonResponse({'error': 'Ви вже завершили свій хід'},
status=400)
    if not place.can_end_turn:
        return JsonResponse({'error': 'Ви не можете завершити свій хід'},
status=400)
    if place.is_kicked:
        room.place_set.remove(place)
        room.save()
        current_place_id = place.id
        next_place = Place.objects.filter(room=room,
id__gt=current_place_id).first()
        all_players_finished_turn = all(place.turn_finished for place in
room.place_set.all())
```

```
        if next_place is None:
            if all_players_finished_turn:
                start_voting(voting_id_room)
            return JsonResponse({'message': 'Всі гравці завершили хід'})
```

В другій частині функції описується не менш важлива частина коду, яка робить наступні дії:

1. Перевіряється, чи хід гравця ще не завершений. Якщо хід вже завершено, повертається відповідь JSON з помилкою та статусом 400.

2. Перевіряється, чи гравець може закінчити хід. Якщо гравець не може закінчити хід, повертається відповідь JSON з помилкою та статусом 400.

3. Перевіряється, чи гравець не був вигнаний. Якщо так, він видаляється зі списку гравців, які ще не завершили хід. Потім визначається наступний гравець, якому потрібно зробити хід.

4. Якщо всі гравці вже завершили свій хід, починається голосування за вигнання гравця.

```
        else:
            next_player_name = next_place.player_name
            next_player = User.objects.get(username=next_player_name)
            room.current_turn_player = next_player
            room.save()
            return JsonResponse({'message': 'Хід завершено успішно'})
    place.turn_finished = True
    place.can_end_turn = False
    place.save()
    current_place_id = place.id
    next_place = Place.objects.filter(room=room,
id__gt=current_place_id).first()
    all_players_finished_turn = all(place.turn_finished for place in
room.place_set.all())
    if next_place is None:
        if all_players_finished_turn:
            start_voting(voting_id_room)
```

```
            return JsonResponse({'message': 'Всі гравці завершили хід'})
        else:
            next_player_name = next_place.player_name
            next_player = User.objects.get(username=next_player_name)
            room.current_turn_player = next_player
            room.save()
            return JsonResponse({'message': 'Хід завершено успішно'})
```

В кінці функції виконуються завершальні дії з важливою для правильної роботи логіко:

1. Якщо гравець ще не був вигнаний, його хід позначається як завершений.

2. Отримується наступний гравець, який ще не завершив хід. Якщо немає наступного гравця, і всі гравці завершили свій хід, починається голосування за вигнання гравця.

3. Якщо є наступний гравець, оновлюється поле `current_turn_player` на наступного гравця.

4. Повертається відповідь у форматі JSON, що підтверджує успішне завершення ходу.

```
def start_voting(room_id):
    room = get_object_or_404(Room, id=room_id)
    if not room.game_started:
        return JsonResponse({'error': 'Гра ще не розпочалася'},
status=400)
    places_in_room = Place.objects.filter(room=room)
    all_players_finished_turn = all(place.turn_finished for place in
places_in_room)
```

Цей фрагмент коду виглядає як функція Python, призначений для початку голосування в кімнаті ідентифікованій за допомогою `room_id`. В ньому міститься наступна логіка:

1. `get_object_or_404`: Це функція, яка намагається отримати об'єкт кімнати з бази даних за допомогою ідентифікатора `room_id`. Якщо об'єкт не

знайдено, вона повертає помилку 404 (Not Found). Це може бути частиною фреймворку, такого як Django.

2. Перевірка, чи гра вже почалася (`if not room.game_started`). Якщо `game_started` дорівнює False, функція повертає помилку у формі JSON з повідомленням про те, що гра ще не розпочалася. Це передбачає, що в кімнаті є параметр `game_started`, який вказує, чи розпочата гра.

3. Отримання списку місць в кімнаті (`places_in_room`) за допомогою фільтрації об'єктів типу `Place`, які належать даній кімнаті.

4. Перевірка, чи всі гравці завершили свій хід (`all(place.turn_finished for place in places_in_room)`). Це використовує вбудовану функцію Python `all`, яка повертає True, якщо всі елементи ітерабельного об'єкта є істинними. У цьому випадку, для кожного місця (place) в кімнаті перевіряється, чи завершив гравець свій хід. Якщо всі гравці завершили свій хід, тоді код продовжує виконання.

```
    if not all_players_finished_turn:
        return JsonResponse({'error': 'Не всі гравці завершили свій
хід'}, status=400)
    room.voting_started = True
    room.turn_ended = True
    room.save()
    return JsonResponse({'message': 'Голосування розпочато'})
```

Ця частина коду виконується, якщо не всі гравці завершили свій хід:

1. Повертається відповідь у форматі JSON з помилкою та статусом 400, що свідчить про те, що не всі гравці завершили свій хід.

Якщо всі гравці завершили свій хід:

1. Параметру `voting_started` кімнати присвоюється значення True. Це, ймовірно, показує, що голосування розпочалося.

2. Параметру `turn_ended` кімнати також присвоюється значення True. Це може вказувати на те, що поточний раунд гри завершено.

3. Зміни зберігаються у базі даних за допомогою методу `save()`.

4. Повертається відповідь у форматі JSON з повідомленням про те, що голосування розпочалося.

```
def vote_endpoint(request):
    if request.method == 'POST':
        selected_player_name = request.POST.get('selected_player_name')
        if not selected_player_name:
            return JsonResponse({'error': 'Не вказано гравця для
голосування'}, status=400)
        voter = request.user
        room_id = request.POST.get('room_id')
        room = get_object_or_404(Room, id=room_id)
```

Цей фрагмент коду призначений для обробки HTTP-запитів типу POST, що надходять на ендпоінт для голосування. Він виконує такі функціональні задачі в проекті:

1. Перевірка, чи метод запиту є POST. Це важливо для забезпечення безпеки та відповідності REST-принципам. Якщо метод не є POST, то, ймовірно, буде викинута помилка.

2. Отримання імені обраного гравця для голосування з POST-запиту. Якщо ім'я не було вказане (або воно пусте), то повертається відповідь з помилкою 400, що свідчить про невірний або неповний запит.

3. Отримання об'єкта користувача, який голосує. Це зазвичай здійснюється через `request.user`, який містить інформацію про поточного користувача, залогіненого у системі.

4. Отримання ідентифікатора кімнати, в якій проводиться голосування, також з POST-запиту.

5. Спроба отримати об'єкт кімнати за допомогою ідентифікатора `room_id`. Якщо кімната не знайдена, відповідь буде статусом 404, що означає, що запитана сторінка не існує.

```
        try:
            place = Place.objects.get(room=room,
player_name=voter.username)
        except Place.DoesNotExist:
            return JsonResponse({'error': 'Ви не берете участь у цій грі
або вас вже викреслено з кімнати'}, status=400)
```

У цьому фрагменті коду відбувається спроба отримати об'єкт місця (place) за допомогою моделі Place. Параметри цього об'єкта - кімната (room) та ім'я гравця (player_name), яке дорівнює ім'я користувача, який голосує (voter.username).

Якщо спроба отримати об'єкт не вдається (Place.DoesNotExist), це означає, що гравець не бере участь у грі або вже був викреслений з кімнати. У такому випадку повертається відповідь у форматі JSON із відповідним повідомленням про помилку та статусом 400 (Bad Request), щоб повідомити клієнта про те, що щось пішло не так з його сторони.

```
        try:
            target_player =
User.objects.get(username=selected_player_name)
        except User.DoesNotExist:
            return JsonResponse({'error': 'Обраний гравець не
знайдений'}, status=400)
        existing_vote = Vote.objects.filter(voter=voter,
room=room).exists()
        if existing_vote:
            return JsonResponse({'error': 'Ви вже проголосували у цій
кімнаті'}, status=400)
        Vote.objects.create(voter=voter, target_player=target_player,
room=room)
        place.voted = True
        place.save()
```

Вже в наступному фрагменті коду спочатку спробуємо знайти користувача за його ім'ям. Якщо користувача не знайдено, повертається відповідь з помилкою "Обраний гравець не знайдений".

Після цього перевіряється, чи користувач вже голосував у цій кімнаті. Якщо так, повертається відповідь з помилкою "Ви вже проголосували у цій кімнаті" у форматі повідомлення.

Якщо обидві перевірки пройшли успішно, створюється новий голос (Vote) з вказаною інформацією. Крім того, позначається, що гравець проголосував, зміни зберігаються у базі даних.

```
        all_players_voted = all(place.voted for place in
room.place_set.all())
        if all_players_voted:
            winners = determine_winner(room_id)
            winner_names = ', '.join(winner.username for winner in
winners)
            for winner in winners:
                winner_place = Place.objects.get(room=room,
player_name=winner.username)
                winner_place.is_kicked = True
                winner_place.save()
            message = f'Гравець {winner_names} за результатом
голосування'
```

У цьому фрагменті коду перевіряється, чи всі гравці в кімнаті проголосували. Якщо так, визначаються переможці гри, їх імена об'єднуються у рядок. Кожен переможець позначається як викреслений з кімнати, та зберігаються зміни у базі даних. Формується повідомлення про переможців гри.

```
            start_new_turn(room_id)
            return JsonResponse({'message': message}, status=200)
        else:
```

```
            return JsonResponse({'message': 'Голосування зараховано'},
status=200)
    else:
        return JsonResponse({'error': 'Метод не підтримується'},
status=405)
```

Отримавши всі голоси, розпочинається новий раунд гри, і повідомлення про переможців відправляється користувачам. Якщо всі гравці проголосували, раунд завершується і гра переходить до наступного етапу. Інакше, якщо не всі гравці проголосували, голосування залишається відкритим.

Але, якщо метод запиту не є POST, повертається помилка "Метод не підтримується" зі статусом 405 (Method Not Allowed).

```
def start_new_turn(room_id):
    room = Room.objects.get(id=room_id)
    places_in_room = Place.objects.filter(room=room)
    first_place = places_in_room.filter(is_kicked=False).first()
    if first_place:
        first_player = User.objects.get(username=first_place.player_name)
        room.current_turn_player = first_player
        room.save()
```

Код отримує об'єкт кімнати та фільтрує всі місця у кімнаті. Потім він вибирає перше місце без прапорця викреслення. Якщо таке місце знайдено, визначається гравець цього місця як поточний гравець кімнати, та зміни зберігаються.

```
        for place in places_in_room:
            if not place.is_kicked:
                place.turn_finished = False
                place.can_end_turn = True
                place.voted = False
                place.save()
    room.game_started = True
    room.voting_started = False
```

```
    room.turn_ended = False
    room.save()
    return JsonResponse({'message': 'Початок нового ходу'})
```

Цей фрагмент коду встановлює прапорці для всіх місць у кімнаті, щоб позначити початок нового раунду гри. Після цього, прапорці `game_started`, `voting_started` та `turn_ended` для кімнати також встановлюються. Нарешті, повертається відповідь у форматі JSON з повідомленням про початок нового ходу у грі.

```
def determine_winner(room_id):
    vote_counter = calculate_votes(room_id)
    if vote_counter:
        max_votes = max(vote_counter.values())
        winners = [player for player, votes in vote_counter.items() if
votes == max_votes]
        return winners
    else:
        return None
```

Ця функція визначає переможців гри для певної кімнати. Вона спочатку обчислює кількість голосів для кожного гравця, використовуючи іншу функцію `calculate_votes`. Після цього вона визначає гравців з максимальною кількістю голосів та повертає їх як переможців. Якщо не було отримано жодного голосу, функція повертає `None`.

```
def calculate_votes(room_id):
    room = Room.objects.get(id=room_id)
    all_votes = Vote.objects.filter(room=room)
    vote_counter = Counter(vote.target_player for vote in all_votes)
    return vote_counter
```

Наступна функція обчислює кількість голосів для кожного гравця у певній кімнаті. Вона спочатку отримує об'єкт кімнати за допомогою ідентифікатора `room_id`. Потім вона фільтрує всі голоси, що належать до цієї кімнати. Кількість голосів для кожного гравця обчислюється за допомогою лічильника `Counter` з модуля `collections`. Нарешті, функція повертає цей лічильник, де ключі - це гравці, а значення - це кількість голосів, які вони отримали.

