

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Органайзер ведучого гри «Підземелля і дракони»

Виконав студент групи ПЗк-21-1
БОРЧКАКІВСЬКИЙ Андрій Сергійович

Керівник роботи:
КОНЦЕДАЙЛО Валерій Валерійович

Консультант:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **БОРЧАКІВСЬКИЙ Андрій Сергійович**

Керівник роботи: **КОНЦЕДАЙЛО Валерій Валерійович**

Тема роботи: **«Органайзер ведучого гри «Підземелля і дракони»,
затверджена Наказом закладу вищої освіти від «23» лютого 2024 р. №74/с**

Вихідні дані для роботи: **розробка органайзера ведучого гри «підземелля і дракони».**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Концедайло В.В., Чижмотря О.В.	27.02.2024р.	27.02.2024р.
2	Концедайло В.В., Чижмотря О.В.	27.02.2024р.	27.02.2024р.
3	Концедайло В.В., Чижмотря О.В.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмної реалізації органайзера ведучого гри «Підземелля і дракони» та пояснювальної записки. Пояснювальна записка до дипломної роботи містить 62 сторінки, 26 ілюстрацій, 9 таблиць, 1 додаток та 30 літературних джерел.

Метою цієї роботи є розробка WEB-орієнтованого органайзера ведучого гри «Підземелля і дракони», з можливістю проводити власні ігрові сесії з елементами автоматизації.

У роботі викладено аналіз та вимоги до системи, а також проведено порівняльний аналіз з існуючими аналогами. Описано процес розробки бази даних та програмної платформи для органайзера. Нарешті, розглянуто порядок встановлення сервісу, описано інтерфейс та надано інструкції для користувачів.

КЛЮЧОВІ СЛОВА: REACT, МОВА ПРОГРАМУВАННЯ, КЛІЄНТ, API, БАЗА ДАНИЙ, MONGODB NODE.JS,

					ІПЗ.КР.Б – 121 – 24 – ПЗ							
Змн.	Арк.	№ докум.	Підпис	Дата	«Органайзер ведучого гри «Підземелля і дракони» Пояснювальна записка			Літ.	Арк.	Аркушів		
Розроб.		Борчаківський А.С								3	62	
Керівник		Концедайло В.В.						Житомирська політехніка, група ІПЗк-21-1				
Рецензент		Єфремов Ю.М.										
Зав. каф.		Вакалюк Т.А.										

ABSTRACT

The bachelor's thesis consists of the software implementation of a game master organizer for "Dungeons & Dragons" and an explanatory note. The explanatory note of the thesis comprises 62 pages, 26 illustrations, 9 tables, 1 appendix, and references to 30 literary sources.

The aim of this work is to develop a web-oriented game master organizer for "Dungeons & Dragons" with the ability to conduct gaming sessions with automation elements.

The work includes an analysis and requirements of the system, as well as a comparative analysis with existing analogs. It describes the process of developing a database and a software platform for online announcements with auction elements. Finally, the installation process of the platform is discussed, along with a description of the interface and instructions for users.

KEYWORDS: REACT, PROGRAMMING LANGUAGE, CLIENT, API, DATABASE, MONGODB, NODE.JS

		<i>Борчаківський А.С</i>			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		<i>Кониедайло В.В</i>				4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ВИМОГ ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ ОРГАНАЙЗЕРА	8
1.1 Технічне завдання на розробку органайзера.....	8
1.2 Аналіз існуючого програмного забезпечення за тематикою проекту	11
1.3 Аналіз вимог системи та користувачів органайзера	15
1.4 Обґрунтування вибору інструментальних засобів та архітектури органайзера	17
Висновки до першого розділу.....	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ ОРГАНАЙЗЕРА ВЕДУЧОГО ГРИ «ПІДЗЕМЕЛЛЯ І ДРАКОНИ».....	20
2.1 Опис акторів та алгоритмів роботи.....	20
2.2 Розробка бази даних.....	23
2.3 Комунікації і послідовність взаємодії об'єктів органайзера	28
Висновки до другого розділу	33
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОРГАНАЙЗЕРОМ ВЕДУЧОГО ГРИ «ПІДЗЕМЕЛЛЯ І ДРАКОНИ».....	34
3.1 Порядок встановлення та налаштування органайзера	34
3.2 Опис інтерфейсу програмного продукту	35
3.3 Тестування органайзера.....	41
Висновки до третього розділу.....	43
ВИСНОВКИ.....	44
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	45
ДОДАТКИ.....	48

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних;

ОС – операційна система

ПЗ – Програмне забезпечення

СКБД – Система керування базами даних

ТЗ – технічне завдання

API – application programming interface

JS – Java Script

		<i>Борчаківський А.С</i>			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		<i>Кониедайло В.В</i>				6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Кваліфікаційна робота бакалавра на тему «Органайзер ведучого гри «Підземелля і дракони» спрямована на створення інструменту, що полегшить підготовку та проведення гри для майстра (ведучого). Гра «Підземелля і дракони» є популярною настільною рольовою грою, яка вимагає від ведучого значної підготовки та управління різними аспектами гри, такими як створення та підтримка сюжету, контроль персонажів та монстрів, керування ресурсами і багато іншого.

Мета роботи полягає у розробці зручного та функціонального органайзера, який допоможе ведучому ефективно планувати і проводити ігрові сесії, зменшивши час на підготовку та підвищивши якість гри.

Мета обумовлює такі завдання:

- Розробка ТЗ до органайзеру.
- Аналіз уже існуючого аналогічного ПЗ.
- Аналіз вимог до органайзеру та до користувача.
- Проектування та реалізація органайзеру.
- Опис інтерфейсу користувача.

Об'єктом дослідження є WEB-орієнтований органайзер, що дозволяє ведучому створювати та проводити ігрові сервіси.

Предметом дослідження є інструменти та методи, що використовуються для підготовки та проведення гри, а також можливості їх оптимізації за допомогою органайзера.

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				7
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ВИМОГ ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ ОРГАНАЙЗЕРА

1.1 Технічне завдання на розробку органайзера

Назва додатку – органайзер ведучого гри «Підземелля і дракони»

Мета – проектування та реалізація органайзера для ведучого гри «Підземелля і дракони», що дозволить користувачам створювати власні ігрові сесії та проводити їх. Також органайзер має давати можливість створювати та використовувати власні пресети ворогів.

Актуальність - Гра «Підземелля і дракони» (Dungeons & Dragons, D&D) залишається однією з найпопулярніших настільних рольових ігор у світі вже протягом кількох десятиліть. Її популярність продовжує зростати, залучаючи все більше нових гравців та ведучих. У зв'язку з цим виникає потреба у зручних та ефективних інструментах, які могли б полегшити ведучим підготовку до гри, організацію ігрового процесу та управління різноманітними аспектами гри. Розробка органайзера для ведучого гри «Підземелля і дракони» є актуальною, оскільки дозволить значно підвищити ефективність підготовки та проведення ігрових сесій. Використання такого інструменту допоможе зменшити кількість помилок, підвищити якість гри та забезпечити більш комфортні умови для ведучого. Це, в свою чергу, сприятиме ще більшому залученню нових гравців та популяризації гри.

Технічні вимоги:

- Сервіс має підтримувати одночасну роботу більше, ніж 300 користувачів
- Час відгуку при виконанні стандартних задач становить максимум 5 секунд, при виконанні більш складних задач – 20 секунд.
- Сервіс має відповідати усім стандартам інтерфейсу користувача Microsoft® Windows® або ОС UNIX.
- Інтерфейс сервісу має бути виконаний у форматі WEB-додатку.
- Усі запити має обробляти сервер.

Функціональні вимоги.

		Борчаківський А.С.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

Авторизація та реєстрація користувачів.

Використання органайзеру є доступним лише для зареєстрованих користувачів, тому на сервісі має бути створений функціонал для створення облікових засобів та для авторизації користувачів в системі.

Створення ігрових сесій

Кожен користувач має можливість створювати ігрові сесії та запросити в сесію до 4х користувачів. Управління над сесією виконується лише ведучим, а інші учасники надають доступ до власних ігрових персонажів.

Створення власних персонажів

Кожен користувач має можливість створити власного ігрового персонажа, якого в майбутньому можна додати до ігрової сесії. Кожен персонаж має можливість бути долученим лише до однієї сесії та бути видаленим у разі потреби.

Створення ворогів

Кожен користувач має можливість створити власних ворогів, кожного з яких, при потребі, будь-який майстер може додати до власної ігрової сесії та використовувати в ігровому процесі.

Додавання ворогів до сесії

Кожен ведучий має можливість додати до сесії ворогів, що були створені раніше іншими користувачами, або самим ведучим. Він може фільтрувати ворогів по їх іменам на сторінці додавання ворога.

Проведення битв

Ведучий може проводити бої між гравцями та ворогами. Для цього має бути розроблено спеціальне вікно, де вороги та гравці мають ставати в чергу на атаку. Атаку проводить завжди перший боєць в черзі, ціль що буде атаковано визначає ведучий.

Отримання запрошень до сесій

Кожен користувач має можливість отримувати, переглядати та приймати запрошення до сесій.

Застосовність:

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В				9
Змн.	Арк.	№ докум.	Підпис	Дата		

1. Зручність використання:

Інтерфейс має бути створений у вигляді веб-додатку з простим і мінімалістичним дизайном. Основна вимога - інтерфейс повинен бути інтуїтивно зрозумілим.

Надійність:

1. Доступність:

Сервіс має бути доступним для користувачів цілодобово. Недоступність сервісу може бути виправдана лише технічними роботами та аварійними ситуаціями

2. Напрацювання на відмову:

Сервіс зобов'язаний безвідмовно працювати мінімум 30 робочих днів.

3. Норма дефектів:

Кількість помилок не має перевищувати 1 на 10000 рядків коду

Продуктивність:

1. Одночасно працюючі користувачі:

Сервіс має підтримувати одночасну роботу більше, ніж 300 користувачів

2. Час відгуку:

Час відгуку при виконанні стандартних задач становить максимум 5 секунд, при виконанні більш складних задач – 20 секунд.

Придатність до експлуатації:

1. Масштабованість

Оскільки сервіс буде випущено в публічне користування не є можливим точно визначити кількість одночасних користувачів, що буду працювати з сервісом. Для початку мінімальною кількістю користувачів, яка може одночасно працювати з сервісом, є 400 клієнтів. Але ця кількість має бути збільшена з підвищенням популярності сервісу.

2. Оновлення версій:

Оновлення версій повинні здійснюватися автоматично через систему контролю версій та систему оновлення на комп'ютерах користувачів.

Обмеження проектування:

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В				10
Змн.	Арк.	№ докум.	Підпис	Дата		

1. Застосовувані стандарти:

Система повинна відповідати стандартам інтерфейсу користувача для операційних систем Microsoft® Windows® та UNIX, а також сучасним принципам дизайну веб-додатків.

2. Вимоги до середовища виконання

Система має задовольняти зазначеним нижче вимогами на комп'ютері в наступній мінімальній комплектації:

- 2048 МБ оперативної пам'яті
- 10 ГБ вільного простору на диску
- Процесор з тактовою частотою 2 ГГц
- ОС Windows або Linux
- Інтернет з'єднання

3. Вимоги до СКБД і доступу до даних:

В основі системи повинна використовуватися промислова СКБД, орієнтована на документи, наприклад, MongoDB. Усі запити до даних мають здійснюватися через інтеграцію з MongoDB.

Антивимоги:

1. Не використовувати дані, що можуть порушувати законодавство
2. Не використовувати дані, що можуть зашкодити користувачам

Бізнес логіка:

Реклама сервісу на інших сайтах. Умова: власні кошти.