```
def get_vote_results(request, room_id):
    vote_counts = (
        Vote.objects.filter(room_id=room_id)
        .values('target_player__username')
        .annotate(total_votes=Count('id'))
    )
    vote_counts_list = list(vote_counts)
    return JsonResponse({'voteCounts': vote_counts_list},
encoder=DjangoJSONEncoder)
```

Фрагмент коду визначає функцію `get_vote_results`, яка призначена для отримання результатів голосування у вказаній кімнаті. Код виконує такі функціональні задачі:

1. Функція виконує запит до бази даних, щоб отримати кількість голосів за кожного гравця у вказаній кімнаті. Для цього вона використовує метод `filter` для моделі `Vote`, де `room_id` дорівнює переданому `room_id`. Метод `values` дозволяє вибрати поля, які повернуться з запиту, у цьому випадку це `target_player__username`, що вказує на ім'я гравця, за якого проголосували. Метод `annotate` додає до кожного запису поле `total_votes`, яке представляє загальну кількість голосів за кожного гравця.

2. Результат запиту зберігається у змінній `vote_counts`.

3. Звертаючись до `QuerySet` `vote_counts`, функція перетворює його в список словників з використанням `list(vote_counts)`. Кожен словник містить ім'я гравця (`target_player__username`) і загальну кількість голосів (`total_votes`).

4. Нарешті, результат повертається у форматі JSON за допомогою `JsonResponse`. Ключ `''voteCounts''` містить список словників, який містить ім'я гравця та кількість голосів за нього. Щоб коректно обробити об'єкти дати Python у JSON, використовується параметр `encoder=DjangoJSONEncoder`.

Після цього методу створено метод для умовного відкривання характеристик для інших гравців.

```
@login_required
def toggle_visibility(request, character_card_id, characteristic):
    try:
        character_card = CharacterCard.objects.get(id=character_card_id)
    except CharacterCard.DoesNotExist:
        return JsonResponse({'error': 'Character card not found'},
status=404)
    if character_card.player != request.user:
        return JsonResponse({'error': 'You are not the owner of this
character card'}, status=403)
```

Ця функція, доступна лише для залогованих користувачів, змінює видимість певної характеристики на карті персонажа. Спочатку перевіряється існування карти персонажа за її ідентифікатором. Якщо карта не знайдена, повертається відповідь з помилкою "Character card not found" і статусом 404. Далі перевіряється, чи користувач є власником цієї карти. Якщо ні, повертається відповідь з помилкою "You are not the owner of this character card" і статусом 403.

```
    if characteristic == 'bio':
        character_card.bio_hidden = not character_card.bio_hidden
    elif characteristic == 'health':
        character_card.health_hidden = not character_card.health_hidden
    elif characteristic == 'phobia':
        character_card.phobia_hidden = not character_card.phobia_hidden
    elif characteristic == 'hobby':
        character_card.hobby_hidden = not character_card.hobby_hidden
```

```
    elif characteristic == 'knowledge':
        character_card.knowledge_hidden = not
character_card.knowledge_hidden
```

Цей фрагмент коду змінює видимість конкретної характеристики карти персонажа. Якщо `characteristic` дорівнює `'bio'`, змінюється видимість біографії (`bio_hidden`), якщо `'health'`, змінюється видимість здоров'я (`health_hidden`), якщо `'phobia'`, змінюється видимість фобії (`phobia_hidden`), якщо `'hobby'`, змінюється видимість хобі (`hobby_hidden`), а якщо `'knowledge'`, змінюється видимість знань (`knowledge_hidden`).

```
    elif characteristic == 'additional_info':
        character_card.additional_info_hidden = not
character_card.additional_info_hidden
    elif characteristic == 'luggage':
        character_card.luggage_hidden = not character_card.luggage_hidden
    else:
        return JsonResponse({'error': 'Invalid characteristic'},
status=400)
    character_card.save()
    return JsonResponse({'success': 'Visibility toggled successfully'})
```

Цей фрагмент коду додає обробку для двох додаткових характеристик. Якщо `characteristic` дорівнює `'additional_info'`, змінюється видимість додаткової інформації (`additional_info_hidden`), якщо `'luggage'`, змінюється видимість багажу (`luggage_hidden`). Якщо ж значення `characteristic` не відповідає жодному з перерахованих, повертається відповідь з помилкою "Invalid characteristic" і статусом 400. Після зміни видимості відповідної характеристики, зберігаються зміни об'єкта `character_card` і повертається відповідь з повідомленням про успішне виконання операції.

3.3.3 Файли відображення сторінок

Для початку розглянемо файл index.html, що відповідає за список кімнат де проходять ігри, та форму для створення нової кімнати.

На початку файлу одразу одразу прописано {% include 'header.html' %}, що використовується для включення вмісту іншого шаблону у поточний. Після чого підключається Bootstrap для його використання з метою стилізації сторінки <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">

Вже в середині тегу body розміщений основний код що оформляє сторінку. HTML-код зображений на рисунку (рис 3.1) відображає сторінку для управління кімнатами. Сторінка містить два блоки: перший призначений для перегляду створених кімнат, а другий - для створення нові кімнати. У першому блоку форма пошуку кімнат за їх назвою. Кожна кімната відображається у вигляді списку з назвою та кількістю гравців.

```
    <div class="form-group flex-grow-1 mr-2">
        <label for="searchRoom" class="sr-only">Search Room</label>
        <input type="text" class="form-control rounded-20 w-100" id="searchRoom" placeholder="Search by room name">
    </div>
    <button type="submit" class="btn rounded-20" style="...">Search</button>
</form>
<ul class="list-unstyled">
    {% for room in rooms %}
    <li class="room-item border-bottom py-2">
        <div class="d-flex justify-content-between align-items-center" style="...">
            <div class="room-details">
                <span class="room-name mr-2 ml-3">{{ room.name }}</span>
            </div>
            <div class="actions mr-2 my-2 d-flex align-items-center">
                <div>
                    <span class="room-name mr-2 ml-3">{{ room.name }}</span>
                </div>
                <div class="mr-auto">
                    <span class="player-count badge badge-pill badge-light">{{ room.players.count }}</span>
                    <span class="player-label ml-2">players</span>
                </div>
                <a href="{% url 'room_detail' room.id %}" class="btn btn-dark connect-button ml-2">Connect</a>
            </div>
```

Рисунок 3.1 – Перша частина файлу index.html

На наступному рисунку (рис 3.2) зображено функціонал, коли користувач може обрати кімнату та доєднатися до неї за допомогою кнопки "Connect". Другий блок містить форму для створення нової кімнати. Користувач вводить назву кімнати у відповідному полі та натискає кнопку "Create Room", щоб створити нову кімнату.

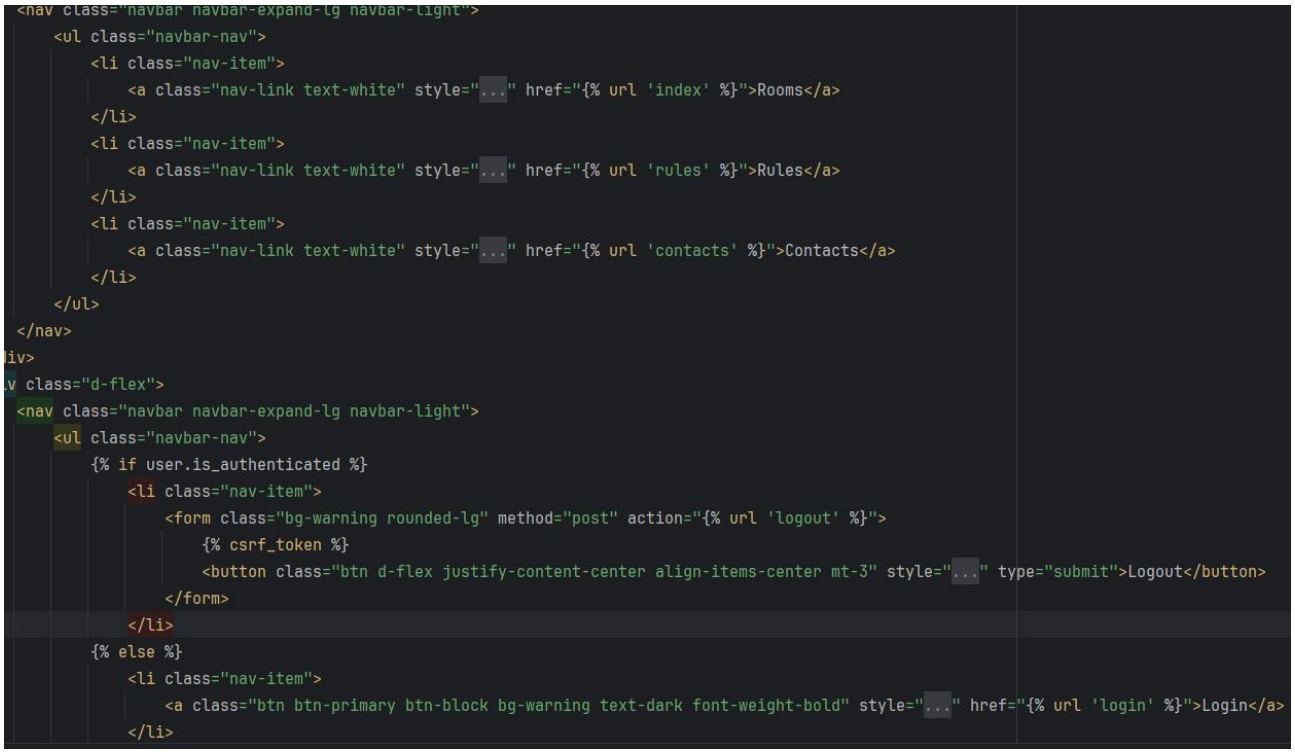
```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit" class="btn rounded-20" style="color: #000000; background-color: #F2B705; width: 290px; height: 63
</form>
```

Рисунок 3.2 – Друга частина файлу index.html

На наступному рисунку (рис 3.3) код створює HTML-заголовок `<header>` для веб-сторінки, стилізований за допомогою Bootstrap і CSS стилів. Стилізація заголовка включає фон кольору #222222 (темно-сірий) та контейнер , який розташовує елементи в одній лінії з рівними відступами.

Логотип містить посилання на головну сторінку. Окрім цього заголовок логотипу має клас `text-warning`, що задає йому жовтий колір. Навігаційне меню розділене на дві частини: перше використовує компонент навігаційної панелі Bootstrap, та складається з трьох елементів (`Rooms`, `Rules`, `Contacts`), кожен з яких має посилання і білий текст (`text-white`) шрифтом розміром 20px.

Друге навігаційне меню розташоване у `<div class="d-flex">` і містить блок, який перевіряє, чи користувач аутентифікований `{% if user.is_authenticated %}`. Якщо користувач аутентифікований, відображається форма з кнопкою `Logout` для виходу з облікового запису, яка має жовтий фон `bg-warning` та округлі краї `border-radius: 1.5rem`. Якщо користувач не аутентифікований, відображається кнопка `Login` для входу в систему, стилізована як жовта `bg-warning` з чорним текстом і жирним шрифтом `text-dark font-weight-bold`, а також з чорним обрамленням `border-color: black; border-width: 2px`.