- Покупки та послуги: сервіс не призначений для отримання коштів напряму від користувачів. Увесь прибуток може бути отримано шляхом інтеграції реклами.

1.2 Аналіз існуючого програмного забезпечення за тематикою проекту

У сучасному цифровому світі рольові ігри, зокрема «Підземелля і дракони», переживають нову хвилю популярності. Разом з цим зростає і потреба у зручних та ефективних інструментах для їх підготовки та проведення. Зараз існує безліч веб-додатків, які спрямовані на полегшення

		Борчаківський А.С.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

цього процесу, кожен з них з унікальними можливостями та призначенням для конкретної аудиторії.

D&D Beyond

D&D Beyond – це офіційна цифрова платформа (рис. 1.1) для гри «Підземелля і дракони». Вона надає доступ до бази даних правил, персонажів, монстрів, магічних предметів та інших ресурсів. Користувачі можуть створювати та керувати персонажами, а також готувати матеріали для гри за допомогою цифрових інструментів [29].

Переваги:

- Повний доступ до всіх офіційних правил, монстрів, заклинань та інших матеріалів.
- Легкий у використанні, з інтуїтивним інтерфейсом.
- Доступ до матеріалів з будь-якого пристрою через інтернет.

Недоліки:

- Базові функції доступні безкоштовно, але для повного доступу до контенту необхідна платна підписка.
- Менше можливостей для кастомізації у порівнянні з іншими інструментами.

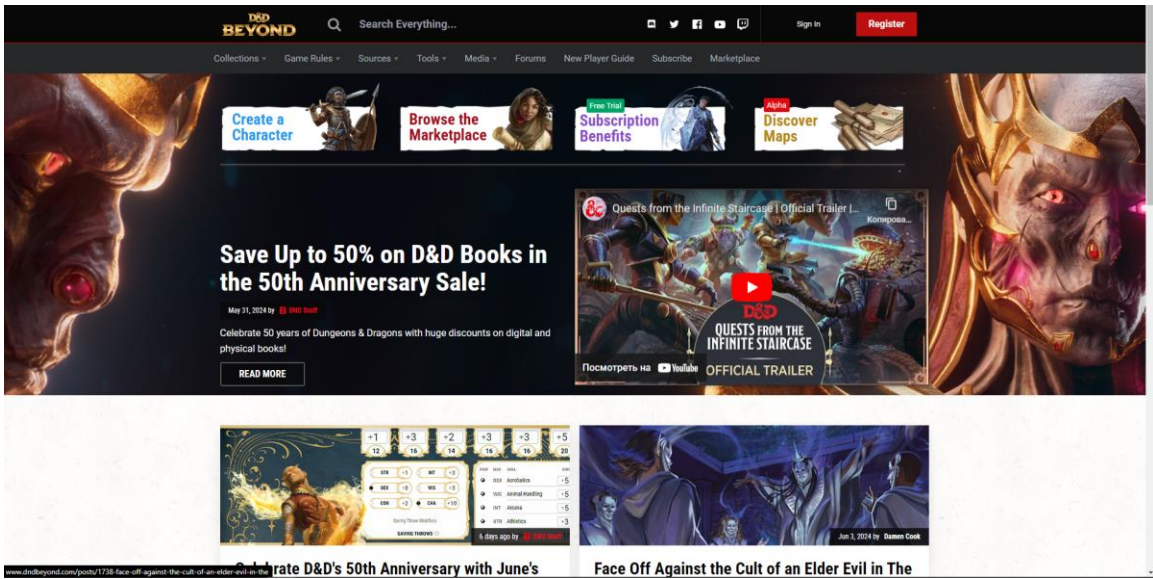


Рис.1.1 – Інтерфейс D&D Beyond

Roll20

Roll20 – це веб-платформа (рис. 1.2) для проведення настільних рольових ігор онлайн. Вона надає можливість створювати віртуальні ігрові столи, завантажувати карти, фішки, а також використовувати інтерактивні інструменти для ведення гри [30].

Переваги:

- Підтримка карт, фішок, а також інтеграція з іншими інструментами для ведення гри.
- Можливість використовувати аудіо та відео для створення більш інтерактивної гри.
- Базові функції доступні безкоштовно, що дозволяє користувачам почати грати без додаткових витрат.

Недоліки:

- Може бути складно для нових користувачів через багатий функціонал і складність налаштувань.
- Для гри необхідний постійний доступ до інтернету, що може бути проблематично для деяких користувачів.

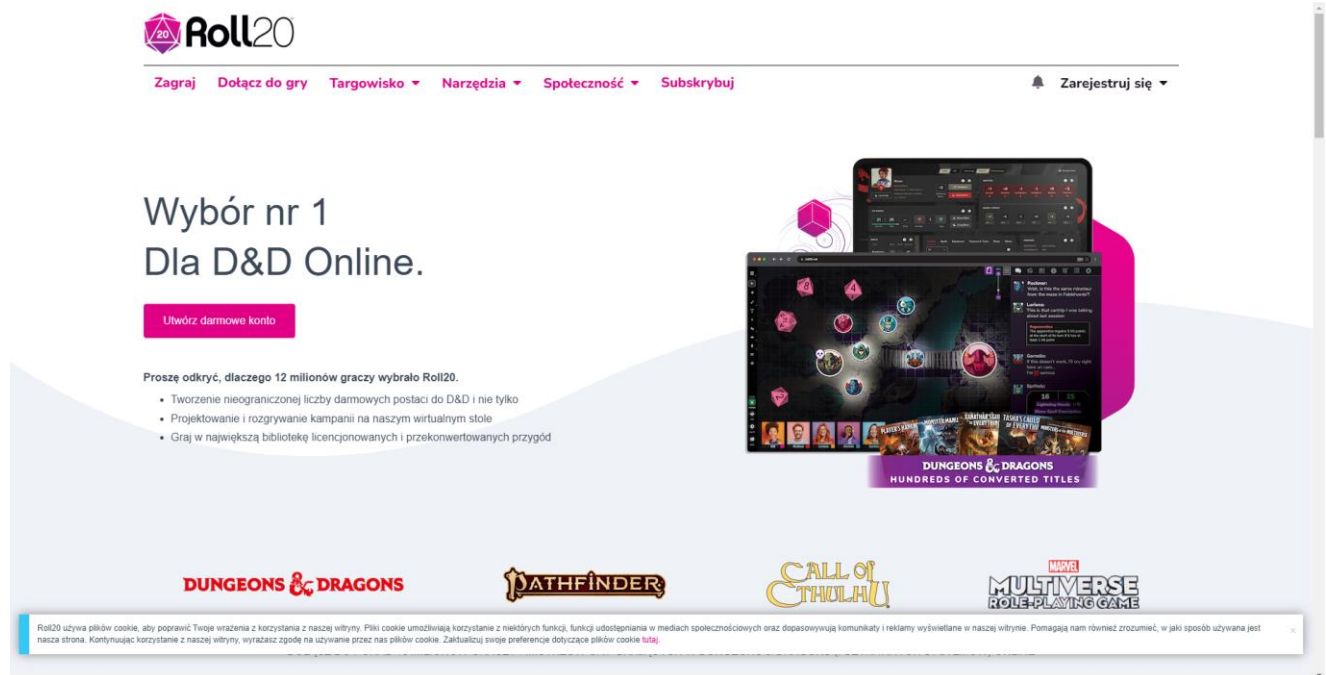


Рис.1.2 – Інтерфейс Roll20

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				13
Змн.	Арк.	№ докум.	Підпис	Дата		

Fantasy Grounds

Fantasy Grounds – це програмне забезпечення для проведення настільних рольових ігор в онлайн та офлайн режимах (рис. 1. 3). Воно підтримує різні системи рольових ігор і надає можливість використовувати модулі, карти, персонажів та інші ресурси [28].

Переваги:

- Можливість використання різних систем рольових ігор, не тільки D&D.
- Широкі можливості для кастомізації та використання додаткових модулів.
- Можливість працювати без постійного доступу до інтернету.

Недоліки:

- Висока вартість ліцензії та додаткових модулів.
- Важкий інтерфейс та складний процес налаштування, що може відлякати нових користувачів.

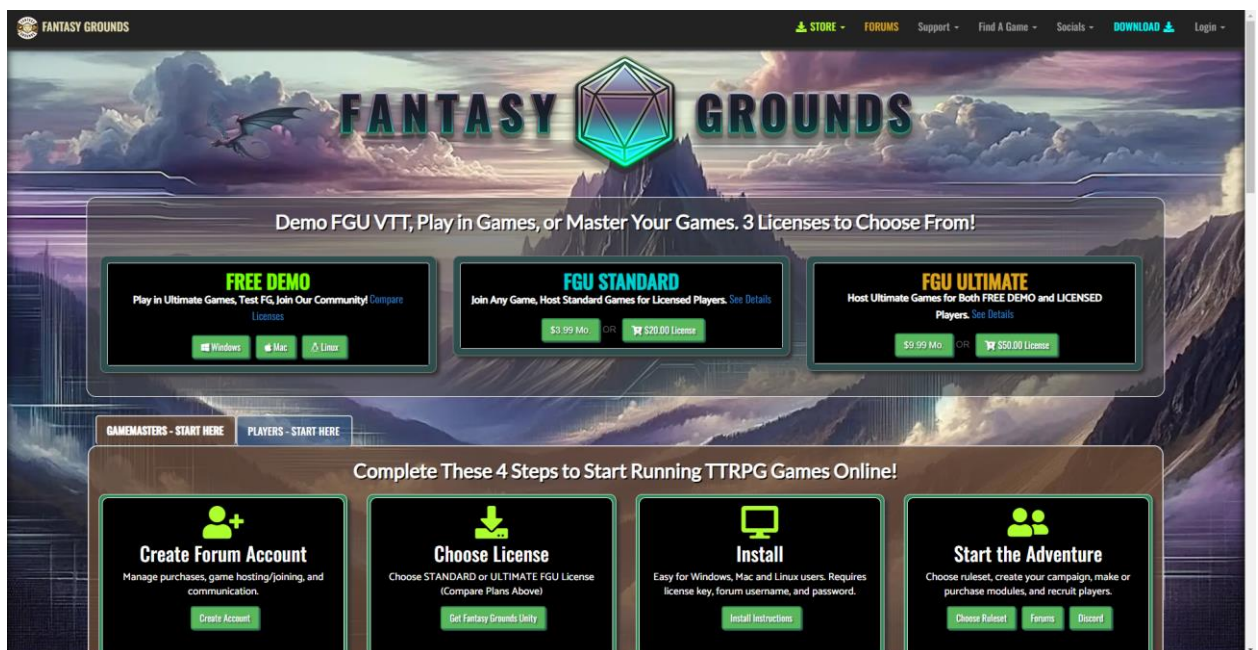


Рис.1.3 – Інтерфейс Fantasy Grounds

Аналізуючи три зазначені сервіси, можна побачити, що кожен з них має свої унікальні переваги та недоліки. D&D Beyond є найкращим вибором для тих, хто шукає офіційну інтеграцію та зручний інтерфейс, але обмежений у кастомізації. Roll20 підходить для тих, хто хоче створити інтерактивні ігрові

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В				14
Змн.	Арк.	№ докум.	Підпис	Дата		

сесії з мінімальними витратами, але готовий витратити час на освоєння платформи. Fantasy Grounds є потужним інструментом для досвідчених користувачів, які потребують високого рівня кастомізації, але готові платити за ліцензію та докладати зусиль для налаштування.