```
<nav class="navbar navbar-expand-lg navbar-light">
    <ul class="navbar-nav">
        <li class="nav-item">
            <a class="nav-link text-white" style="..." href="{% url 'index' %}">Rooms</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-white" style="..." href="{% url 'rules' %}">Rules</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-white" style="..." href="{% url 'contacts' %}">Contacts</a>
        </li>
    </ul>
</nav>
iv>
v class="d-flex">
<nav class="navbar navbar-expand-lg navbar-light">
    <ul class="navbar-nav">
        {% if user.is_authenticated %}
            <li class="nav-item">
                <form class="bg-warning rounded-lg" method="post" action="{% url 'logout' %}">
                    {% csrf_token %}
                    <button class="btn d-flex justify-content-center align-items-center mt-3" style="..." type="submit">Logout</button>
                </form>
            </li>
        {% else %}
            <li class="nav-item">
                <a class="btn btn-primary btn-block bg-warning text-dark font-weight-bold" style="..." href="{% url 'login' %}">Login</a>
            </li>
```

Рисунок 3.3 – Файл header.html

На зображенні (рис 3.4) зображений код, який використовується для відображення сторінки з правилами гри.

Використовується стандартна структура HTML сторінки і як і на попередніх сторінках та імпортується Bootstrap для того, щоб реалізувати дизайн рішення.

На сторінці з правилами використовуються теги h1 та h2 для позначення заголовків та підзаголовків з метою структурування контенту на сторінці. Теги p використовуються для позначення абзаців, що також допомагає у структуризації контенту. Окрім цього код створює заголовок веб-сторінки з темним фоном, логотипом "Bunker", та навігаційним меню.

У меню є посилання на сторінки "Rooms", "Rules" і "Contacts". Крім того, праворуч відображається кнопка "Logout" для автентифікованих користувачів або кнопка "Login" для неавтентифікованих, стилізовані відповідно до дизайну.

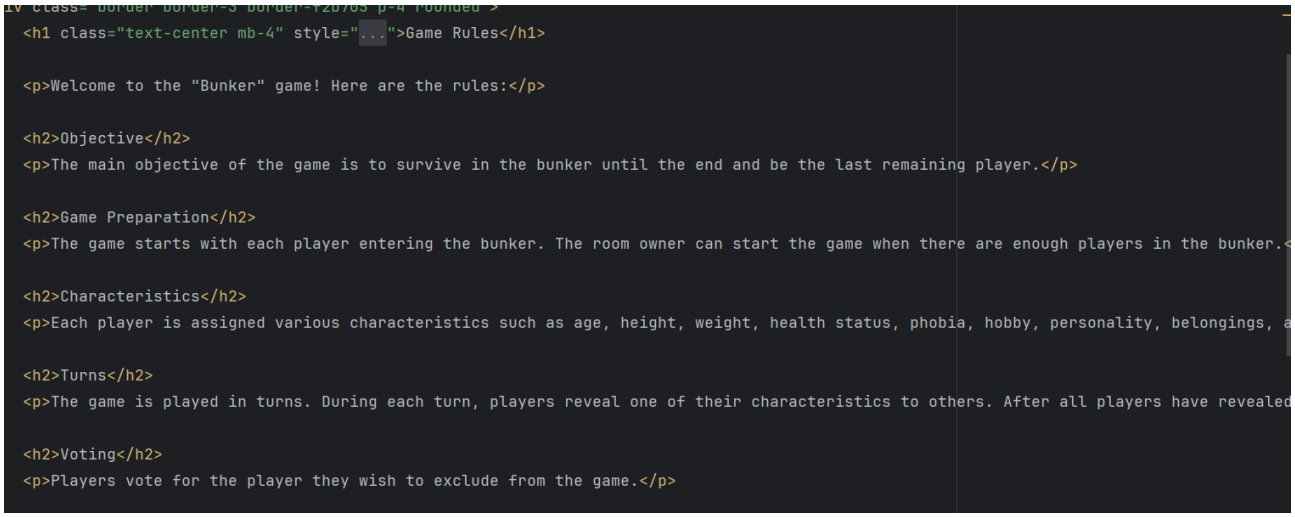


Рисунок 3.4 – Файл rules.html

На наступному рисунку (рис 3.5) зображений код, в якому створено сторінку "Contact Us", що містить контактну інформацію для зв'язку з веб-сайтом. Спочатку використовується {% include 'header.html' %} для включення заголовку, який, як очікується, містить у собі основні елементи дизайну, використовуючи Bootstrap і CSS.

HTML-код починається з визначення мови та кодування сторінки. Заголовок встановлюється як "Contact Us". Підключається CSS-файл Bootstrap для стилізації.

Всередині тіла документу встановлюється чорний фон. Контейнер визначається з класом "container" для вирівнювання та маргінів. У цьому контейнері розміщується основний контент сторінки.

Структура сторінки включає рядок з одним стовпцем шириною 8 колонок. У середині стовпця розташована картка з контактною інформацією. Заголовок "Contact Us" відображається в центрі сторінки і має жовтий колір.

Контактна інформація представлена відомостями про електронну пошту з посиланням, яке відкриває стандартний електронний клієнт з вказаною адресою. Текст контактної електронної адреси стилізований білим кольором для контрастності з чорним фоном.

```
<div class="mb-4">
    <h5>Email:</h5>
    <p class="text-white"><a href="mailto:marian.l.hrab@gmail.com" class="text-decoration-none text-white">marian.l.hrab@gmail.com</a
</div>
```

Рисунок 3.5 – Файл contacts.html

У файлі registration.html зберігається код, який окрім стандартного імпортування Bootstrap та header, також використовує форми з методом POST для надсилання даних на сервер (рис 3.6). Після чого використовується токен {% csrf_token %}. Цей токен використовується для захисту від міжсайтового скриптінгу. Далі виводиться поля форми реєстрації у вигляді абзаців, що зроблено з метою відображення кожного з полів в новому рядку. В кінці цього блоку форми знаходить кнопка Register, яка надсилає форму на сервер. Після чого знаходиться посилання на сторінку з логіном у випадку якщо користувач вже зареєстрований і хоче зробити логін.

```
ntainer mt-5">
="row justify-content-center">
lass="col-md-6">
iv class="card text-white mt-5" style="...">
  <div class="card-header">Register</div>
  <div class="card-body">
      <form method="post">
          {% csrf_token %}
          {{ form.as_p }}
          <button type="submit" class="btn btn-primary btn-block bg-warning text-dark font-weight-bold" style="...">Register</button>
      </form>
  </div>
  <div class="card-footer text-center " style="...">
      <a href="{% url 'register' %}" class="text-decoration-none text-warning">Already registered? Login here</a>
  </div>
div>
```

Рисунок 3.6 – Файл registration.html

Схоже до файлу registration.html у файлі login.html знаходиться форма з полями логін та пароль відповідно (рис 3.7). Окрім цього розміщений токен csrf та кнопка Login, що надсилає дані форму на сервер. Після цього розміщене

посилання на сторінку реєстрації у випадку якщо користувач ще не має зареєстрованого профілю, або потрапив на сторінку випадково.

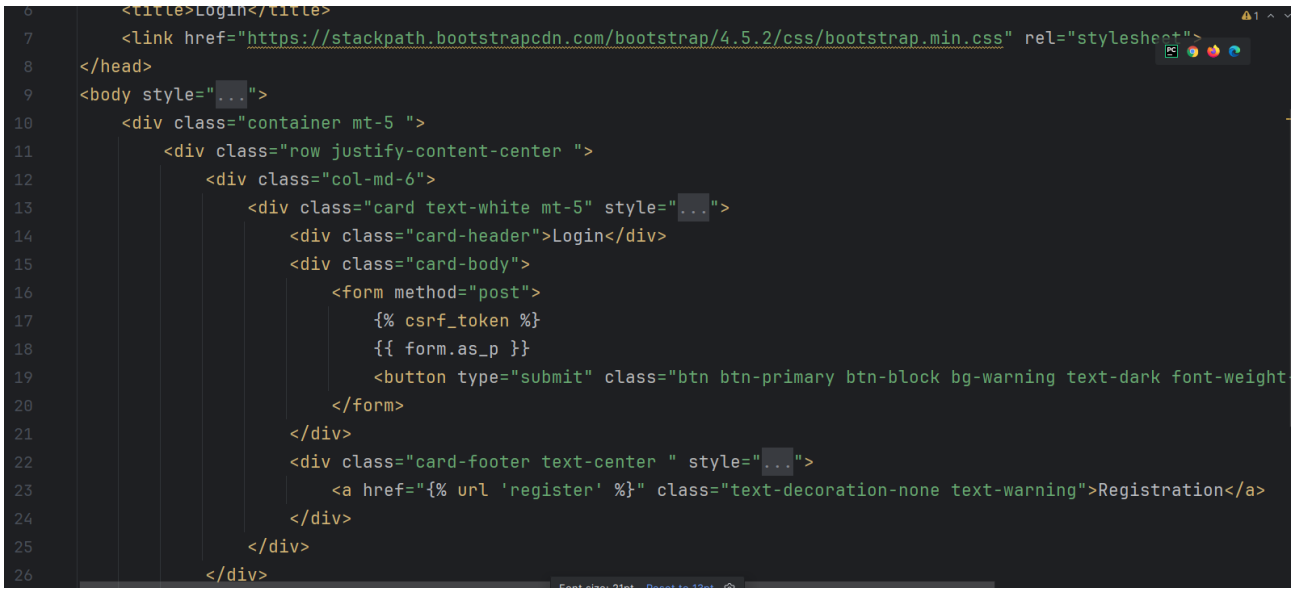


Рисунок 3.7 – Файл login.html

Останнім і найбільшим є файл room_details.html, в якому відбуваються основні процеси гри (рис 3.8). У цьому HTML-коді створено сторінку з назвою "Game Room", яка складається з різноманітних елементів, стилізованих за допомогою Bootstrap та власних CSS-стилів. Далі було описано значну частину файлу. За необхідності можна переглянути код з коментарями у додатку Д.

Спочатку використовується {% include 'header.html' %} для включення заголовку, що містить у собі навігаційне меню, логотип та інші елементи.

У тілі документу знаходиться посилання на CSS-файл Bootstrap для застосування стилів. Додатково визначаються власні CSS-стилі для елементів, які використовуються на сторінці. Наприклад, класи .card-container, .card, .rounded-xxl визначають розташування та вигляд карток, які відображаються на сторінці. Клас .show-button задає стилі для кнопок.

```
<style>
    .card-container {
        display: flex;
        flex-wrap: wrap;
    }
    .card {
        flex: 1;
        margin: 10px;
    }
    .rounded-xxl {
        border-radius: 1.5rem;
    }
    .show-button{
        background-color: #0D0D0D;
        width: 60px;
        height: 50px;
    }
</style>
```

Рисунок 3.8 – Перша частина файлу room_details.html

Далі в середи тегу body файлу починається опис головної сторінки проекту (рис 3.9). Перша частина коду відображає список місць з можливістю резервування, де кожне' місце може бути зайнятим гравцем або залишитися вільним, і відображає кнопку для резервування місця, якщо воно вільне.

```
<body class="text-white " style="background-color: #0D0D0D;">
    <div class="container mt-5">
        {% if room.game_started == False %}
            <ul class="list-group">
                {% for place in places %}
                <li id="place{{ place.id }}" class="list-group-item d-flex justify-content-between align-items-center bg-dark border-white border-top ">
                    <div class="">
                        {% if place.player_name %}
                            <span>{{ place.player_name }}</span>
                        {% else %}
                            <span class="text-muted">Вільне місце</span>
                        {% endif %}
                        <span class="badge badge-primary badge-pill">Place №{{ forloop.counter }}</span>
                    </div>
                    {% if not place.player_name %}
                        <button class="btn btn-primary" onclick="takePlace({{ place.id }})">Take a Place</button>
                    {% endif %}
                </li>
                {% endfor %}
            </ul>
        {% endif %}
```

Рисунок 3.9 – Друга частина файлу room_details.html

Після чого зберігається код, що відображає список повідомлень, які прописані в файлі views.py, якщо такі є. Використовуючи класи Bootstrap, він стилізує ці повідомлення, надаючи їм різні кольори, форми та розміри відповідно до їх тегів (рис 3.10).

```
{% if messages %}
    <ul class="messages mt-3">
        {% for message in messages %}
            <li class="alert alert-{{ message.tags }}">{{ message }}</li>
        {% endfor %}
    </ul>
{% endif %}
```

Рисунок 3.10 – Третя частина файлу room_details.html

Наступний код відображає повідомлення про завершення гри, якщо гра вже завершилась. Текст цього повідомлення пояснює ситуацію, що виникла як результат гри та пояснює, яке майбутнє чекає гравців та їх персонажів та які умови для успішного виживання потрібні (рис 3.11).

```
{% if room.game_finished %}
    <div class="alert alert-warning mt-3" role="alert">
        The players who remained outside the bunker survived and decided to take revenge on those who got into the bunker. In
    </div>
{% endif %}
```

Рисунок 3.11 – Четверта частина файлу room_details.html

Наступний код розміщує текстове повідомлення, що відображає поточного гравця, якщо гра розпочалась і ще не завершилась (рис 3.12).

```
<div class="ml-5 d-flex justify-content-center">
    {% if room.game_started == True and room.game_finished == False%}
        <p id="currentTurnPlayer" class="display-4 font-weight-bold mb-5">Current player: {{ room.current_turn_player.username }}</p>
    {% endif %}
</div>
```

Рисунок 3.12 – П'ята частина файлу room_details.html

Далі в HTML-коді створено форму для голосування за гравців у віртуальній кімнаті гри (рис 3.13). Відбираються гравці, за яких можна проголосувати, враховуючи статус гри та рішення про закінчення ходу.

Форма розташована всередині блоку з класом "ml-5", що відступає від лівого краю. Умовна конструкція `{% if room.voting_started and room.game_finished == False %}` перевіряє, чи розпочалося голосування та чи не завершено гру.

У разі відповідності умові, форма відображається. Вона містить список вибору для вибору гравця, якого хочуть проголосувати. Якщо гравець проголосував, кнопка "Vote" активна (не відключена).

Цей фрагмент HTML дозволяє гравцям вибирати інших гравців для голосування в межах правил гри, якщо голосування є можливим на даний момент.