Таблиця 1.1

Порівняння сервісів

Аспект	Органайзер ведучого гри «Підземелля і дракони»	D&D Beyond	Roll20	Fantasy Grounds
Інтеграція з офіційними матеріалами	-	+	+/-	+/-
Інтерфейс та зручність використання	+	+	-	-
Кастомізація	-	-	+	+
Доступність	+	-	+	-
Підтримка офлайн режиму	-	-	-	+

1.3 Аналіз вимог системи та користувачів органайзера

Мета аналізу вимог полягає у зборі та аналізі вищого рівня потреб і можливостей для органайзера ведучого гри «Підземелля і дракони». Вимоги зосереджують увагу на функціональності, необхідній для співвласників та цільових користувачів, а також на причинах, що породжують ці потреби.

Бізнес-вимоги:

1. Основні вимоги: сервіс розробляється для надання можливості створення і проведення ігрових сесій по грі «Підземелля і дракони»
2. Представлення проекту: сервіс реалізовано у вигляді WEB-додатку, що містить інструменти та функціонал для проведення ігрових сесій по грі «Підземелля і дракони»

		Борчаківський А.С.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

3. Прискорення пошуку: реалізовано поле для фільтрації ворогів по їх іменам

4. Підтримка великої кількості користувачів: сервер може підтримувати одночасну роботу великої кількості користувачів та ефективно обробляти усі їх запити

Вимоги користувачів:

Гравець:

1. Має можливість створювати власних персонажів
2. Має можливість створювати нових ворогів
3. Має можливість переглядати запрошення до сесій
4. Має можливість приймати запрошення до сесій

Ведучий:

1. Має можливість створювати нові сесії
2. Має можливість створювати власних персонажів
3. Має можливість створювати нових ворогів
4. Має можливість запрошувати гравців до сесій
5. Має можливість додавати ворогів до сесії
6. Має можливість редагувати інформацію про персонажів гравців
7. Має можливість проводити бої в сесії

Адміністратор:

1. Має можливість переглядати новостворених ворогів
2. Має можливість оборювати або видаляти нових ворогів

На основі вимог була розроблена діаграма варіантів використання.

Звернувши увагу на діаграму (рис. 1.4) використання можна побачити усі дії, що доступні користувачам з різними ролями. Варто звернути увагу, що розмежування ролей гравця і ведучого є умовним, тобто користувач, що створив сесію вважається ведучим, а користувач, що ще не створював сесій вважається гравцем [5].

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В				16
Змн.	Арк.	№ докум.	Підпис	Дата		

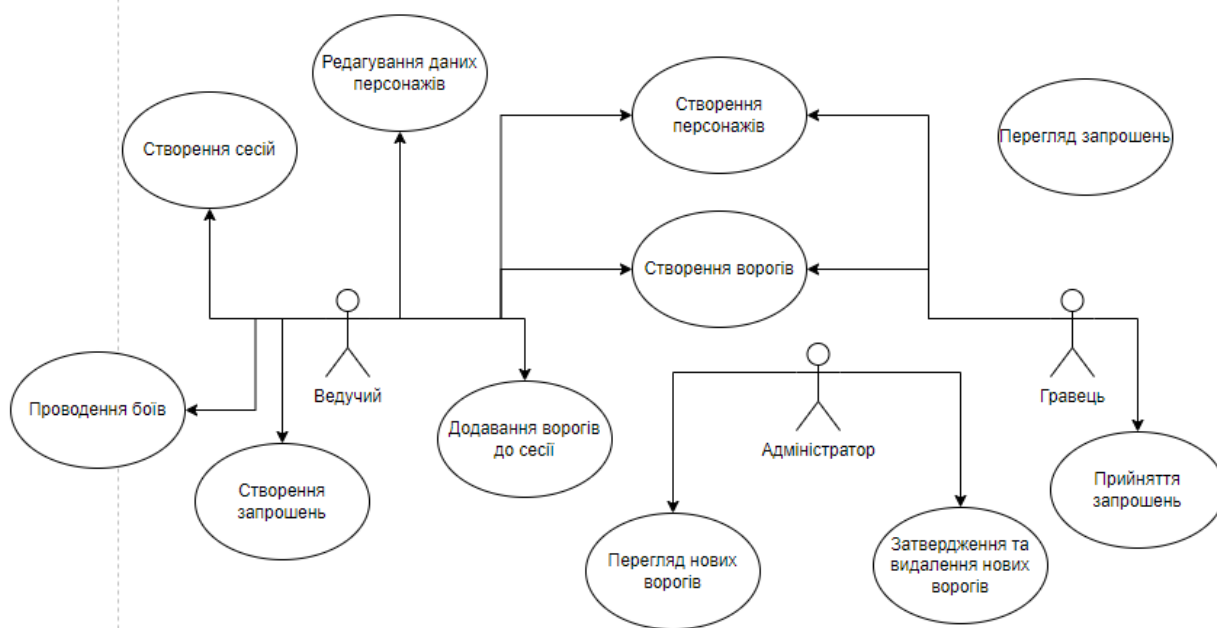


Рис. 1.4 – Діаграма варіантів використання

1.4 Обґрунтування вибору інструментальних засобів та архітектури органайзера

Клієнт-серверна архітектура є однією з найбільш поширених та ефективних моделей в розробці веб-додатків. У цій архітектурі взаємодія між користувачем (клієнтом) та сервером відбувається через передачу запитів та відповідей.

Ось декілька причин, чому саме клієнт-серверна архітектура є вигідним вибором для розробки органайзеру ведучого гри «Підземелля і дракони»:

Масштабованість: Клієнт-серверна архітектура дозволяє легко масштабувати систему, розділивши функціональність між клієнтами та серверами. Це особливо важливо для веб-додатків, які мають велику кількість користувачів та потребують швидкого відгуку на запити.

Централізоване управління: Сервер відповідає за централізоване зберігання та управління даними, що дозволяє підтримувати їх узгодженість та надійність. Це важливо для гри «Підземелля і дракони», де інформація про персонажі, сюжет та інші аспекти повинні бути доступні та актуальні для всіх учасників гри.

Розділення обов'язків: Клієнтська частина може бути оптимізована для конкретного типу пристрою (наприклад, комп'ютер, планшет, смартфон), тим

самим забезпечуючи оптимальний досвід користувача на кожному з них. У той же час, серверна частина може керувати бізнес-логікою та забезпечувати однакову функціональність для всіх клієнтів.

Безпека: Клієнт-серверна архітектура дозволяє реалізувати рівень безпеки, контролюючи доступ до серверних ресурсів та обмежуючи можливості злоумисників. Це особливо важливо для ігрових платформ, де конфіденційність та цілісність даних є пріоритетом.

З урахуванням цих факторів та потреб користувачів розробка платформи для ведення гри «Підземелля і дракони» на базі клієнт-серверної архітектури видається оптимальним та перспективним рішенням.

Для розробки органайзера ведучого гри «Підземелля і дракони» було використано стек MERN, який включає в себе MongoDB як базу даних, Express.js [6] як серверну рамку, React.js [9] для розробки клієнтської частини та Node.js [17] для серверного середовища виконання JavaScript [16].

На серверній частині було використано додаткові інструменти для забезпечення безпеки, зберігання даних та автентифікації. Для генерації та перевірки токенів доступу було використано бібліотеку jsonwebtoken [2]. Для обробки файлів, зокрема для завантаження зображень, було використано Multer [15]. Для хешування паролів користувачів та забезпечення безпеки їх зберігання в базі даних було використано бібліотеку Bcrypt [10]. Для зручної взаємодії з базою даних MongoDB було використано об'єктно-документний моделювальник Mongoose.

На клієнтській частині було використано різноманітні інструменти та бібліотеки для створення зручного та ефективного користувацького інтерфейсу. Bootstrap було використано для швидкої розробки стильового дизайну та компонентів. Yup [3] було використано для валідації даних на клієнтській стороні, що допомагає забезпечити правильність введених користувачем даних. React-router-dom було використано для маршрутизації в клієнтській частині, дозволяючи легко навігувати між різними сторінками

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В				18
Змн.	Арк.	№ докум.	Підпис	Дата		

додатку. Axios було використано для здійснення HTTP-запитів з клієнтської частини, що дозволяє взаємодіяти з сервером та отримувати та надсилати дані.

Цей стек технологій дозволяє розробляти ефективний та функціональний продукт, забезпечуючи високу швидкість розробки та зручний користувацький досвід. Кожен інструмент та бібліотека використовується з певною метою, що дозволяє покращити роботу як клієнтської, так і серверної частини додатку [26].

Висновки до першого розділу

У даному розділі було ретельно розглянуто технічне завдання на розробку органайзера ведучого гри «Підземелля і дракони». Було визначено ціль та мету проекту, а також відзначено актуальність даного напрямку в сучасному світі настільних рольових ігор.

Було проведено порівняльний аналіз існуючих платформ для ведення настільних рольових ігор, зокрема Roll20, Fantasy Grounds та Astral TableTop, що надають інструменти для гри «Підземелля і дракони». Це дозволило виявити їх переваги та недоліки, а також визначити необхідний функціонал для розроблюваного органайзера.

На основі аналізу вимог користувачів та системи, було обрано клієнт-серверну архітектуру для реалізації органайзера. Було описано інструменти та технології, які будуть використовуватися для розробки веб-інтерфейсу та серверної частини.

Цей розділ закладає міцну основу для подальшого проектування та розробки органайзера ведучого гри «Підземелля і дракони», забезпечуючи глибоке розуміння вимог та технологій, необхідних для досягнення поставлених цілей.

		Борчаківський А.С.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ОРГАНАЙЗЕРА ВЕДУЧОГО ГРИ «ПІДЗЕМЕЛЛЯ І ДРАКОНИ»

2.1 Опис акторів та алгоритмів роботи

Проектування органайзеру було вирішено почати з аналізу активностей та створення діаграм використання.

Основні актори (табл. 2.1) це користувачі з наступними ролями: гравець, ведучий, адміністратор.

Таблиця 2.1

Користувачі

Актор	Короткий опис
Гравець	Має можливість створювати персонажів, ворогів, переглядати запрошення та приймати їх
Ведучий	Має можливість створювати персонажів та ворогові, створювати сесії та запрошення до них. Додавати ворогів до сесії та проводити битви
Адміністратор	Має можливість переглядати нових ворогів та затверджувати або видаляти новостворених ворогів.

Кожен з акторів може взаємодіяти з іншими акторами використовуючи WEB-інтерфейс додатку, а також надсилати запити на сервер та отримувати відповіді.

Першою діаграмою (рис. 2.1) буде розроблятися діаграма активності реєстрації нового користувача [12]. Дана діаграма показує, що при реєстрації в системі користувач має перейти до вікна авторизації, де у випадку, коли у користувача ще не має облікового запису, він переходить до вкладки реєстрації. Після переходу користувач має ввести свої дані для внесення їх в систему та збереження даних про нього. Далі користувач натискає кнопку

«Зареєструватися» та дані відправляються на сервер. Після реєстрації користувач буде мати можливість вводити свої дані на сторінці авторизації та використовувати усі можливості, що надає сервіс для користувачів, які мають обліковий запис та авторизувалися в нього.

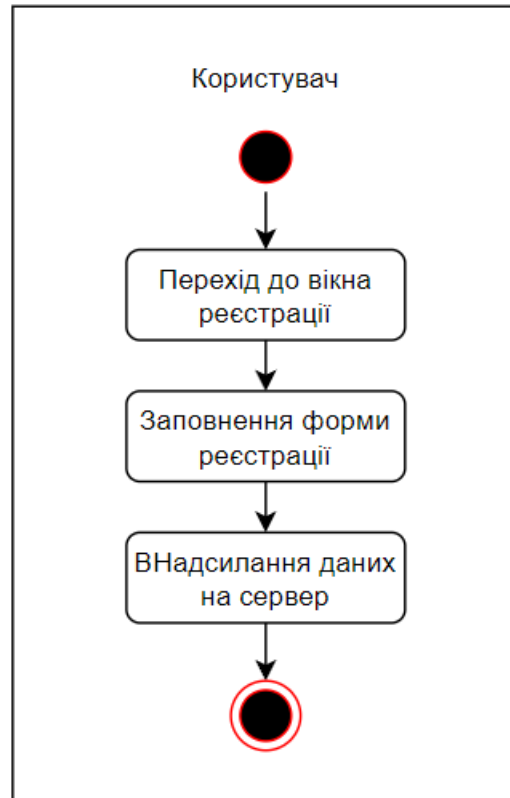


Рис. 2.1 – Діаграма активності для реєстрації нового користувача

Діаграма активності для створення ворога та затвердження його адміністратором (рис. 2.2) показує, що створення нового ворога ініціюється користувачем. Для цього, користувачу потрібно потрапити на сторінку створення ворогів, де він має ввести дані про ворога в запропоновану форму, та натиснути кнопку «Додати ворога». Після обробки запиту на сервері новостворені вороги потрапляють на сторінку адміністратор, де він може переглянути усі дані про ворогів та вирішити, чи є якісь дані, що порушують правила сервісу. Прийнявши рішення, адміністратор може натиснути відповідну кнопку та вирішити, чи буде ворог доданий в загальне користування. Після прийняття адміністратором ворога вінд починає відображатися для усіх користувачів, а ведучі можуть додати ворогові до своїх сесій.

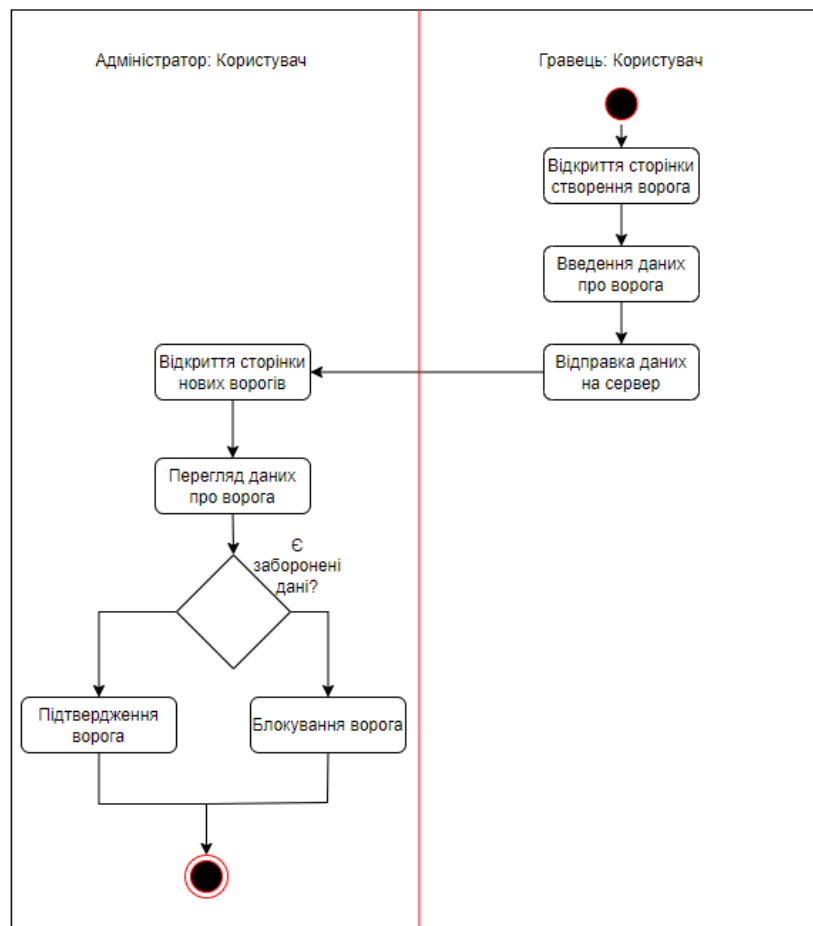


Рис. 2.2 – Діаграма активності створення та затвердження ворогів

Розглянувши діаграму активності (рис. 2.3) створення сесії видно, що ведучий ініціює створення сесії. Для створення сесій він повинен натиснути кнопку «Створити сесію». Далі користувач вводить назву сесії та переходить до сторінки створення запрошень. На даній сторінці користувач може надіслати до 4х запрошень користувачам, використовуючи їх імена користувача. Неможливо створити 2 запрошення для одного користувача, у разі спроби виникне повідомлення, що запрошення уже існує. Після надсилань запрошень користувач повинен натиснути кнопку «Створити сесію» і сесія буде створена. Усі створені сесії можна буде побачити на головній сторінці, де в верхній її частині вони і знаходяться.

Таким чином було розроблено три діаграми активності для основних дій, що можуть бути виконані актори, що приймають участь в роботі з сервісом.

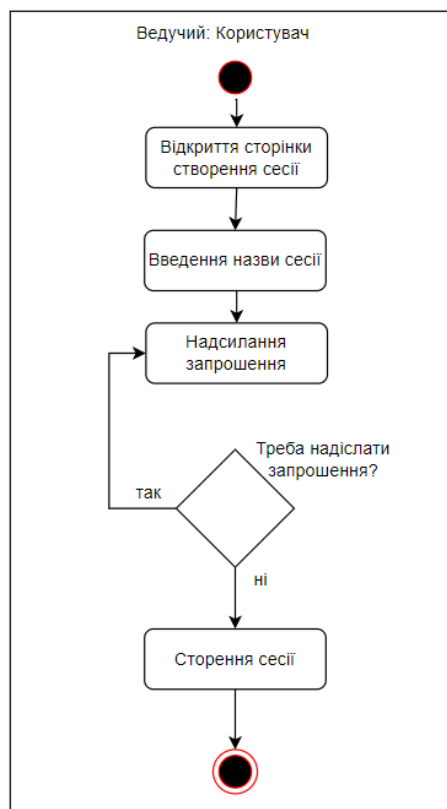


Рис. 2.3 – діаграма активності створення сесії

2.2 Розробка бази даних

Після опису акторів було прийнято рішення перейти до розробки БД, в якій будуть зберігатися усі потрібні записи. Розроблена база даних складається з 6 таблиць, кожна з яких відповідає за збереження своїх даних (табл. 2.2).

Таблиця 2.2

Опис таблиць

Назва таблиці	Короткий опис
Character	Зберігає дані про персонажа, що створив користувач
User	Зберігає усі дані про користувача
Invite	Зберігає запрошення до сесій для користувачів
Session	Зберігає дані про сесію та її учасників
Enemy	Зберігає дані про ворогів
EnemySession	Зберігає дані про ворогів та зв'язує їх з сесією

Кожна таблиця була створена з використанням бібліотеки-драйвера, для роботи з базою даних MongoDB, що дозволило змінювати дані про таблиці прямо в процесі розробки використовуючи програмний синтаксис.

Звернувши увагу на діаграму (рис. 2.4), можна побачити усі таблиці БД та їх зв'язки. Описуючи зв'язки більш детально, можна побачити, що такі таблиці як Invite, Session та User мають по декілька зв'язків, в тому числі між собою. З точки зору таблиці Invite видно, що вона пов'язана з таблицею User, оскільки зберігає ідентифікатор користувача, для якого було створено запрошення та з таблицею Session, оскільки зберігає ідентифікатор сесії, до якої це запрошення створене. Таблиця Users також має зв'язок з таблицею Character, що дає змогу швидко отримувати дані до усіх персонажів користувачів. Також таблиця Session має зв'язок з таблицею EnemySession, завдяки чому в сесії завжди можна звернутися до усіх ворогів, що були до неї додані.

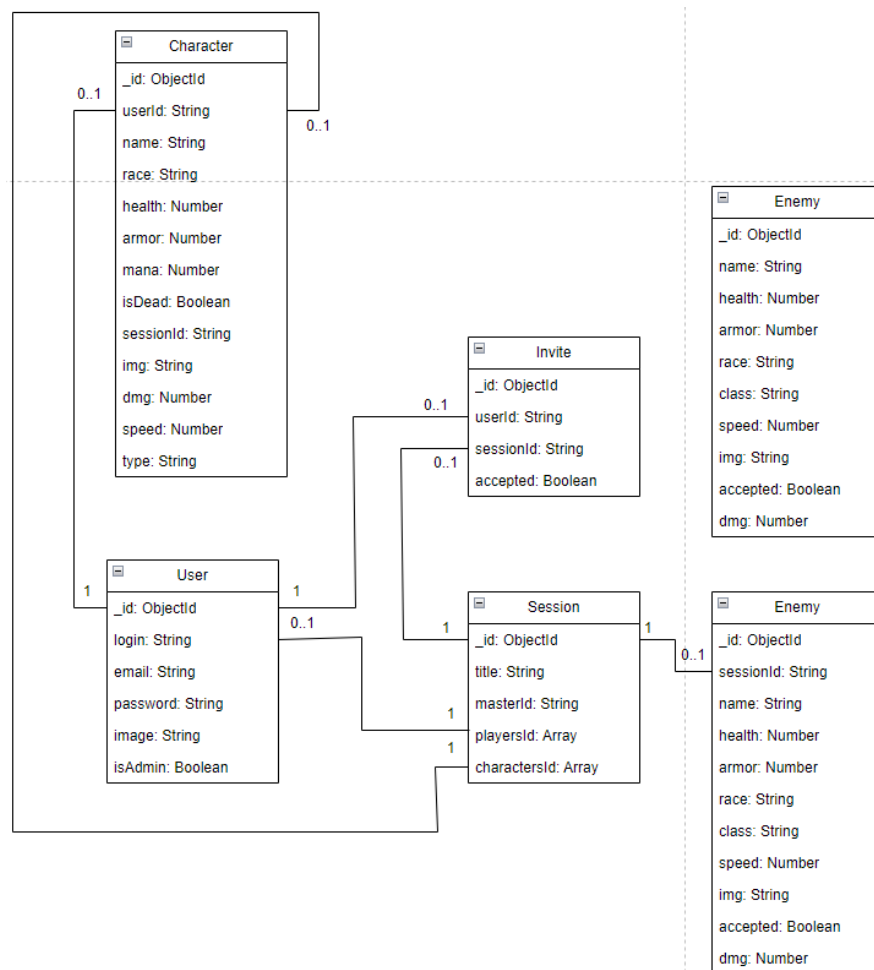


Рис. 2.4 – Візуалізація БД

Для більш детального розуміння структури БД перейдемо до детального розгляду кожної таблиці, починаючи з таблиці Character (табл. 2.3). В даній таблиці організовано зберігання усіх даних, про персонажів, яких створюють користувачі. У тому числі тут організований зв'язок між персонажем та сесією, в якій він приймає участь.

Таблиця 2.3

Структура таблиці “Characters”

Characters			
Назва поля	Тип даних	PK/FK	Опис поля
_id	ObjectId	PK	Зберігає унікальний ідентифікатор персонажу
userId	String	FK	Зберігає унікальний ідентифікатор користувача
name	String	-	Зберігає ім'я, надане користувачем
race	String	-	Зберігає назву раси, надану користувачем
health	Number	-	Зберігає кількість здоров'я персонажа
armor	Number	-	Зберігає кількість броні персонажа
mana	Number	-	Зберігає кількість мани персонажа
isDead	Boolean	-	Прапорець, що відслідковує смерть персонажа
sessionId	String	FK	Ідентифікатор сесії, до якої прив'язаний персонаж
img	String	-	Назва фотографії персонажа
dmg	Number	-	Кількість шкоди персонажа
speed	Number	-	Швидкість персонажа
type	String	-	Тип запису

Далі розглянемо таблицю Users (табл. 2.4), яка створена для збереження усіх даних про користувачів сервісу, таких як фото профілю, логін, хеш

пароллю тощо. Дана таблиця пов’язана з таблицею з персонажами, таблицею з сесіями та таблицею з запрошеннями, оскільки усі ці таблиці мають посилання на користувача, яких до них відноситься.