```
{% if room.voting_started and room.game_finished == False %}
    <form id="voteForm" method="post" action="{% url 'vote_endpoint' room_id=room.id %}">
        {% csrf_token %}
        <input type="hidden" name="room_id" value="{{ room.id }}">
        <div class="form-group">
            <label for="selected_player_name">Choose player:</label>
            <select id="selected_player_name" name="selected_player_name" class="form-control" style="...">
                {% if room.turn_ended and room.voting_started %}
                    {% for player in players %}
                        {% if player != user %}
                            <option value="{{ player.username }}">{{ player.username }}</option>
                        {% endif %}
                    {% endfor %}
                {% endif %}
            </select>
        </div>
        <button type="submit" class="btn btn-primary mt-2" {% if not room.turn_ended and not room.voting_started %} disabled
    </form>
{% endif %}
iv>
```

Рисунок 3.13 – Шоста частина файлу room_details.html

Код на рисунку (рис 3.14) відображає картки персонажів для кожного місця у кімнаті, якщо гра розпочалась і гравець на цьому місці ще не вигнаний з кімнати (`is_kicked == False`). Картки персонажів мають на увазі набір характеристик, що згруповані у одного гравця. В коді описано, що при умові ,

що гравець є власником цих характеристик, і зараз його хід то він може натиснути кнопку show біля кожної з характеристик, щоб інші гравці змогли побачити їх у себе на екрані. Поки він цього не зробить у них замість характеристик цього гравця відображається текст hidden.

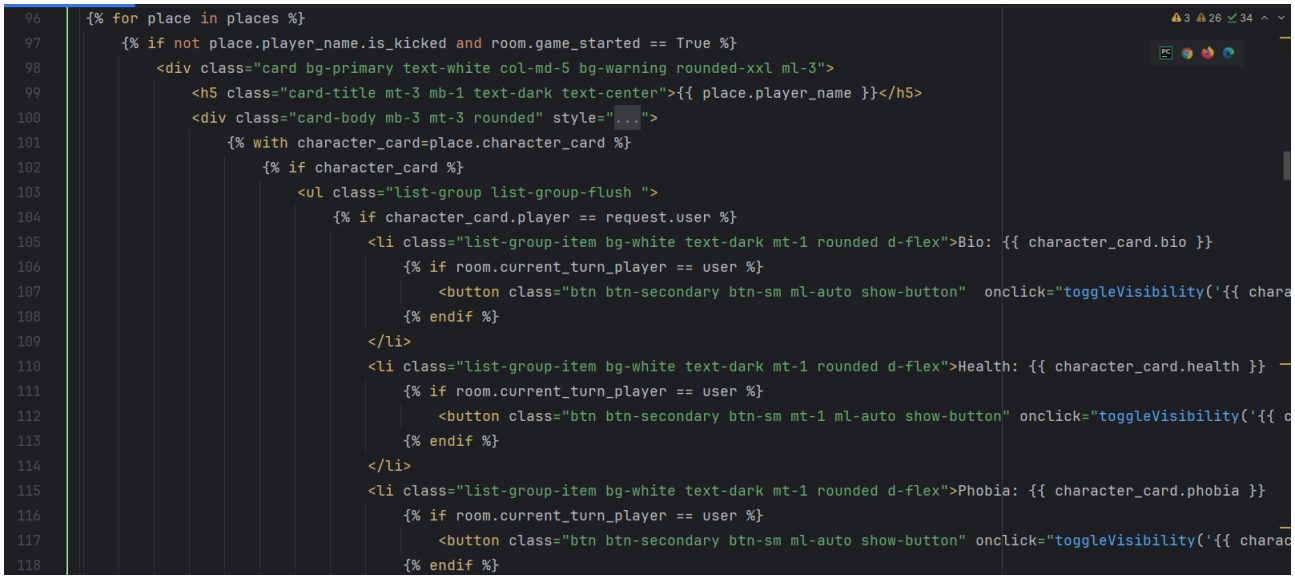


Рисунок 3.14 – Сьома частина файлу room_details.html

На наступному зображенні (рис 3.15) інформація представлена у вигляді списку з певними характеристиками персонажа, такими як біографія, здоров'я, фобії, хобі, знання, додаткова інформація та багаж. Кожна характеристика персонажа відображається як окремий пункт списку з відповідними значеннями. Якщо певна характеристика персонажа прихована (наприклад, біографія або здоров'я), вона замінюється текстом "Hidden".

Умова `{% if not character_card %}` перевіряє, чи існує картка персонажа для гравця. Якщо така картка існує, то відображаються всі характеристики персонажа. Якщо картка персонажа відсутня, то виводиться повідомлення про її відсутність у гравця.

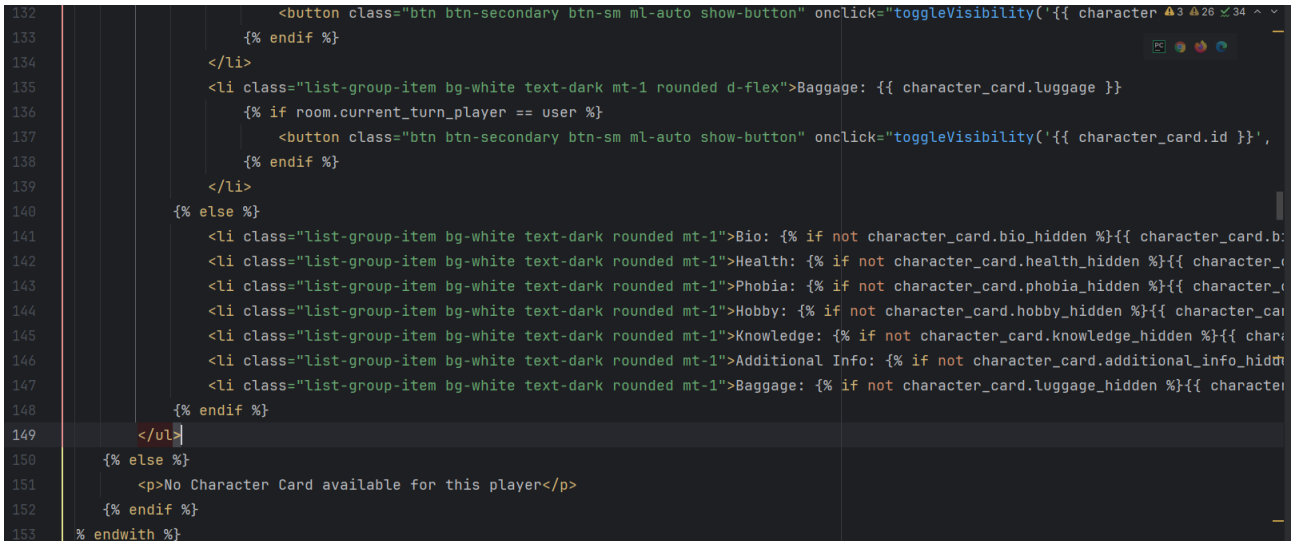


Рисунок 3.15 – Восьма частина файлу room_details.html

У коді, що зображений (рис 3.16) знаходиться кнопка закінчення ходу гравця. Яка доступна у випадку, якщо він не завершив хід і якщо у нього є можливість завершити хід. Окрім цього вона доступна тільки якщо гра не завершена та користувач є тим гравцем кому належить хід. Також згідно умов ще не має бути почато голосування і хід не має бути закінчений. Проте якщо гра ще не почалась користувач отримає повідомлення про те що гра ще не почалась і що потрібно почекати, поки власник кімнати почне гру.

Рисунок 3.16 – Дев’ята частина файлу room_details.html

Після функціоналу з закінчення ходу, знаходить код з логікою про отримання кількості голосів за гравців (рис 3.17). При натисканні на кнопку “Get

Voting Result” Користувач отримує список з гравців, та кількість голосів за них. Користувачам ця кнопка доступна одразу після того як почалось голосування в кімнаті.

```
{% if room.voting_started == True %}
    <div class="container mt-3">
        <div class="row">
            <div class="col">
                <ul id="voteResultsList" class="list-group">
                </ul>
            </div>
        </div>
        <div class="row mt-3 mb-5">
            <div class="col">
                <button onclick="getVoteResults({{ room.id }})" class="btn btn-primary">Get Voting Results</button>
            </div>
        </div>
    </div>
{% endif %}
```

Рисунок 3.17 – Десята частина файлу room_details.html

У наступному фрагменті HTML-коду створено кнопку для видалення кімнати гри (рис 3.18). Кнопка буде відображатися тільки для аутентифікованих користувачів, які є творцем кімнати.

Елемент `<div>` з класами "ml-4", "mt-4" та "mb-4" використовується для вирівнювання кнопки у відповідному місці на сторінці. Умова `{% if user.is_authenticated and room.creator == user %}` перевіряє, чи користувач аутентифікований і чи він є творцем кімнати.

Якщо умова виконується, тобто користувач аутентифікований і він є творцем кімнати, то відображається кнопка з червоним кольором фону "Delete Room". При натисканні на цю кнопку викликається JavaScript-функція `deleteRoom()`, яка передає ідентифікатор кімнати (room.id) для подальшої обробки.

Функція дуже корисна оскільки користувачі постійно будуть створювати нові кімнати де будуть відбуватися ігри, а старі відповідно потрібно постійно видаляти.

```
<div class="ml-4 mt-4 mb-4 ">
    {% if user.is_authenticated and room.creator == user %}
        <button class="btn btn-danger mt-3" onclick="deleteRoom({{ room.id }})">Delete Room</button>
    {% endif %}
</div>
```

Рисунок 3.18 – Одинадцята частина файлу room_details.html

На цьому основний html код закінчений. Тепер у файлі room_details.html залишився тільки JavaScript код.

```
function start_game() {
            var xhr = new XMLHttpRequest();
            xhr.onload = function() {
                if (xhr.status === 200) {
                    window.location.reload();
                } else {
                    alert('Помилка: ' + xhr.statusText);
                }
            };
            xhr.open('POST', '{% url "start_game" room.id %}',
true);
            xhr.setRequestHeader('X-CSRFToken', '{{ csrf_token
}}');
            xhr.send();
        }
```

Цей код допомагає з функціоналом початку гри.

Функція “start_game”:

- Створює запит AJAX для запуску гри.

- Використовує `XMLHttpRequest` для надсилання POST запиту на сервер на URL, отриманий через Django `{% url "start_game" room.id %}`.

- Встановлює заголовок CSRF-токена для захисту від CSRF-атак.

- Після успішного завершення запиту (`xhr.status === 200`), сторінка перезавантажується. Якщо запит не вдається, відображається повідомлення про помилку у грі.

```
setTimeout(function() {
                var messages = document.querySelectorAll('.messages
li');
                if (messages.length > 0) {
                    messages[0].remove();
                }
            }, 5000);
```

Цей фрагмент JavaScript-коду встановлює таймер, який через 5 секунд (5000 мілісекунд) виконує анонімну функцію. Ця функція вибирає всі елементи списку (`li`) з класом `.messages` за допомогою `document.querySelectorAll`. Якщо виявляється хоча б один елемент, перший елемент видаляється з DOM за допомогою методу `remove()`.

```
            $.ajax({
                type: "POST",
                url: "/vote_endpoint/",
                data: {
                    selected_player_name: selectedPlayerName,
                    room_id: roomId,
                    csrfmiddlewaretoken: csrftoken
                },
```

Цей фрагмент JavaScript-коду використовує jQuery для відправки асинхронного POST-запиту на сервер за допомогою методу `$.ajax`. Він відправляє запит на URL `/vote_endpoint/` з даними, що містять ім'я вибраного гравця (`selectedPlayerName`), ідентифікатор кімнати (`roomId`), і CSRF-токен для захисту від CSRF-атак (`csrftoken`).

```
            success: function(response) {
                    if (response.message) {
                        Swal.fire({
```

```
                        title: 'Голосування завершено',
                        text: response.message,
                        icon: 'success'
                    });
                }
            },
            error: function(xhr, status, error) {
                console.error(error);
            }
        });
```

Цей фрагмент JavaScript-коду продовжує визначати поведінку після успішного або невдалого виконання асинхронного POST-запиту.

Якщо запит успішний і сервер повертає відповідь, що містить повідомлення (`response.message`), викликається функція `Swal.fire` з бібліотеки SweetAlert для відображення спливаючого вікна з заголовком "Голосування завершено", текстом повідомлення з відповіді сервера і значком успіху.

Якщо запит не вдається, у консоль браузера виводиться повідомлення про помилку за допомогою `console.error`.

```
document.addEventListener("DOMContentLoaded", function() {
        var roomId = document.getElementById("roomId").value;
        getVoteResults(roomId);
    });
    function getVoteResults(roomId) {
        $.ajax({
            type: "GET",
            url: /get_vote_results/${roomId}/,
            success: function(response) {
                displayVoteResults(response.voteCounts);
            },
            error: function(xhr, status, error) {
                console.error(error);
            }
        });
```

```
        }
```

Цей код ініціює запит для отримання результатів голосування після завантаження сторінки та відображає ці результати за допомогою функції. `displayVoteResults`, якщо запит успішний.