Таблиця 2.4

Структура таблиці “Users”

Users			
Назва поля	Тип даних	PK/FK	Опис поля
_id	ObjectId	PK	Зберігає унікальний ідентифікатор користувача
login	String	-	Зберігає логін користувача
email	String	-	Зберігає пошту користувача
password	String	-	Зберігає хеш пароллю користувача
image	String	-	Зберігає назву фото профілю користувача
isAdmin	Boolean	-	Прапорець, що вказує на адміністратора

Звертаючи увагу таблицю Invite (табл. 2.5) бачимо, що вона розроблена для збереження запрошень від майстра до користувача. Таблиця зберігає ідентифікатор користувача, що отримує запрошення, та ідентифікатор сесії, до якої це запрошення було отримане.

Таблиця 2.5

Структура таблиці “Invite”

Invite			
Назва поля	Тип даних	PK/FK	Опис поля
_id	ObjectId	PK	Зберігає унікальний ідентифікатор запрошення
userId	String	FK	Зберігає ідентифікатор користувача, що отримав запрошення

sessionId	String	FK	Зберігає ідентифікатор сесії, до якої направляє запрошення
accepted	Boolean	-	Прапорець, що вказує чи було прийняте запрошення

Перейдемо до розгляду таблиці Session (табл. 2.6), що зберігає усі створені сесії. Дана таблиця підтримує зв'язок між сесіями, учасниками сесій та персонажами, що прив'язані до сесії.

Таблиця 2.6

Структура таблиці "Session"

Session			
Назва поля	Тип даних	PK/FK	Опис поля
_id	ObjectId	PK	Зберігає унікальний ідентифікатор сесії
title	String	-	Зберігає назву сесії
masterId	String	FK	Зберігає унікальний ідентифікатор ведучого
playersId	Array	FK	Зберігає унікальні ідентифікатори усіх учасників сесії
charactersId	Array	FK	Зберігає унікальні ідентифікатори усіх персонажів в сесії

Надалі варто звернути увагу на таблицю Enemy, що зберігає усіх створених користувачами ворогів, яких в майбутньому можна додати до сесії. Дана таблиця не має зв'язків з іншими таблицями в БД, оскільки є посередником для збереження даних про ворогів. В свою чергу таблиця EnemySession (табл. 2.7) копіює дані з таблиці Enemy і додає ідентифікатор сесії, в якій ворог використовується. Для опису буде використана лише таблиця EnemySession, оскільки таблиці відрізняються лише одним полем sessionId.

Структура таблиці “EnemySession”

EnemySession			
Назва поля	Тип даних	PK/FK	Опис поля
_id	ObjectId	PK	Унікальний ідентифікатор запису
sessionId	String	FK	Унікальний ідентифікатор сесії
name	String	-	Зберігає ім'я ворога
health	Number	-	Зберігає кількість здоров'я ворога
armor	Number	-	Зберігає кількість броні ворога
race	String	-	Зберігає расу ворога
class	String	-	Зберігає клас ворога
speed	Number	-	Зберігає швидкість ворога
img	String	-	Зберігає назву фотографії ворога
accepted	Boolean	-	Прапорець, що вказує чи було одобрено ворога
dmg	Number	-	Зберігає кількість шкоди ворога

2.3 Комунікації і послідовність взаємодії об'єктів органайзера

Після розробки діаграми активності та бази даних переходимо до етапу розробку діаграм послідовностей, що дозволять побачити як саме відбуваються комунікації систему [18].

Переглянувши діаграму послідовності створення сесії (рис. 2.5) видно, що все починається з того, що користувач має заповнити форму с даними про сесію. Після заповнення даних користувач натискає кнопку та надсилає дані на сервер. Сервер обробляє дані та відправляє в БД запит на створення запису. Отримавши відповідь від БД сервер обробляє його та передає клієнту. Оскільки сервіс розробляється з використанням стеку MERN шаблонізатором виступає React, а запити на сервер представляють собою запити на API.



Рис. 2.5 – діаграма послідовності створення сесії

Звертаючи увагу на діаграму послідовності створення ворогів та передачі їх адміністратору (рис. 2.6), можна побачити, що для створення нового ворога користувач вводить дані про ворога та натискає відповідну кнопку, що відправляє ці дані на сервер. На сервері дані обробляються та передаються до БД, звідки приходить відповідь. Отримавши відповідь від БД сервер надсилає відповідь уже користувачу. Далі, коли адміністратор переходить на сторінку для підтвердження ворогів, клієнт відправляє запит на сервер про отримання непідтверджених ворогів, сервер отримує список від БД і надсилає їх на клієнтську частину.

Таким чином на даному етапі розробки було визначено як саме сервер і клієнт мають проводити комунікацію в описаних на діаграмах випадках. Зважаючи на отримані з діаграм дані надалі буде проведено розробку сервісу.

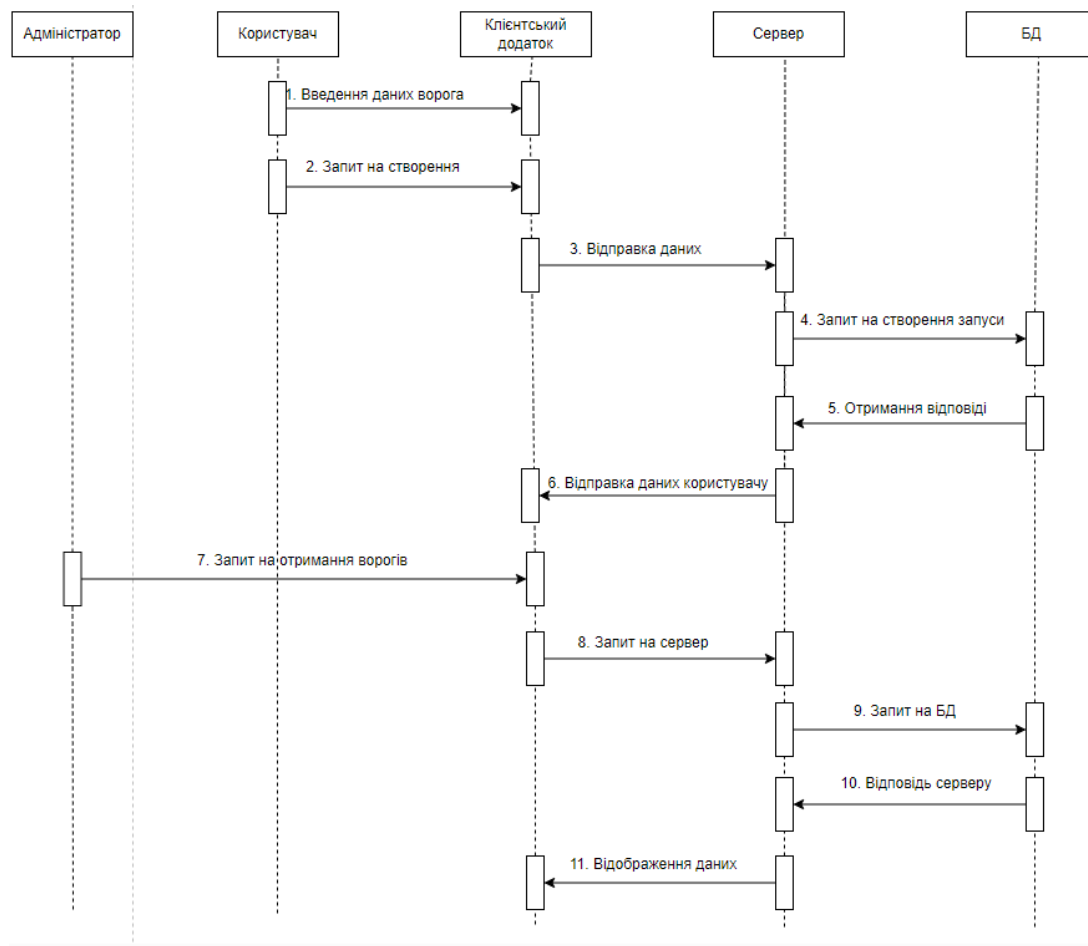


Рис. 2.6 – діаграма послідовності створення і затвердження ворогів

2.4 Реалізація органайзера ведучого гри «Підземелля і дракони»

Поставивши ТЗ до проекту, а також проаналізувавши вимоги до акторів, їх активностей та визначивши послідовності використання сервісу, приймаємо рішення переходити до реалізації програмної частини проекту.

Реалізацію проекту вирішено розпочинати з розробки серверної частини, оскільки зв'язок між клієнтом і сервером відбувається шляхом відправки запитів з клієнта на API.

```

router.post("/new-character", auth, async (req, res) => {
  try{
    const newCharacter = new Character({
      userId: req.user.userId,
      name: req.body.name,
      race: req.body.race,
      health: req.body.health,
      mana: req.body.mana,
      armor: req.body.armor,
      class: req.body.class,
      img: 'char.jpg',
      dmg: req.body.dmg,
    })
  }
})
  
```

```

        speed: 1
      })
      const character = await newCharacter.save()
      res.status(200).json({message:"Персонажа створено"})
    } catch(err){
      res.status(500).json(err)
    }
  })
})

router.get('/', auth, async (req, res) => {
  try{
    const characters = await Character.find({userId: req.user.userId})
    console.log(characters)
    res.status(200).json(characters)
  }catch(err){
    res.status(500).json(err)
  }
})

```

Звертаючи увагу, на програмний код, що представлено вище можна побачити принцип відпрацювання запитів. Прикладом, яким представлений код, виступає реалізація створення персонажів та отримання усіх персонажів, що належать користувачу. Видно, що в першу чергу відбувається створення об'єкту моделі БД, про якій буде розказано нижче. В кожне поле об'єкта записуються дані, що приходять в запиті користувача. Після створення об'єкту відбувається його збереження до БД, а користувач отримує від серверу відповідь. Таким чином представлені запити, що використовуються для додавання записів до БД. Далі показано код, що використовується для отримання записів. Сервер надсилає до БД запит, в якому вказується таблиця до якої потрібно звернутися, а також фільтрація по даним, які саме нас влаштовують. Конкретно в прикладі відбувається пошук персонажів за ідентифікатором користувача, що їх створив.

```

const mongoose = require("mongoose")

const SessionSchema = new mongoose.Schema({
  title:{
    type:String,
    required: true
  },
  masterId:{
    type:String,
    required:true
  },
})

```

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				31
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    playersId:{
      type: Array,
      default: []
    },
    charactersId:{
      type: Array,
      default: []
    }
  })

module.exports = mongoose.model("Session", SessionSchema)

```

Далі було наведено код, який представляє собою створення моделі в БД. Для створення моделей використовується драйвер-бібліотека MongoDB, завдяки чому БД розробляється з використанням синтаксису мови Node.js.

Далі створюється схема даних, в яку ми записуємо назви полів та вимоги, яким мають відповідати дані, що будуть заноситися до таблиці. Наприклад поле title обов'язково має бути рядкового типу даних і має бути занесено при створенні запису, в той час як поле playersId повинно бути масивом і має значення за замовчуванням.