```
function displayVoteResults(voteCounts) {
            var voteResultsList =
document.getElementById("voteResultsList");
            voteResultsList.innerHTML = '';
            voteCounts.forEach(function(item) {
                var listItem = document.createElement('li');
                listItem.textContent = ${item.target_player__username}:
${item.total_votes};
                voteResultsList.appendChild(listItem);
            });
            document.getElementById("voteResultsSection").style.display =
"block";
        }
```

Наступна функція бере результати голосування, створює новий елемент списку для кожного результату і додає їх до списку на сторінці, а також робить видимою секцію з результатами голосування

```
function vote() {
            var selectedPlayerName = $("#selected_player_name").val();
            if (!selectedPlayerName) {
                alert("Будь ласка, виберіть гравця для голосування.");
                return;
            }
            var roomId = $("#voteForm input[name='room_id']").val();
            var url = "{% url 'vote_endpoint' room_id=room.id %}";
```

Ця функція `vote` збирає дані з форми голосування і надсилає їх на сервер для обробки.

Спочатку отримується ім'я вибраного гравця з елемента з ID `selected_player_name`. Якщо ім'я не вказане, показується попередження і функція завершується. Потім отримується ідентифікатор кімнати з прихованого поля форми з ім'ям `room_id`. Нарешті, формується URL для відправки даних на сервер, використовуючи тег Django template `{% url 'vote_endpoint' room_id=room.id %}`, який генерує URL для вказаного ендпоінту з відповідним ідентифікатором кімнати.

```
$.ajax({
          type: "POST",
          url: url,
          data: {
              'selected_player_name': selectedPlayerName,
              'room_id': roomId,
              'csrfmiddlewaretoken': '{{ csrf_token }}'
          },
          success: function(response) {
              alert(response.message);
          },
          error: function(xhr, status, error) {
              console.error(error);
          }
      });
  }
```

Цей фрагмент JavaScript-коду відправляє асинхронний POST-запит на сервер за допомогою методу `$.ajax`. Він використовує отримані раніше дані (ім'я вибраного гравця та ідентифікатор кімнати) для встановлення URL-адреси, до якої буде відправлений запит.

Дані для відправки передаються у вигляді об'єкта у властивості `data`. Цей об'єкт містить ім'я вибраного гравця, ідентифікатор кімнати та CSRF-токен для захисту від CSRF-атак.

Якщо запит успішно виконується, виводиться повідомлення з відповіді сервера. Якщо виникає помилка, у консоль браузера виводиться повідомлення про помилку.

```
var endTurnButtons = document.querySelectorAll('.end-turn-button');
    endTurnButtons.forEach(function(button) {
        button.addEventListener('click', handleEndTurnClick);
    });
```

Цей код отримує кнопку з класом “end-turn-button” на сторінці і додає до неї обробник подій кліку. Коли кнопка буде натиснута, викликається функція “handleEndTurnClick”, яка обробляє завершення ходу.

```
function endTurn(rooId) {
            var csrfToken =
document.querySelector('[name=csrfmiddlewaretoken]').value;
            fetch(/end_turn/${roomId}/, {
                method: 'POST',
                headers: {
                    'X-CSRFToken': csrfToken,
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body: ''
            })
```

Ця функція `endTurn` призначена для завершення поточного раунду гри шляхом відправлення POST-запиту на сервер.

Спочатку вона отримує CSRF-токен з мета-тега на сторінці. Потім використовується функція `fetch`, щоб виконати POST-запит на URL `/end_turn/${roomId}/`.

У хедерах запиту встановлюється CSRF-токен та тип контенту `application/x-www-form-urlencoded`. Пустий рядок передається як тіло запиту.

```
            .then(response => {
                if (response.ok) {
                    var currentTurnButton = document.querySelector('.end-
turn-button[data-room-id="' + roomId + '"]');
                    currentTurnButton.disabled = true;
                    var nextPlayerButton = getNextPlayerButton();
                    if (nextPlayerButton) {
                        nextPlayerButton.disabled = false;
                    }
                    location.reload();
```

Цей фрагмент JavaScript-коду виконує додаткові дії після успішного завершення POST-запиту. Якщо відповідь від сервера успішна, він вимикає кнопку поточного ходу, активує кнопку для наступного гравця (якщо така існує), та перезавантажує сторінку для оновлення даних.

```
                } else {
                    alert('Failed to end turn');
                }
            })
            .catch(error => console.error('Error:', error));
    }
```

Якщо відповідь від сервера не успішна (наприклад, статус відповіді не `200 OK`), виводиться повідомлення про невдале завершення ходу. Якщо виникає помилка під час виконання запиту (наприклад, втрата з'єднання з сервером), виводиться помилка в консоль браузера для подальшого аналізу.

```
function getNextPlayerButton() {
            var endTurnButtons = document.querySelectorAll('.end-turn-
button');
            for (var i = 0; i < endTurnButtons.length; i++) {
                if (!endTurnButtons[i].disabled) {
```

```
                return endTurnButtons[i + 1];
            }
        }
        return null;
    }
```

Функція “getNextPlayerButton” знаходить першу активну кнопку "завершити хід" у списку кнопок. Якщо така кнопка знайдена, функція повертає наступну кнопку після неї, інакше повертає “null”.

```
function toggleVisibility(characterCardId, characteristic) {
        var csrfToken =
document.querySelector('[name=csrfmiddlewaretoken]').value;

fetch(/toggle_visibility/${characterCardId}/${characteristic}/, {
            method: 'POST',
            headers: {
                'X-CSRFToken': csrfToken,
                'Content-Type': 'application/x-www-form-urlencoded',
            },
            body: ''
        })
```

Ця функція `toggleVisibility` призначена для зміни видимості певної характеристики на карті персонажа шляхом відправки POST-запиту на сервер.

Спочатку вона отримує CSRF-токен з мета-тега на сторінці. Потім використовується функція `fetch`, щоб виконати POST-запит на URL `/toggle_visibility/${characterCardId}/${characteristic}/`.

У хедерах запиту встановлюється CSRF-токен та тип контенту `application/x-www-form-urlencoded`. Пустий рядок передається як тіло запиту.

```
        .then(response => {
            if (response.ok) {
                location.reload();
            } else {
```

```
                alert('Failed to toggle visibility');
            }
        })
        .catch(error => console.error('Error:', error));
}
```

Після відправлення запиту на сервер цей код перевіряє, чи відповідь успішна (статус `200 OK`). Якщо так, сторінка перезавантажується для оновлення даних. У разі невдачі виводиться повідомлення про невдалу зміну видимості. Якщо виникає помилка під час виконання запиту, вона виводиться в консоль браузера для подальшого аналізу.

```
function revealCharacteristics() {
      var selectedPlayer = document.getElementById("playerSelect").value;
      var characteristicsDiv =
document.getElementById("characteristics");
      if (selectedPlayer === "player1") {
        characteristicsDiv.hidden = false;
      } else {
        characteristicsDiv.hidden = true;
      }
    }
```

Функція `revealCharacteristics` відповідає за відкриття або приховання характеристик обраного гравця на сторінці.

Вона спочатку отримує значення, обране вибором гравця з елементу з ідентифікатором "playerSelect". Потім отримує доступ до блоку характеристик за допомогою ідентифікатора "characteristics". Якщо обраний гравець є "player1", то блок характеристик стає видимим (параметр hidden встановлюється в false). В іншому випадку, блок характеристик приховується (параметр hidden встановлюється в true).

```
    function takePlace(placeId) {
```

```
    fetch(/take_place/{{ room_id }}/${placeId}/)
        .then(response => response.json())
        .then(data => {
            if (data.error) {
                alert(data.error);
```

Ця функція `takePlace` призначена для взяття місця гравця за допомогою відправки запиту на сервер. Вона виконує асинхронний запит до сервера за допомогою функції `fetch`, передаючи ідентифікатор місця (`placeId`) як параметр запиту. Отримана відповідь перетворюється в об'єкт JSON за допомогою методу `response.json()`. Якщо відповідь містить поле `error`, виводиться відповідне повідомлення за допомогою функції `alert`.

```
      } else {
                const placeElement =
document.getElementById(place${placeId});
                const playerPlaceText =
document.createTextNode(${data.player_name} Place №${data.place_number});
                placeElement.innerHTML = '';
                placeElement.appendChild(playerPlaceText);
            }
        })
        .catch(error => console.error('Error:', error));
}
```

У цьому фрагменті коду перевіряється відповідь від сервера після виконання запиту на взяття місця гравця. Якщо відповідь успішна (без поля "error"), встановлюється вміст елементу місця з ідентифікатором, що відповідає `placeId`, на основі ім'я гравця та номера місця. У випадку невдачі виводиться повідомлення про помилку у консоль браузера.

```
function deleteRoom(roomId) {
        fetch(/delete_room/${roomId}/, {
```

```
            method: 'DELETE',
            headers: {
                'X-CSRFToken': '{{ csrf_token }}',
            }
        })
        .then(response => {
            if (response.ok) {
                window.location.href = '/';
```

Функція `deleteRoom` призначена для видалення кімнати з сервера за допомогою HTTP-запиту методом DELETE. Вона виконує асинхронний запит до сервера за допомогою функції `fetch`, передаючи ідентифікатор кімнати (`roomId`) у URL. У хедерах запиту вказується CSRF-токен для захисту від CSRF-атак. Після успішного видалення кімнати (якщо відповідь сервера має статус `200 OK`), відбувається перенаправлення на головну сторінку сайту.

```
        } else {
                alert('Failed to delete room');
            }
        })
        .catch(error => console.error('Error:', error));
}
```

У цьому фрагменті коду перевіряється відповідь від сервера після виконання запиту на видалення кімнати.

Якщо відповідь успішна (статус `200 OK`), перенаправляється на головну сторінку сайту. У випадку невдачі видалення кімнати виводиться повідомлення про помилку за допомогою `alert`.

В іншому випадку, якщо виникає помилка під час виконання запиту, вона виводиться в консоль браузера для подальшого аналізу.

3.3.4 Додаткові файли.

Один із найважливіших файлів в роботі проекту – це файл urls.py. Він важливий оскільки в ньому зберігаються всі шляхи, які існують в проекті.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name='index'),
    path('register/', views.register, name='register'),
    path('login/', views.user_login, name='login'),
    path('room/<int:room_id>/', views.room_detail, name='room_detail'),
    path('take_place/<int:room_id>/<int:place_id>/', views.take_place,
name='take_place'),
    path('delete_room/<int:room_id>/', views.delete_room,
name='delete_room'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
```

Цей фрагмент коду визначає URL-шаблони для різних сторінок веб-додатку, включаючи адміністративний розділ, сторінку реєстрації, вхід та вихід користувачів, деталі кімнати, взяття місця гравцем та видалення кімнати.

```
    path('start_game/<int:room_id>/', views.start_game,
name='start_game'),

path('toggle_visibility/<int:character_card_id>/<str:characteristic>/',
views.toggle_visibility, name='toggle_visibility'),
    path('end_turn/<int:room_id>/', views.endTurn, name='end_turn'),
    path('vote/<int:room_id>/', views.vote_endpoint,
name='vote_endpoint'),
    path('get_vote_results/<int:room_id>/', views.get_vote_results,
name='get_vote_results'),
    path('rules/', views.rules, name='rules'),
    path('contacts/', views.contacts, name='contacts'),
]
```

Цей фрагмент коду визначає URL-шаблони для різних дій у вашому веб-додатку, таких як початок гри, зміна видимості характеристик, завершення раунду, голосування, отримання результатів голосування, а також сторінки з правилами гри та контактною інформацією.

Окрім цього файлу також, був створений файл utils.py, в якому зберігаються варіанти характеристики для гравців. Далі було описано тільки частину файлу. Переглянути повний код можна у додатку В.

```
BIO_OPTIONS = [
    "Man, 35 years old, unable to give birth.",
]
HEALTH_OPTIONS = [
    "Headache",
]
PHOBIA_OPTIONS = [
    "Fear of heights",
]
```

Цей фрагмент коду містить списки з різними варіантами характеристик: BIO_OPTIONS - характеристики з описом особистості, HEALTH_OPTIONS - характеристики щодо стану здоров'я та PHOBIA_OPTIONS - список фобій.

```
HOBBY_OPTIONS = [
    "Painting",
]
KNOWLEDGE_OPTIONS = [
    "Know how to purify water",
]
ADDITIONAL_INFO_OPTIONS = [
    "Speak multiple languages fluently",
]
LUGGAGE_OPTIONS = [
```

```
    "Seeds",
]
```

Далі в коді прописані списки з різними варіантами характеристик: HOBBY_OPTIONS - захоплення, KNOWLEDGE_OPTIONS - знання, ADDITIONAL_INFO_OPTIONS - додаткова інформація, та LUGGAGE_OPTIONS - предмети або речі, які персонаж має з собою.