```

import React, { useState } from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Button, Modal } from 'react-bootstrap';
import { Registration } from './registration';
import { Authorization } from './authorization';

export const AuthModal = () => {
  const [show, setShow] = useState(false);
  const [isRegistration, setIsRegistration] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);
  const switchMode = () => setIsRegistration(!isRegistration);

  return (
    <>
      <Button variant="light" onClick={handleShow}>
        Увійти
      </Button>

      <Modal show={show} onHide={handleClose}>
        <Modal.Header>
          <Modal.Title>{isRegistration ? "Реєстрація" :
"Авторизація"}</Modal.Title>
        </Modal.Header>
        <Modal.Body>

```

```

        {isRegistration ? <Registration setIsRegistration={setIsRegistration}/>
: <Authorization/>}
      </Modal.Body>
      <Modal.Footer>
        <Button variant="secondary" onClick={switchMode}>
          {isRegistration ? "Авторизація" : "Реєстрація"}
        </Button>
      </Modal.Footer>
    </Modal>
  </>
);
}

```

В представленому коді клієнтської частини видно, як реалізовується архітектура додатків створених на реакт. Кожна частина інтерфейсу розділена на окремі компоненти і в залежності від стану, який встановлюється в додатку, на одних і тих же місцях інтерфейсу можуть відображатися різні дані. В конкретному прикладі видно, що у разі, якщо користувач встановив стан реєстрації, то для нього відображається форма реєстрації, інакше, якщо стан реєстрації встановлено не було, користувач отримує форму для введення даних уже існуючого облікового запису, тобто форму авторизації.

Таким чином було розроблено повний код сервісу, завдяки чому були реалізовані усі поставлені в ТЗ задачі.

Висновки до другого розділу

У цьому розділі був проведений аналіз усіх акторів, що мають взаємодіяти з системою, та були розроблені діаграми використання сервісу цими акторами. Наступним кроком було розроблення БД, визначення потрібних таблиць та полів, що мають там бути. Далі були проаналізовані і розроблені діаграми послідовностей роботи з додатком. Розділ було закінчено написанням коду, тобто реалізацією сервісу.

Таким чином, у цьому розділі було проведено повний цикл проектування органайзеру, від аналізу вимог до розробки бази даних і програмного коду, що дозволило реалізувати всі необхідні функціональні можливості сервісу.

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В				33
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОРГАНАЙЗЕРОМ ВЕДУЧОГО ГРИ «ПІДЗЕМЕЛЛЯ І ДРАКОНИ»

3.1 Порядок встановлення та налаштування органайзера

Щоб зрозуміти, як саме відбувається взаємодія між компонентами органайзера, була розроблена діаграма компонентів (рис. 3.1), що чітко передає взаємозалежність між різними компонентами системи.

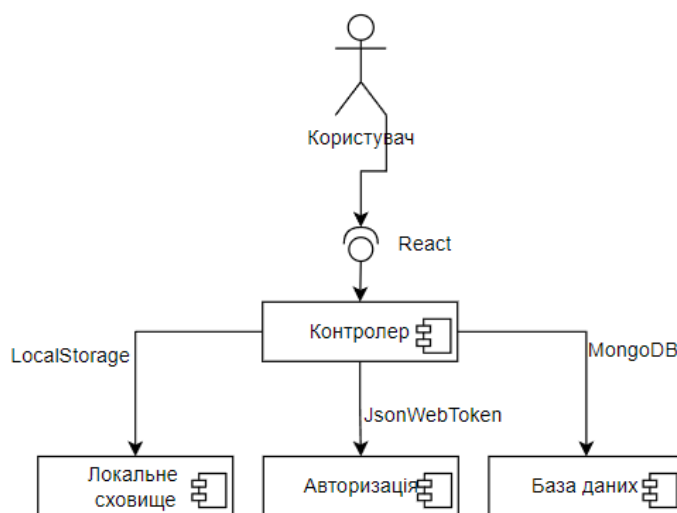


Рис. 3.1 – Діаграма компонентів

Сервіс отримує дані безпосередньо від користувачів. Усі запити, які надходять до сервера, ініціюються користувачем через клієнт на React. Користувач взаємодіє з сервером, надсилаючи запити до контролера. Якщо контролеру потрібно зберегти або змінити записи в базі даних, він звертається до компонента БД. Для авторизації користувачів контролер використовує компонент авторизації, який працює з JSONWebToken. Якщо необхідно зберегти файли (наприклад, зображення ворогів), контролер звертається до компонента локального сховища, який відповідає за збереження файлів. Таким чином, забезпечується взаємодія між усіма компонентами сервісу.

Додатково потрібно зрозуміти принцип розгортання проекту, для чого було розроблено діаграму розгортання (рис. 3.2)

На діаграмі показано, як користувач, використовуючи комп'ютер і підключення до Інтернету, може обмінюватися даними з сервером. Сервер, обробляючи інформацію, отриману в запитах від клієнта, звертається до

відповідних компонентів. Локальне сховище, яке є частиною серверу, зберігає всі файли, завантажені користувачами. База даних розташована віддалено, і сервер сервісу взаємодіє з нею за допомогою спеціального драйвера.

Проаналізувавши діаграму розгортання, можна зробити висновок, що для запуску сервісу необхідно використовувати хостинг, який забезпечує місце для зберігання файлів у локальному сховищі. Оскільки база даних розташована віддалено, її не потрібно розгортати разом із сервером. Однак важливо переконатися, що хостинг дозволяє серверу виконувати запити до зовнішніх джерел.

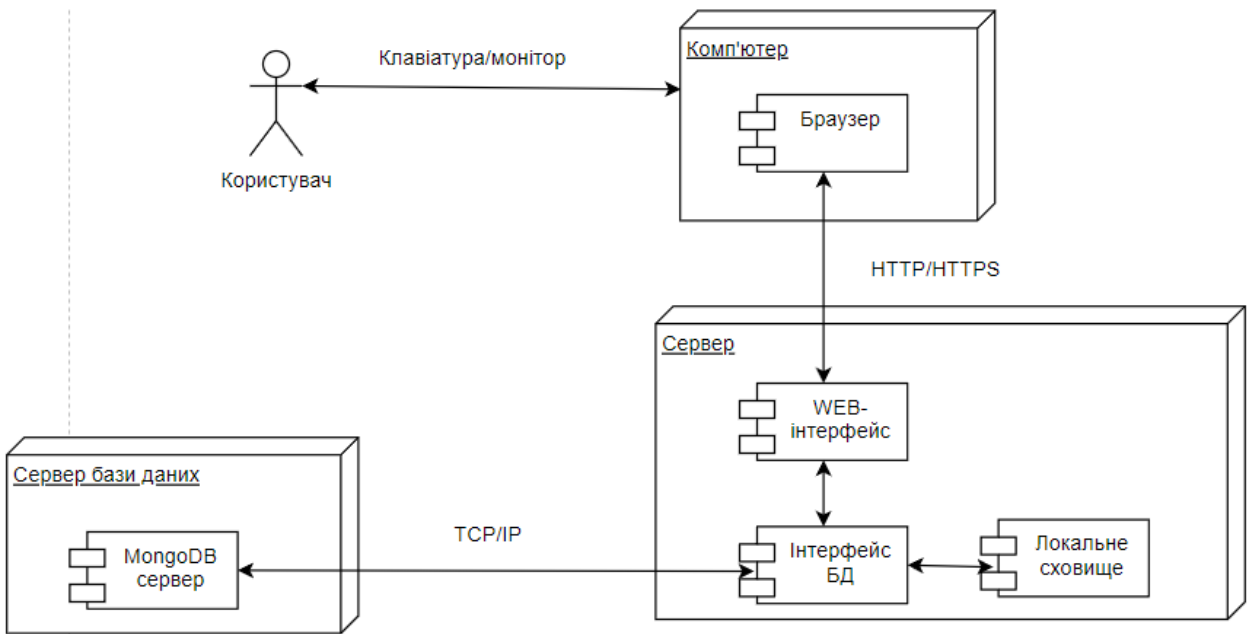


Рис. 3.2 – Діаграма розгортання

3.2 Опис інтерфейсу програмного продукту

При переході на сайт користувач отримає можливість взаємодіяти з інтерфейсом додатку лише після того, як авторизується в систему.

Для авторизації використовується кнопка «Увійти», що знаходиться в шапці сайту (рис. 3.3).



Рис. 3. 3 – Шапка сайту

Поки користувач не увійде до системи, шапка буде мати мінімальний функціонал. Логотип та кнопка для авторизації та реєстрації. Після авторизації користувач отримає більший функціонал в шапці (рис. 3.4).



Рис. 3.4 – Шапка авторизованого користувача

Після авторизації користувач отримує можливість бачити кнопку створення сесій та іконку профілю, що перенаправляє користувача до сторінки профілю. Також з'являється значок сповіщень, де користувач зможе побачити кількість отриманих запрошень до сесій.

Якщо звертати увагу на процес створення сесій, то після натиснення на відповідну кнопку з'явиться модальне вікно (рис. 3.5), де користувачу потрібно буде ввести назву сесії, яку він хоче створити.

Рис. 3.5 – Вікно створення сесії

Ввівши назву сесії та натиснувши кнопку створити, користувач потрапляє до вікна (рис. 3.6)), де він має можливість надіслати запрошення користувачам. При першому створенні є можливість надіслати до 4х запрошень, оскільки максимальний розмір сесії 4 учасники, у разі якщо якесь із запрошень не буде прийнято, нове можна створити на сторінці сесії.

Рис. 3.6 – Вікно надсилання запрошень

Побачити список створених сесій користувач може на головній сторінці (рис. 3.7), там само знаходяться картки ворогів, а також форма додавання нових ворогів.

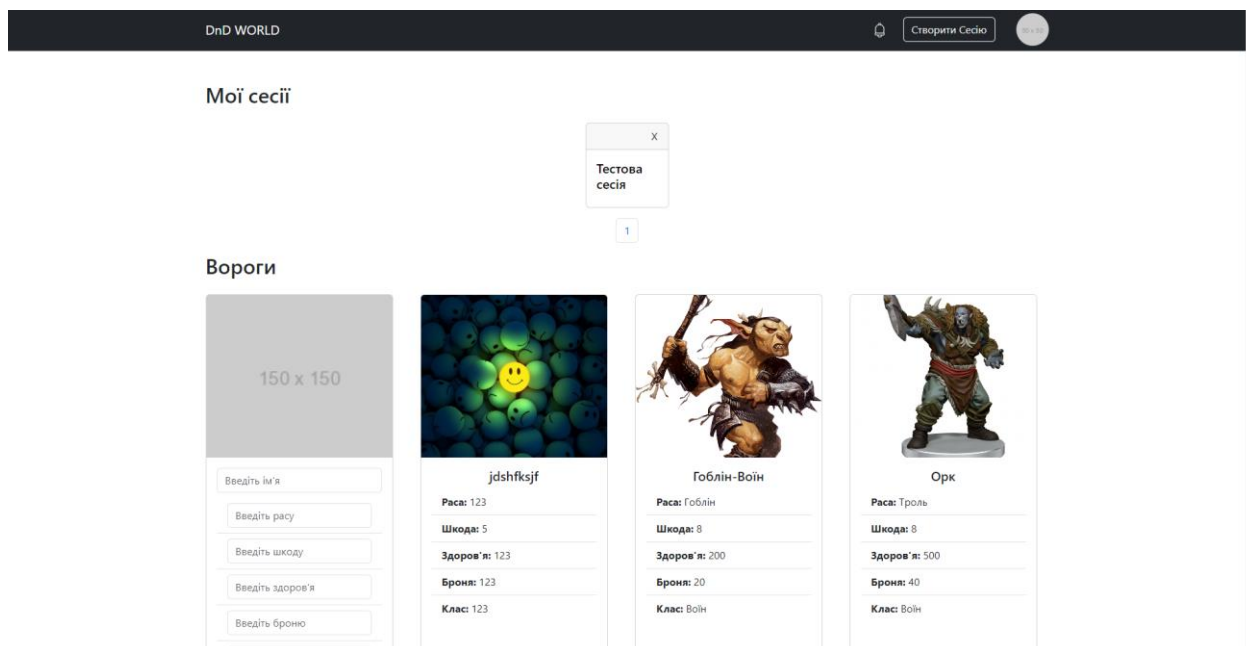


Рис. 3.7 – Головна сторінка

Звертаючи увагу на картку сесії (рис. 3.8) користувач може побачити назву сесії, на сторінку якої він може потрапити, а також хрестик, при натискання на який відбудеться видалення сесії.

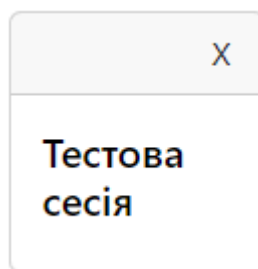


Рис. 3.8 – Картка сесії

Згадуючи, що на головному екрані також знаходиться картка додавання нового ворога, можна побачити, що дана картка представляє собою форму для вводу даних про ворога. Також на картці знаходиться картинка, що символізує відсутність завантаженої картинки. При натисканні з'являється можливість завантажити фото, яке одразу ж відобразиться на картці.

150 x 150

Введіть ім'я

Введіть расу

Введіть шкоду

Введіть здоров'я

Введіть броню

Введіть клас

Створити ворога

Рис. 3.9 – Картка додавання ворога

На головній сторінці також знаходяться картки уже створених ворогів. У їх представленні потреба відсутня, оскільки всі картки ворогів ідентичні картці додавання нового ворога.

Натиснувши на фото профілю в шапці, користувач може потрапити до сторінки власного профілю (рис. 3.10), де у користувача є можливість вийти з облікового запису, або ж створити власного ігрового персонажа.

DnD WORLD

Створити Сесію

150

Введіть ім'я

Введіть расу

Введіть шкоду

Введіть здоров'я

Введіть броню

Введіть клас

Введіть ману

Створити персонажа

Вийти

Рис. 3.10 – Сторінка профілю

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				38
Змн.	Арк.	№ докум.	Підпис	Дата		

Розглядаючи сторінку профілю можна звернути увагу, на картку створення персонажа. Як і картка створення ворога вона представляє собою форму та кнопку, яка надсилає дані на сервер. Єдина різниця є в тому, що користувачі не додають фото профілю для своїх персонажів.

Усі створені персонажі відображаються біля картки створення (рис. 3.11).

Рис. 3.11 – Сторінка профілю зі створеним персонажем

Повертаючись до питання сесій, користувач може прийняти запрошення через вікно (рис. 3.12), що відкривається при натисканні на значок сповіщення в шапці. Натиснувши кнопку «Прийняти» користувачу буде запропоновано додати до сесії персонажа (рис. 3.13,) а при натисканні кнопки «Відхилити» запрошення буде видалено

Рис. 3.12 – Вікно запрошень

Оберіть персонажа для гри



Грибок

До запрошень

Рис. 3.13 – Вікно додавання персонажа

З точки зору майстра, після того як усі гравці прийняли запрошення до сесії, є сенс перейти на сторінку сесії (рис. 3.13). На даній сторінці видно усі картки персонажів, також ведучий може змінити параметри шкоди, здоров'я і броні, якщо вони не відповідають контексту сесії.

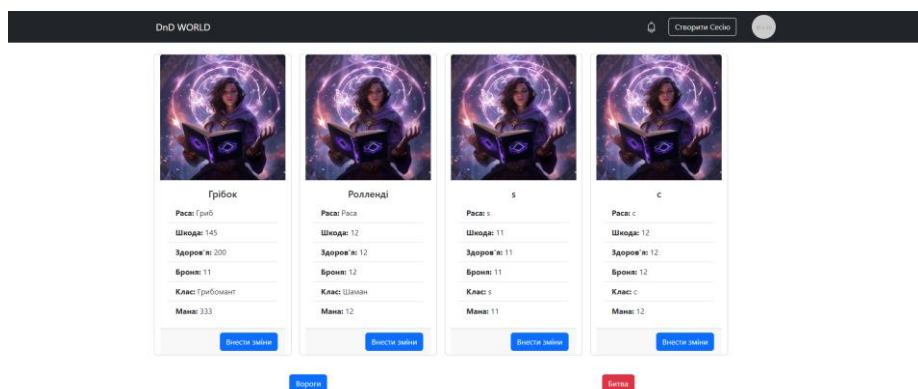


Рис. 3.14 – сторінка сесії

На даній сторінці також знаходяться дві кнопки: вороги та битва.

При натисканні на кнопку «вороги» користувач відкриває вікно (рис. 3.15), де він може переглянути ворогів та додати їх до сесії, якщо є така потреба. Також у вікні додавання ворогів до сесії реалізована фільтрація по імені.

У вікні (рис. 3.16), що відкривається по натисканню на кнопку «битва», користувач може запустити бій, де всі члени сесії і вороги будуть вистроєні по порядку ходу. При натисканні на кнопку «Нанести удар» ведучий вибирає, кого буде атакувати перший персонаж у черзі.

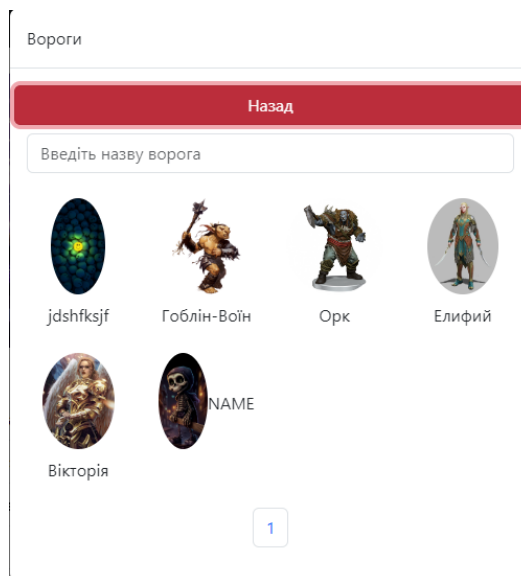


Рис. 3.15 – Вікно з ворогами

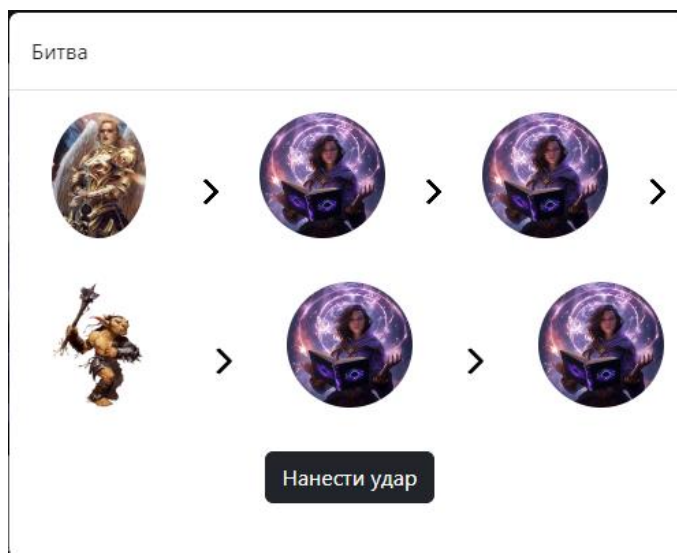


Рис. 3.16 – Вікно битви

3.3 Тестування організера

Наступним кроком розглянемо процес тестування розробленого сервісу, який є важливою складовою забезпечення його якості та надійності. Основною метою тестування є виявлення можливих помилок та несправностей у роботі системи до її впровадження в експлуатацію. Тестування проводиться вручну шляхом створення та виконання тест-кейсів, які охоплюють різні аспекти функціональності сервісу.

Ручне тестування передбачає ретельне перевіряння всіх ключових компонентів та процесів сервісу, зокрема його взаємодію з користувачем, обробку запитів, роботу з базою даних та збереження файлів у локальному

сховищі. Використання тест-кейсів дозволяє систематизувати процес тестування, забезпечуючи покриття всіх можливих сценаріїв використання сервісу. У цьому розділі буде детально описано підхід до створення тест-кейсів, критерії їхнього виконання та результати тестування.

Таблиця 3.1

Тестування організатору

Опис тесту	Кроки тестування	Дані тестування	Очікуваний результат	Фактичний результат	Пройдено / провалено
Перевірка авторизації	1. Відкрити вікно авторизації 2. Ввести дані користувача 3. Натиснути кнопку «Увійти»	Email: soup.andrey@gmail.com password: 11111111	Відбувається успішна авторизація	Відбувається успішна авторизація	Пройдено
Видалення сесії	1. Відкрити головну сторінку 2. Натиснути на кнопку видалення сесії	-	Сесія видаляється	Сесія видаляється	Пройдено
Перевірка видалення персонажа	1. Відкрити сторінку профілю 2. Натиснути кнопку «Видалити персонажа»	-	Персонаж видаляється	Персонаж видаляється	Пройдено
Перевірка додавання ворога до сесії	1. Відкрити сторінку сесії 2. Відкрити сторінку ворогів 3. Додати ворога	Title: гоблін	Ворог додається до сесії	Ворог додається до сесії	Пройдено

		Борчаківський А.С.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Концедайло В.В.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

Продовження таблиці 3.1

Перевірка прийняття запрошення	1. Відкрити вікно з запрошеннями 2. Обрати потрібне запрошення 3. Натиснути кнопку «Прийняти»	-	Відкривається вікно вибору персонажа	Відкривається вікно вибору персонажа	Пройдено
--------------------------------------	---	---	--	--	----------

Висновки до третього розділу

В розділі було розглянуто та встановлено порядок налаштування сервісу, були розроблені діаграми компонентів та розгортання, для розуміння архітектури та взаємодії компонентів. Також було розглянуто інтерфейс користувача та була надана інструкція до нього. Розділ було закінчено проведенням тестів, що дозволили виявити помилки, що були допущені на етапі розробки.

ВИСНОВКИ

В першому розділі було глибоко проаналізовано технічні вимоги до розробки органайзеру ведучого гри «Підземелля і дракони», встановивши ціль та мету проекту та підкресливши актуальність даного напрямку в галузі настільних рольових ігор. Чітко визначивши переваги та недоліки існуючих сервісів, було зроблено вибір функціоналу для майбутньої системи. Обрана клієнт-серверна архітектура та використання відповідних технологій встановлюють тверду основу для подальшої розробки.

В другому розділі був проведений повний цикл проектування органайзеру, від аналізу вимог до реалізації сервісу. Були визначені взаємодії між акторами, розроблена база даних та діаграми послідовностей роботи. Було успішно реалізовано функціонал через написання програмного коду.

В третьому розділі було завершено проект, шляхом налаштування сервісу, аналізу архітектури та інтерфейсу. Були проведені тести, що дозволили виключити можливі помилки в роботі сервісу.

Результатом роботи є спроектований та розроблений органайзер ведучого гри «Підземелля і дракони», який відповідає усім вимогам, що були поставлені. В процесі виконання завдання були засвоєні знання, що отримувалися протягом навчання.

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				44
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NPM bootstrap [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/bootstrap>.
2. NPM jsonwebtoken [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/jsonwebtoken>.
3. NPM yup [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/yup>.
4. Shelley S. Learning Node: Moving to the Server-Side 2nd Edition / Shelley Shelley., 2016. – 288 с.
5. Як будувати UML-діаграми [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/forums/topic/40575/>.
6. Express documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>.
7. MacDonald M. HTML5: The Missing Manual, 2nd Edition The Book That Should Have Been in the Box / Matthew MacDonald., 2014. – 518 с.
8. React-router-dom tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/en/main/start/tutorial>.
9. React.js documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.legacy.reactjs.org/>.
10. NPM bcrypt [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/bcrypt>.
11. MERN Stack Explained [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/resources/languages/mern-stack>.
12. Types of UML Diagrams [Електронний ресурс] – Режим доступу до ресурсу: <https://www.lucidchart.com/blog/types-of-UML-diagrams>.
13. Діаграма розгортання [Електронний ресурс] – Режим доступу до ресурсу: <https://studfile.net/preview/5010027/page:6/>.
14. NPM Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/axios>.

		Борчаківський А.С.			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Конюдаїло В.В.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

15. NPM multer [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/multer>.
16. JavaScript Tutorial [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.w3schools.com/js/>.
17. Node.js documentation [Електронний ресурс] – Режим доступу до ресурсу:
<https://nodejs.org/docs/latest/api/>.
18. Діаграма послідовності [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.maxzosim.com/sequence-diagrams/>.
19. Robin W. The Road to React: Your journey to master plain yet pragmatic React.js / Wieruch Robin., 2018. – 286 с.
20. Banks A. Learning React: Modern Patterns for Developing React Apps / A. Banks, E. Porcello., 2020. – 307 с.
21. Gascón U. Node.js for Beginners: A comprehensive guide to building efficient, full-featured web applications with Node.js / Ulises Gascón., 2024. – 382 с.
22. Lim G. Beginning Node.js, Express & MongoDB Development / Greg Lim., 2019. – 154 с.
23. Flanagan D. Javascript: The Definitive Guide: Master the World's Most-Used Programming Language / David Flanagan., 2020. – 704 с.
24. Code Quickly. Learn JavaScript Quickly: A Complete Beginner's Guide to Learning JavaScript, Even If You're New to Programming (Crash Course With Hands-On Project) / Code Quickly., 2020. – 174 с.
25. Biswas N. Ultimate Full-Stack Web Development with MERN: Design, Build, Test and Deploy Production-Grade Web Applications with MongoDB, Express, React and NodeJS (English Edition) / Nabendu Biswas., 2023. – 294 с.
26. Subramanian V. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node / Vasan Subramanian., 2019. – 536 с.
27. NPM nodemon [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.npmjs.com/package/nodemon>.
28. Fantasy Grounds [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.fantasygrounds.com/>.

		Борчаківський А.С.			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Концедайло В.В.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

29. D&D Beyond [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.dndbeyond.com/>.

30. Roll20 [Електронний ресурс] – Режим доступу до ресурсу:
<https://roll20.net/>.

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				47
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				48
Змн.	Арк.	№ докум.	Підпис	Дата		

Вміст файлу Character.js