Окрім цього у файлі settings.py були додані наступні два рядки
ALLOWED_HOSTS = ['localhost', '127.0.0.1', 'd9f3-176-117-186-27.ngrok-free.app']
CSRF_TRUSTED_ORIGINS = ['https://d9f3-176-117-186-27.ngrok-free.app']
Для того, щоб розмістити сайт в мережі інтернет, та мати доступ до нього з будь якого місця з доступом до інтернету [10].

Висновок до розділу 3

На основі проробленої роботи в розділі 3 було створено функціональний продукт, який надає користувачам можливість з мінімальними зусиллями насолоджуватися грою в мережі онлайн. Розроблено зручний інтерфейс, де користувачі можуть легко та зрозуміло отримати те, за чим вони зайшли на сайт.

ВИСНОВОК

У цій кваліфікаційній роботі було розроблено онлайн-платформу для гри в настільну гру "Bunker". Метою дослідження було створення зручного та функціонального веб-сайту, який дозволяє гравцям насолоджуватися грою в онлайн-режимі. Для досягнення цієї мети було проведено аналіз існуючих рішень, визначено їхні недоліки та запропоновано нові підходи до реалізації платформи в мережі інтернет.

На початковому етапі роботи було проведено детальний аналіз літератури та існуючих досліджень у галузі розробки онлайн-ігор. Це дозволило виявити аспекти та проблеми, які потребували подальшого вивчення. На основі цього аналізу було сформульовано функціональні вимоги до майбутньої платформи.

У процесі розробки платформи використовувалися сучасні технології, такі як Python, Django, JavaScript, HTML, CSS та Bootstrap. Це дозволило створити продукт з інтуїтивно зрозумілим інтерфейсом та високою продуктивністю. Особлива увага приділялася забезпеченню інтерактивності платформи, що дозволяє користувачам взаємодіяти з грою у режимі реального часу.

Розроблена платформа має ряд переваг. Вона зручна у використанні і може бути легко розширена новими функціями. Це робить її привабливою для широкого кола гравців, які можуть насолоджуватися грою незалежно від їхнього місцезнаходження. Платформа також забезпечує високу продуктивність та стабільність, що є важливими характеристиками для онлайн-ігор.

Практичне значення отриманих результатів полягає у можливості використання розробленої платформи як основи для створення інших інноваційних продуктів та покращення існуючих ігрових платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CSS Tricks. URL: https://css-tricks.com/ (дата звернення: 24.03.2024).

2. Django Documentation. URL: https://docs.djangoproject.com/en/stable/(дата звернення: 16.01.2024).

3. Django Packages. URL: https://djangopackages.org/ (дата звернення: 24.05.2024).

4. Етапи життєвого циклу розробки ПЗ. URL: https://icstudio.online/post/etapi-zhittyevogo-ciklu-rozrobki-pz (дата звернення: 02.03.2023).

5. Git Documentation. URL: https://git-scm.com/doc (дата звернення: 24.05.2024).

6. HTML Documentation. URL: https://html.spec.whatwg.org/ (дата звернення: 04.04.2024).

7. JavaScript.info. URL: https://javascript.info/ (дата звернення: 24.05.2024).

8. Кнут Д. Мистецтво програмування. Том 1. Основні алгоритми. – М.: Вільямс, 2001. – 672 с.

9. MDN Web Docs. URL: https://developer.mozilla.org/ (дата звернення: 18.03.2024).

10. Ngrok Documentation. URL: https://ngrok.com/docs (дата звернення: 04.04.2024).

11. Platform basing Programing. Model-View-Template. URL: https://pbp-fasilkom-ui.github.io/ganjil-2023/en/assignments/tutorial/tutorial-1/ (дата звернення: 23.01.2024).

12. PyCharm Features вебсайт. URL: https://www.jetbrains.com/pycharm/features/ (дата звернення: 01.03.2023).

13. Python Advantages and Disadvantages – Step in the right direction вебсайт. URL: https://techvidvan.com/tutorials/python-advantages-and-disadvantages/ (дата звернення: 27.02.2023).

14. Python Documentation. URL: https://docs.python.org/3/ (дата звернення: 15.01.2024).

15. Python Package Index (PyPI). URL: https://pypi.org/ (дата звернення: 29.01.2024).

16. Python вебсайт. URL: https://www.python.org/ (дата звернення: 28.02.2023).

17. Real Python: Python Tutorials. URL: https://realpython.com/ (дата звернення: 01.03.2024).

18. SQLite3 Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 24.05.2024).

19. The Python Tutorial. URL: https://docs.python.org/3/tutorial/ (дата звернення: 16.03.2024).

20. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/ (дата звернення: 23.03.2024).

ДОДАТКИ

Додаток А

Файл models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.conf import settings
from .utils import BIO_OPTIONS, HEALTH_OPTIONS, PHOBIA_OPTIONS,
HOBBY_OPTIONS, KNOWLEDGE_OPTIONS, \
    ADDITIONAL_INFO_OPTIONS, LUGGAGE_OPTIONS
from django.db.models.signals import pre_delete
from django.dispatch import receiver


class Room(models.Model):
    initial_players_count = models.PositiveIntegerField(default=0)
    name = models.CharField(max_length=100)
    char_by_turn = models.IntegerField(default=1)
    max_players = models.IntegerField()
    password = models.CharField(max_length=100, blank=True, null=True)
    creator = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='created_rooms')
    current_player = models.ForeignKey(User, on_delete=models.SET_NULL,
null=True, blank=True,

related_name='current_player_rooms')
    game_started = models.BooleanField(default=False)
    current_turn_player = models.ForeignKey(User,
on_delete=models.SET_NULL, null=True, blank=True,

related_name='current_turn_player_rooms')
    turn_ended = models.BooleanField(default=False)
    voting_started = models.BooleanField(default=False)
    game_finished = models.BooleanField(default=False)
```

```
    def str(self):
        return self.name


@receiver(pre_delete, sender=Room)
def delete_character_cards(sender, instance, **kwargs):
    places = instance.place_set.all()

    character_cards = CharacterCard.objects.filter(place__room=instance)
    character_cards.delete()


class CharacterCard(models.Model):
    characteristic_opened = models.BooleanField(default=False)
    player        =        models.OneToOneField(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE)
    bio = models.TextField(blank=True)
    health = models.CharField(blank=True, max_length=100)
    phobia = models.CharField(blank=True, max_length=100)
    hobby = models.CharField(blank=True, max_length=100)
    knowledge = models.CharField(blank=True, max_length=100)
    additional_info = models.TextField(blank=True)
    luggage = models.TextField(blank=True)

    bio_hidden = models.BooleanField(default=True)
    health_hidden = models.BooleanField(default=True)
    phobia_hidden = models.BooleanField(default=True)
    hobby_hidden = models.BooleanField(default=True)
    knowledge_hidden = models.BooleanField(default=True)
    additional_info_hidden = models.BooleanField(default=True)
    luggage_hidden = models.BooleanField(default=True)

    BIO_OPTIONS = BIO_OPTIONS
    HEALTH_OPTIONS = HEALTH_OPTIONS
    PHOBIA_OPTIONS = PHOBIA_OPTIONS
    HOBBY_OPTIONS = HOBBY_OPTIONS
    KNOWLEDGE_OPTIONS = KNOWLEDGE_OPTIONS
```

```
    ADDITIONAL_INFO_OPTIONS = ADDITIONAL_INFO_OPTIONS
    LUGGAGE_OPTIONS = LUGGAGE_OPTIONS

class Place(models.Model):
    room = models.ForeignKey('Room', on_delete=models.CASCADE)
    player_name = models.CharField(max_length=100, blank=True, null=True)
    character_card          =          models.ForeignKey('CharacterCard',
on_delete=models.SET_NULL, null=True, blank=True)
    turn_finished = models.BooleanField(default=False)
    can_end_turn = models.BooleanField(default=True)
    voted = models.BooleanField(default=False)
    is_kicked = models.BooleanField(default=False)

    def str(self):
        return f'Character Card for {self.player_name}'


class Vote(models.Model):
    voter = models.ForeignKey(User, on_delete=models.CASCADE)
    target_player = models.ForeignKey(User, on_delete=models.CASCADE,
                                      related_name='received_votes'
    room = models.ForeignKey(Room, on_delete=models.CASCADE
    created_at = models.DateTimeField(auto_now_add=True
    def str(self):
        return      f"Vote      from      {self.voter.username}      to
{self.target_player.username} in room {self.room.id}"
```

Додаток Б

Файл views.py

```
def end_game(room):
    room.game_finished = True
    room.save()
    return JsonResponse({'error': 'Game finished'}, status=200)
def start_new_turn(room_id):
    room = get_object_or_404(Room, id=room_id)
    players = room.place_set.all()
    initial_players_count = room.place_set.count()
    # Підраховуємо кількість живих гравців
    alive_players_count = players.filter(is_kicked=False).count()

    if alive_players_count == initial_players_count // 2 or
alive_players_count == (initial_players_count // 2) + 1:
        # Якщо так, гра закінчується
        end_game(room)
    else:
        places_in_room = Place.objects.filter(room=room)

        # Вибрати першого гравця в кімнаті для початку нового ходу
        first_place = places_in_room.filter(is_kicked=False).first()  #
Фільтруємо за is_kicked
        if first_place:
            first_player                                              =
User.objects.get(username=first_place.player_name)
            room.current_turn_player = first_player
            room.save()

            for place in places_in_room:
                if not place.is_kicked:
                    place.turn_finished = False
                    place.can_end_turn = True
                    place.voted = False
                    place.save()
```

```
            room.game_started = True
            room.voting_started = False
            room.turn_ended = False
            room.save()

            return JsonResponse({'message': 'New turn'})


def calculate_votes(room_id):
    room = Room.objects.get(id=room_id)
    all_votes = Vote.objects.filter(room=room)
    vote_counter = Counter(vote.target_player for vote in all_votes)
    return vote_counter


def determine_winner(room_id):
    vote_counter = calculate_votes(room_id)
    if vote_counter:
        max_votes = max(vote_counter.values())
        winners = [player for player, votes in vote_counter.items() if
votes == max_votes]
        return winners
    else:
        return None


def get_vote_results(request, room_id):
    # Отримати кількість голосів за кожного гравця у вказаній кімнаті
    vote_counts = (
        Vote.objects.filter(room_id=room_id)
        .values('target_player__username')  # Групувати за іменем гравця,
за якого проголосували
        .annotate(total_votes=Count('id'))  # Підрахунок кількості голосів
за кожного гравця
    )

    # Перетворюємо QuerySet в список словників
```

```
    vote_counts_list = list(vote_counts)

    # Повертаємо результат у форматі JSON
    return JsonResponse({'voteCounts': vote_counts_list},
encoder=DjangoJSONEncoder)


@login_required
def toggle_visibility(request, character_card_id, characteristic):
    # Отримати характеристику по її ідентифікатору
    try:
        character_card = CharacterCard.objects.get(id=character_card_id)
    except CharacterCard.DoesNotExist:
        return JsonResponse({'error': 'Character card not found'},
status=404)

    # Перевірити, чи користувач є власником цієї характеристики
    if character_card.player != request.user:
        return JsonResponse({'error': 'You are not the owner of this
character card'}, status=403)

# Змінити видимість характеристики згідно з параметром
    if characteristic == 'bio':
        character_card.bio_hidden = not character_card.bio_hidden
    elif characteristic == 'health':
        character_card.health_hidden = not character_card.health_hidden
    elif characteristic == 'phobia':
        character_card.phobia_hidden = not character_card.phobia_hidden
    elif characteristic == 'hobby':
        character_card.hobby_hidden = not character_card.hobby_hidden
    elif characteristic == 'knowledge':
        character_card.knowledge_hidden = not
character_card.knowledge_hidden
    elif characteristic == 'additional_info':
        character_card.additional_info_hidden = not
character_card.additional_info_hidden
    elif characteristic == 'luggage':
```

```
        character_card.luggage_hidden = not character_card.luggage_hidden
    else:
        return    JsonResponse({'error':    'Invalid    characteristic'},
status=400)

    character_card.characteristic_opened = True
    character_card.save()

    return JsonResponse({'success': 'Visibility toggled successfully'})
def rules(request):
    return render(request, 'rules.html')
def contacts(request):
    return render(request, 'contacts.html')
```

Додаток В

Файл utils.py

```
BIO_OPTIONS = [
    "Woman, 28 years old, able to give birth.",
    "Man, 35 years old, unable to give birth.",
    "Woman, 32 years old, able to give birth.",
    "Man, 40 years old, unable to give birth.",
    "Woman, 25 years old, able to give birth.",
    "Man, 45 years old, unable to give birth.",
    "Woman, 22 years old, able to give birth.",
    "Man, 38 years old, unable to give birth.",
    "Woman, 30 years old, able to give birth.",
    "Man, 42 years old, unable to give birth.",
    "Woman, 33 years old, able to give birth.",
    "Man, 39 years old, unable to give birth.",
    "Woman, 29 years old, able to give birth.",
    "Man, 36 years old, unable to give birth.",
    "Woman, 31 years old, able to give birth.",
    "Man, 43 years old, unable to give birth.",
    "Woman, 26 years old, able to give birth.",
    "Man, 37 years old, unable to give birth.",
    "Woman, 34 years old, able to give birth.",
    "Man, 41 years old, unable to give birth."
]
HEALTH_OPTIONS = [
    "Common Cold",
    "Headache",
    "Nausea",
    "Diarrhea",
    "Minor Injury",
    "Sprain",
    "Fracture",

    "Pneumonia",
    "Infection",
    "Radiation Sickness",
```

```
    "Poisoning",
    "Mental Disorder",
    "Severe Injury",
    "Asthma",
    "Diabetes",
    "Heart Disease",
    "Cancer",
    "Chronic Mental Disorder",
    "Disability",
    "Sepsis",
    "Internal Bleeding",
    "Stroke",
    "Heart Attack",
    "Metastatic Cancer",
]
PHOBIA_OPTIONS = [
    "Fear of heights",
    "Fear of spiders",
    "Fear of the dark",
    "Fear of public speaking",
    "Fear of enclosed spaces",
    "Fear of flying",
    "Fear of snakes",
    "Fear of crowds",
    "Fear of needles",
    "Fear of germs or dirt",
    "Fear of clowns",
    "Fear of vomiting",
    "Fear of dogs",
    "Fear of thunder and lightning",
    "Fear of being alone",
    "Fear of blood",
    "Fear of drowning",
    "Fear of fire",
    "Fear of mirrors",
    "Fear of ghosts",
]
```

```
HOBBY_OPTIONS = [
    "Painting",
    "Basketball",
    "Gaming",
    "Cooking",
    "Reading",
    "Photography",
    "Hiking",
    "Singing",
    "Dancing",
    "Writing",
    "Playing a musical instrument",
    "Fishing",
    "Gardening",
    "Cycling",
    "Knitting",
    "Woodworking",
    "Yoga",
    "Traveling",
    "Collecting stamps",
    "Chess",
    "Pottery",
]
KNOWLEDGE_OPTIONS = [
    "Know how to purify water",
    "Know basic first aid",
    "Know how to start a fire without matches",
    "Know how to identify edible plants",
    "Know how to build a shelter",
    "Know how to navigate without a compass",
    "Know basic self-defense techniques",
    "Know how to forage for food",
    "Know how to preserve food",
    "Know how to tie various knots",
    "Know how to make basic tools",
    "Know how to identify signs of dangerous weather",
    "Know how to signal for help",
```

```
    "Know how to administer CPR",
    "Know how to set traps for small game",
    "Know how to mend clothing",
    "Know how to construct simple weapons for self-defense",
    "Know how to identify different types of terrain",
    "Know how to find or create sources of light",
    "Know how to navigate using the stars",
]
ADDITIONAL_INFO_OPTIONS = [
"Know how to perform basic car maintenance",
"Speak multiple languages fluently",
"Have experience in wilderness survival",
"Know how to operate various types of firearms",
"Have a background in engineering or mechanics",
"Have medical training or experience",
"Be proficient in coding or computer programming",
"Have experience in farming or agriculture",
"Know how to navigate using celestial navigation",
"Have experience in hunting or tracking",
"Be skilled in negotiation and diplomacy",
"Possess knowledge of local flora and fauna",
"Have experience in construction or carpentry",
"Know how to generate electricity from alternative sources",
"Be trained in martial arts or self-defense",
"Possess knowledge of herbal medicine",
"Have experience in off-grid living",
"Be proficient in wilderness cooking",
"Possess knowledge of radio communication",
"Be skilled in improvisation and problem-solving",
]
LUGGAGE_OPTIONS = [
"3 AK-47",
"The block of water",
"Seeds",
"First aid kit",
"Multi-tool",
"Solar-powered charger",
```

```
"Portable water filter",
"Tent",
"Sleeping bag",
"Canned food",
"Fire starter kit",
"Flashlight with extra batteries",
"Map and compass",
"Emergency shelter",
"Survival knife",
"Portable stove",
"Rope or paracord",
"Emergency blanket",
"Signal mirror",
"Whistle",
"Duct tape",
"Emergency radio",
"Fishing gear",
"Compact axe or hatchet",
"Emergency cash",
"Personal hygiene items",
"Notepad and pen",
"Solar blanket",
]
```

Додаток Д

Файл room_details.html

```
{% include 'header.html' %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Game Room</title>
    <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css" rel="stylesheet">
    <style>
        .card-container {
            display: flex;
            flex-wrap: wrap;
        }
        .card {
            flex: 1;
            margin: 10px;
        }
        .rounded-xxl {
            border-radius: 1.5rem;
        }
        .show-button{
            background-color: #0D0D0D;
            width: 60px;
            height: 50px;
        }
    </style>
</head>
<body class="text-white " style="background-color: #0D0D0D;">
    <div class="container mt-5">
        {% if room.game_started == False %}
            <ul class="list-group">
```

```
                {% for place in places %}
                <li id="place{{ place.id }}" class="list-group-item d-
flex justify-content-between align-items-center bg-dark border-white
border-top ">
                    <div class="">
                        {% if place.player_name %}
                            <span>{{ place.player_name }}</span>
                        {% else %}
                            <span class="text-muted">Вільне місце</span>
                        {% endif %}
                        <span class="badge badge-primary badge-
pill">Place №{{ forloop.counter }}</span>
                    </div>
                    {% if not place.player_name %}
                        <button class="btn btn-primary"
onclick="takePlace({{ place.id }})">Take a Place</button>
                    {% endif %}
                </li>
                {% endfor %}
            </ul>
        {% endif %}

        {% if messages %}
            <ul class="messages mt-3">
                {% for message in messages %}
                    <li class="alert alert-{{ message.tags }}">{{ message
}}</li>
                {% endfor %}
            </ul>
        {% endif %}

        {% if room.game_finished %}
            <div class="alert alert-warning mt-3" role="alert">
                The players who remained outside the bunker survived and
decided to take revenge on those who got into the bunker. In order to
resist them, the inhabitants of the bunker need to have strong people,
those who know martial arts, or have and know how to handle firearms
```

```
            </div>
        {% endif %}
    </div>

    <div class="ml-5 d-flex justify-content-center">
        {% if room.game_started == True and room.game_finished == False%}
            <p id="currentTurnPlayer" class="display-4 font-weight-bold
mb-5">Current player: {{ room.current_turn_player.username }}</p>
        {% endif %}
    </div>

    <div>
       <div class="ml-5">
            {% if room.voting_started and room.game_finished == False %}
                <form id="voteForm" method="post" action="{% url
'vote_endpoint' room_id=room.id %}">
                    {% csrf_token %}
                    <input type="hidden" name="room_id" value="{{ room.id
}}">
                    <div class="form-group">
                        <label for="selected_player_name">Choose
player:</label>
                        <select id="selected_player_name"
name="selected_player_name" class="form-control" style="width: 200px;">
                            {% if room.turn_ended and room.voting_started
%}
                                {% for player in players %}
                                    {% if player != user %}
                                        <option value="{{ player.username
}}">{{ player.username }}</option>
                                    {% endif %}
                                {% endfor %}
                            {% endif %}
                        </select>
                    </div>
```

```
                    <button type="submit" class="btn btn-primary mt-2" {%
if not room.turn_ended and not room.voting_started %} disabled {% endif
%}>Vote</button>
                </form>
            {% endif %}
        </div>
<div class="container ">
            <div class="row card-container ">
                {% for place in places %}
                    {% if not place.player_name.is_kicked and
room.game_started == True %}
                        <div class="card bg-primary text-white col-md-5
bg-warning rounded-xxl ml-3">
                            <h5 class="card-title mt-3 mb-1 text-dark
text-center">{{ place.player_name }}</h5>
                            <div class="card-body mb-3 mt-3 rounded"
style="background-color: #0D0D0D;">
                                {% with
character_card=place.character_card %}
                                    {% if character_card %}
                                        <ul class="list-group list-group-
flush ">
                                            {% if character_card.player
== request.user %}
                                                <li class="list-group-
item bg-white text-dark mt-1 rounded d-flex">Bio: {{ character_card.bio
}}
                                                    {% if
room.current_turn_player == user %}
                                                        <button
class="btn btn-secondary btn-sm ml-auto show-button"
onclick="toggleVisibility('{{ character_card.id }}', 'bio'
)">Show</button>
                                                    {% endif %}
                                                </li>
```

```
                                                    <li class="list-group-
item bg-white text-dark mt-1 rounded d-flex">Health: {{
character_card.health }}
                                                        {% if
room.current_turn_player == user %}
                                                            <button
class="btn btn-secondary btn-sm mt-1 ml-auto show-button"
onclick="toggleVisibility('{{ character_card.id }}',
'health')">Show</button>
                                                        {% endif %}
                                                    </li>
                                                    <li class="list-group-
item bg-white text-dark mt-1 rounded d-flex">Phobia: {{
character_card.phobia }}
                                                        {% if
room.current_turn_player == user %}
                                                            <button
class="btn btn-secondary btn-sm ml-auto show-button"
onclick="toggleVisibility('{{ character_card.id }}',
'phobia')">Show</button>
                                                        {% endif %}
                                                    </li>
                                                    <li class="list-group-
item bg-white text-dark mt-1 rounded d-flex">Hobby: {{
character_card.hobby }}
                                                        {% if
room.current_turn_player == user %}
                                                            <button
class="btn btn-secondary btn-sm ml-auto show-button"
onclick="toggleVisibility('{{ character_card.id }}',
'hobby')">Show</button>
                                                        {% endif %}
                                                    </li>
                                                    <li class="list-group-
item bg-white text-dark mt-1 rounded d-flex">Knowledge: {{
character_card.knowledge }}
```

```
                                                    {% if
room.current_turn_player == user %}
                                                        <button
class="btn btn-secondary btn-sm ml-auto show-button"
onclick="toggleVisibility('{{ character_card.id }}',
'knowledge')">Show</button>
                                                    {% endif %}
                                                </li>
                                                <li class="list-group-
item bg-white text-dark mt-1 rounded d-flex">Additional Info: {{
character_card.additional_info }}
                                                    {% if
room.current_turn_player == user %}
                                                        <button
class="btn btn-secondary btn-sm ml-auto show-button"
onclick="toggleVisibility('{{ character_card.id }}',
'additional_info')">Show</button>
{% endif %}
                                                </li>
                                                <li class="list-group-
item bg-white text-dark mt-1 rounded d-flex">Baggage: {{
character_card.luggage }}
                                                    {% if
room.current_turn_player == user %}
                                                        <button
class="btn btn-secondary btn-sm ml-auto show-button"
onclick="toggleVisibility('{{ character_card.id }}',
'luggage')">Show</button>
                                                    {% endif %}
                                                </li>
                                            {% else %}
                                                <li class="list-group-
item bg-white text-dark rounded mt-1">Bio: {% if not
character_card.bio_hidden %}{{ character_card.bio }}{% else %}Hidden{%
endif %}</li>
                                                <li class="list-group-
item bg-white text-dark rounded mt-1">Health: {% if not
```

```
character_card.health_hidden %}{{ character_card.health }}{% else
%}Hidden{% endif %}</li>
                                                    <li class="list-group-
item bg-white text-dark rounded mt-1">Phobia: {% if not
character_card.phobia_hidden %}{{ character_card.phobia }}{% else
%}Hidden{% endif %}</li>
                                                    <li class="list-group-
item bg-white text-dark rounded mt-1">Hobby: {% if not
character_card.hobby_hidden %}{{ character_card.hobby }}{% else
%}Hidden{% endif %}</li>
                                                    <li class="list-group-
item bg-white text-dark rounded mt-1">Knowledge: {% if not
character_card.knowledge_hidden %}{{ character_card.knowledge }}{% else
%}Hidden{% endif %}</li>
                                                    <li class="list-group-
item bg-white text-dark rounded mt-1">Additional Info: {% if not
character_card.additional_info_hidden %}{{ character_card.additional_info
}}{% else %}Hidden{% endif %}</li>
                                                    <li class="list-group-
item bg-white text-dark rounded mt-1">Baggage: {% if not
character_card.luggage_hidden %}{{ character_card.luggage }}{% else
%}Hidden{% endif %}</li>
                                                {% endif %}
                                            </ul>
                                        {% else %}
                                            <p>No Character Card available
for this player</p>
                                        {% endif %}
                                    {% endwith %}
                                </div>
                            </div>
                        {% endif %}
                    {% empty %}
                        <div class="col-12">
                            <div class="alert alert-info">Кімната
порожня</div>
                        </div>
```

```
                {% endfor %}
            </div>
        </div>
    </div>

    <div class="d-flex justify-content-center align-items-center mt-5 ">
        {% if room.game_finished == False %}
            {% if room.current_turn_player %}
                {% if room.current_turn_player == user %}
                    {% if not place.turn_finished and not
place.can_end_turn %}
                        {% if not room.turn_ended and not
room.voting_started %}
                            <button class="btn btn-success end-turn-
button btn-lg rounded-xxl  px-5"  style="background-color: #038C33" data-
room-id="{{ room_id }}">End Turn</button>
                        {% endif %}
                    {% endif %}
                {% endif %}
            {% else %}
                <p>Game is not started yet. Please wait for room owner to
start a game... </p>
            {% endif %}
        {% endif %}
    </div>

    <dev>
        {% if user.is_authenticated and user == room.creator and not
room.game_starteId %}
            <form method="post" action="{% url 'start_game' room.id %}">
                {% csrf_token %}
                <button type="submit" class="btn btn-success mt-3 ml-
4">Start Game</button>
            </form>
        {% endif %}
    </dev>
{% if room.voting_started == True %}
```

```
        <div class="container mt-3">
            <div class="row">
                <div class="col">
                    <ul id="voteResultsList" class="list-group">
                    </ul>
                </div>
            </div>
            <div class="row mt-3 mb-5">
                <div class="col">
                    <button onclick="getVoteResults({{ room.id }})"
class="btn btn-primary">Get Voting Results</button>
                </div>
            </div>
        </div>
    {% endif %}

    <div class="ml-4 mt-4 mb-4 ">
        {% if user.is_authenticated and room.creator == user %}
            <button class="btn btn-danger mt-3" onclick="deleteRoom({{
room.id }})">Delete Room</button>
        {% endif %}
    </div>

    <script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.mi
n.js"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.
js"></script>
    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@10"></script>
i
    <script>
        function start_game() {
            var xhr = new XMLHttpRequest();
            xhr.onload = function() {
```

```
            if (xhr.status === 200) {
                window.location.reload();
            } else {
                alert('Помилка: ' + xhr.statusText);
            }
        };
        xhr.open('POST', '{% url "start_game" room.id %}', true);
        xhr.setRequestHeader('X-CSRFToken', '{{ csrf_token }}');
        xhr.send();
    }

    setTimeout(function() {
        var messages = document.querySelectorAll('.messages li');
        if (messages.length > 0) {
            messages[0].remove();
        }
    }, 5000);
    $.ajax({
        type: "POST",
        url: "/vote_endpoint/",
        data: {
            selected_player_name: selectedPlayerName,
            room_id: roomId,
            csrfmiddlewaretoken: csrftoken
        },
        success: function(response) {
            if (response.message) {
                Swal.fire({
                    title: 'Голосування завершено',
                    text: response.message,
                    icon: 'success'
                });
            }
        },
        error: function(xhr, status, error) {
            console.error(error);
        }
```

```
        });
    </script>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></
script>

    <script>
        document.addEventListener("DOMContentLoaded", function() {
            var roomId = document.getElementById("roomId").value;
            getVoteResults(roomId);
        });
        function getVoteResults(roomId) {
            $.ajax({
                type: "GET",
                url: /get_vote_results/${roomId}/,
                success: function(response) {
                    displayVoteResults(response.voteCounts);
                },
                error: function(xhr, status, error) {
                    console.error(error);
                }
            });
        }

        function displayVoteResults(voteCounts) {
            var voteResultsList =
document.getElementById("voteResultsList");
            voteResultsList.innerHTML = '';
            voteCounts.forEach(function(item) {
                var listItem = document.createElement('li');
                listItem.textContent = ${item.target_player__username}:
${item.total_votes};
                voteResultsList.appendChild(listItem);
            });
            document.getElementById("voteResultsSection").style.display =
"block";
        }
```

```
function vote() {
            var selectedPlayerName = $("#selected_player_name").val();
            if (!selectedPlayerName) {
                alert("Будь ласка, виберіть гравця для голосування.");
                return; // Припинити виконання функції, якщо гравець не
вибрано
            }
            var roomId = $("#voteForm input[name='room_id']").val();
            var url = "{% url 'vote_endpoint' room_id=room.id %}";
            $.ajax({
                type: "POST",
                url: url,
                data: {
                    'selected_player_name': selectedPlayerName, //
Змінено на передачу імені гравця
                    'room_id': roomId,
                    'csrfmiddlewaretoken': '{{ csrf_token }}'
                },
                success: function(response) {
                    alert(response.message);
                },
                error: function(xhr, status, error) {
                    console.error(error);
                }
            });
        }
    function handleEndTurnClick(event) {
        console.log('Button clicked!');
        var roomId = event.target.getAttribute('data-room-id');
        endTurn(roomId);
    }

    var endTurnButtons = document.querySelectorAll('.end-turn-button');
    endTurnButtons.forEach(function(button) {
        button.addEventListener('click', handleEndTurnClick);
    });
```

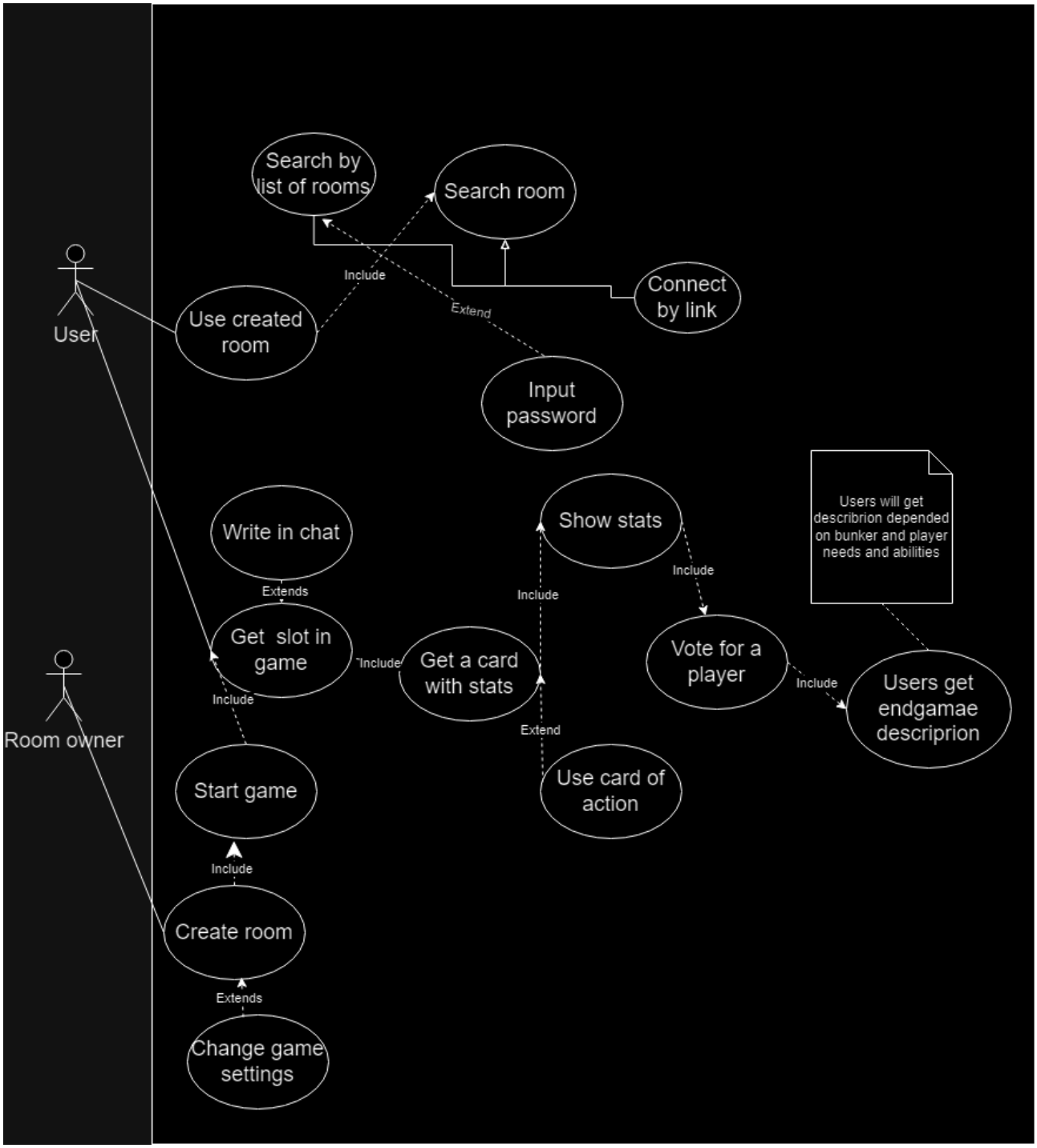
```
    function endTurn(roomId) {
            var csrfToken =
document.querySelector('[name=csrfmiddlewaretoken]').value;
            fetch(/end_turn/${roomId}/, {
                method: 'POST',
                headers: {
                    'X-CSRFToken': csrfToken,
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body: ''
            })
            .then(response => {
                if (response.ok) {
                    var currentTurnButton = document.querySelector('.end-
turn-button[data-room-id="' + roomId + '"]');
                    currentTurnButton.disabled = true;
                    var nextPlayerButton = getNextPlayerButton();
                    if (nextPlayerButton) {
                        nextPlayerButton.disabled = false;
                    }
                    location.reload();
                } else {
                    alert('Failed to end turn');
                }
            })
            .catch(error => console.error('Error:', error));
    }

        function getNextPlayerButton() {
            var endTurnButtons = document.querySelectorAll('.end-turn-
button');
            for (var i = 0; i < endTurnButtons.length; i++) {
                if (!endTurnButtons[i].disabled) {
                    return endTurnButtons[i + 1];
                }
            }
            return null;
```

```
        }

        function toggleVisibility(characterCardId, characteristic) {
            var csrfToken =
document.querySelector('[name=csrfmiddlewaretoken]').value;

fetch(/toggle_visibility/${characterCardId}/${characteristic}/, {
                method: 'POST',
                headers: {
                    'X-CSRFToken': csrfToken,
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body: ''
            })
            .then(response => {
                if (response.ok) {
                    location.reload();
                } else {
                    alert('Failed to toggle visibility');
                }
            })
            .catch(error => console.error('Error:', error));
}

    function revealCharacteristics() {
      var selectedPlayer = document.getElementById("playerSelect").value;
      var characteristicsDiv =
document.getElementById("characteristics");
// Встановлюємо атрибут hidden в залежності від вибраного гравця
      if (selectedPlayer === "player1") {
        characteristicsDiv.hidden = false;
      } else {
        characteristicsDiv.hidden = true;
      }
    }
        function takePlace(placeId) {
    fetch(/take_place/{{ room_id }}/${placeId}/)
```

```
        .then(response => response.json())
        .then(data => {
            if (data.error) {
                alert(data.error);
            } else {
                const placeElement =
document.getElementById(place${placeId});
                const playerPlaceText =
document.createTextNode(${data.player_name} Place №${data.place_number});
                placeElement.innerHTML = '';
                placeElement.appendChild(playerPlaceText);
            }
        })
        .catch(error => console.error('Error:', error));
}
    function deleteRoom(roomId) {
        fetch(/delete_room/${roomId}/, {
            method: 'DELETE',
            headers: {
                'X-CSRFToken': '{{ csrf_token }}',
            }
        })
        .then(response => {
            if (response.ok) {
                // Перенаправлення на головну сторінку, якщо видалення
пройшло успішно
                window.location.href = '/';
            } else {
                alert('Failed to delete room');
            }
        })
        .catch(error => console.error('Error:', error));
    }
    </script>
</body>
</html>
```

Додаток Е

UML- діаграма



метадані

Заголовок

Розробка онлайн-платформи для гри в настільну гру "Бункер"

Автор
Граб М.

Науковий керівник / Експерт
кандидат технічних наук Сергій Ващишак

підрозділ
King Danylo University

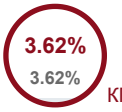
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

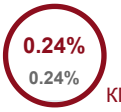
Заміна букв	ß	1
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	28

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КП 2

25
Довжина фрази для коефіцієнта подібності 2

16104
Кількість слів

122565
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	https://www.autoscripts.net/how-to-logout-in-django/	39	0.24 %
2	https://essuir.sumdu.edu.ua/bitstream/123456789/92906/1/Levytskiy_bac_rob%20.pdf	16	0.10 %
3	https://simpleisbetterthancomplex.com/tutorial/2017/02/18/how-to-create-user-sign-up-view.html	16	0.10 %
4	https://dev.to/coderasha/create-advanced-user-sign-up-view-in-django-step-by-step-k9m	15	0.09 %
5	Березовський.docx [redacted] 6/19/2023 Odessa National Polytechnic University (ІКС, кафедра інформац. технологій)	14	0.09 %