```

const mongoose = require("mongoose")

const CharacterSchema = new mongoose.Schema({
  userId:{
    type:String,
    required: true
  },
  name:{
    type:String,
    required: true
  },
  race:{
    type:String,
    required: true
  },
  health:{
    type:Number,
    required: true,
  },
  armor:{
    type:Number,
    required: true
  },
  class:{
    type:String,
    required: true
  },
  mana:{
    type:Number,
    required: true
  },
  isDead:{
    type:Boolean,
    default: false
  },
  img: {
    type: String,
    default: false
  },
  dmg: {
    type: Number,
    required: true
  },
  speed: {
    type: Number,
    require: true
  },
  sessionId:{

```

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				49
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        type:String,
        default: 'None'
    },
    type: {
        type:String,
        default: 'Character'
    }
})

module.exports = mongoose.model("Character", CharacterSchema)

```

Вміст файлу auth.js

```

const router = require('express').Router()
const User = require("../models/User")
const bcrypt = require("bcrypt")
const jwt = require("jsonwebtoken")

//Реєстрація
router.post("/register", async (req, res) => {
    const userMail = await User.findOne({email:req.body.email})
    const userLogin = await User.findOne({login:req.body.login})
    if(!userLogin && !userMail){
        try{
            const salt = await bcrypt.genSalt(10)
            const hashedPassword = await bcrypt.hash(req.body.password, salt)
            const newUser = new User({
                login: req.body.login,
                email: req.body.email,
                password: hashedPassword
            })

            const user = await newUser.save()
            res.status(201).json({message:"Обліковий запис створено"})
        } catch(err){
            console.log(err)
        }
    }
    else if(userMail){
        return res.status(400).json({message:"Користувач з такою поштою уже існує"})
    }
    else if(userLogin){
        return res.status(400).send({message:"Користувач з таким логіном уже існує"})
    }
})

//Авторизація

```

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

router.post("/login", async (req, res)=>{
  try{
    const user = await User.findOne({email:req.body.email})
    console.log(user)
    if(!user){
      return res.status(400).send({message:"Користувача з такою поштою не існує"})
    }
    const validPassword = await bcrypt.compare(req.body.password, user.password)
    if(!validPassword){
      return res.status(400).json({message:"Невірний пароль"})
    }

    const token = jwt.sign(
      {userId: user.id},
      "borchakivskyikey",
      {expiresIn: '8h'}
    )
    res.status(200).json({token, userId: user.id, message:"Вхід виконано"})
  } catch(err){
    res.status(500).json(err)
  }
})

module.exports = router

```

Вміст файлу character.js

```

const router = require('express').Router()
const User = require("../models/User")
const Character = require("../models/Character")
const Session = require("../models/Session")
const auth = require('../middleware/auth.middleware')

router.post("/new-character", auth, async (req, res) => {
  try{
    const newCharacter = new Character({
      userId: req.user.userId,
      name: req.body.name,
      race: req.body.race,
      health: req.body.health,
      mana: req.body.mana,
      armor: req.body.armor,
      class: req.body.class,
      img: 'char.jpg',
      dmg: req.body.dmg,
      speed: 1
    })
    const character = await newCharacter.save()
    res.status(200).json({message:"Персонажа створено"})
  }

```

		Борчаківський А.С			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Кониедайло В.В				51
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    } catch(err){
      res.status(500).json(err)
    }
  })

  router.get('/', auth, async (req, res) => {
    try{
      const characters = await Character.find({userId: req.user.userId})
      console.log(characters)
      res.status(200).json(characters)
    }catch(err){
      res.status(500).json(err)
    }
  })

  router.get('/no_session', auth, async (req, res) => {
    try{
      const characters = await Character.find({userId: req.user.userId,
sessionId: "None", isDead: false})
      res.status(200).json(characters)
    }catch(err){
      res.status(500).json(err)
    }
  })

  router.post('/', auth, async (req, res) => {
    try{
      console.log('sss')
      const character = await Character.findById(req.body.characterId)
      const user = await User.findById(req.user.userId)
      if(req.user.userId !== character.userId && !user.isAdmin){
        return res.status(400).json({message: "Ви не можете видалити персонажа, який вам не належить"})
      }

      const session = await Session.findOne({
        charactersId: {$in: [req.body.characterId]}
      })

      if(session){
        const session = await Session.findOneAndUpdate(
          {
            charactersId: {$in: [req.body.characterId]}
          },
          {
            $pull:{
              charactersId: req.body.characterId,
              playersId: character.userId
            }
          }
        )
      }
    }
  })

```

		Борчаківський А.С			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Кониедайло В.В				52
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        await session.save()
    }
    const char = await Character.findByIdAndDelete(req.body.characterId)
    res.status(200).json({message:"Персонажа видалено"})
} catch(err){
    res.status(500).json(err)
}
})

router.put("/:id/:sessionId", async (req, res) => {
    const session = await Session.findById(req.params.sessionId)
    const master = await User.findById(session.masterId)
    if(req.body.userId == master._id){
        try{
            const character = await Character.findByIdAndUpdate(req.params.id, {
                $set: req.body
            })
            res.status(200).json("Зміни успішно внесені")
        } catch(err){
            res.status(500).json(err)
        }
    } else{
        return res.status(404).send("Сторінка недоступна")
    }
})

router.get('/:id', auth, async (req, res) => {
    try{
        const char = await Character.findById(req.params.id)
        res.status(200).json(char)
    } catch(err){
        res.status(500).json(err)
    }
})

router.post("/get_many", auth, async (req, res) => {
    const { ids } = req.body
    try{
        const chars = await Character.find({_id: {$in: ids}})
        res.status(200).json(chars)
    } catch(err){
        res.status(500).json(err)
    }
})

router.post('/change', auth, async (req, res) => {
    try{
        const char = await Character.findById(req.body.characterId)
        char.dmg = req.body.dmg
        char.health = req.body.health
        char.armor = req.body.armor
    }
})

```

		Борчаківський А.С			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Концедайло В.В				53
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        await char.save()
        res.status(200).json({message: "Зміни внесені", char: char})
    }catch(err){
        res.status(500).json(err)
    }
})

module.exports = router

```

Вміст файлу Profile.jsx

```

import React, {useState, useEffect, useContext} from "react"
import { AuthContext } from "../context/AuthContext"
import { Header } from "../components/header"
import { CharacterCard } from "../components/characterCard"
import { Container, Row, Col, Button } from "react-bootstrap";
import { Popup } from "../components/popup";
import axios from "axios";

export const Profile = () => {

    const [characters, setCharacters] = useState([])
    const [notification, setNotification] = useState()

    const { token, logout } = useContext(AuthContext);

    const fetchCharacters = async () => {
        await axios.get("http://localhost:5000/api/character/", {
            headers: {
                Authorization: `Bearer ${token}`
            }
        }).then(res => {
            setCharacters(res.data)
        }).catch(err => {
        })
    }

    useEffect(() => {
        if(token){
            fetchCharacters()
        }
    }, [token])

    return(
        <>
            <Header/>
            <Container className="mt-5">
                <Row>
                    <Col xs={12} sm={6} md={4} lg={3} className="d-flex align-
items-stretch">

```

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Кониедайло В.В				54
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        <CharacterCard isadd={true}
fetchCharacters={fetchCharacters} setNotification={setNotification}/>
      </Col>
    {
      characters.map((character, i) => (
        <Col key={i} xs={12} sm={6} md={4} lg={3}
className="d-flex align-items-stretch">
          <CharacterCard character={character}
isadd={false} fetchCharacters={fetchCharacters}
setNotification={setNotification}/>
        </Col>
      ))
    }
  </Row>
  <Button variant="primary" className="mt-3" onClick={logout}>
    Вийти
  </Button>
</Container>
{notification && <Popup message={notification}
setMessage={setNotification}/>}
</>
)
}

```

Вміст файлу Main.jsx

```

import { Header } from "../components/header";
import { Container } from "react-bootstrap";
import { Maps } from "../components/maps";
import { Objects } from "../components/objects";
import { Sessions } from "../components/sessions";
import './Main.css'
import { useState } from "react";
import { Popup } from "../components/popup";

export const Main = () => {

  const [isMap, setIsMap] = useState(false)
  const [notification, setNotification] = useState()

  return (
    <div>
      <Header/>
      <Container className="mt-5">
        <h2 className="me-3">Мої цecії</h2>
        <Sessions setMessage={setNotification}/>
        <div className="d-flex justify-content-between">
          <h2 className="me-3 hover-effect" onClick={()=>
setIsMap(false)}>Вороги</h2>
        </div>
      </div>
    {

```

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Концедайло В.В				55
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        isMap ? (
            <Maps/>
        ) : (
            <Objects/>
        )
    }
    </Container>
    {notification && <Popup message={notification}
setMessage={setNotification}/>}
    </div>
    );
}

```

Вміст файлу newSessionModal.jsx

```

import React, { useState, useContext, useEffect } from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Button, Modal, Form, Row, Col } from 'react-bootstrap';
import { InviteField } from './inviteField';
import axios from 'axios';
import { AuthContext } from '../context/AuthContext';

export const NewSessionModal = () => {
    const [show, setShow] = useState(false);
    const [sessionId, setSessionId] = useState('')
    const [title, setTitle] = useState('')
    const [invites, setInvites] = useState([])
    const { token } = useContext(AuthContext);

    const handleClose = () => setShow(false);
    const handleShow = () => setShow(true);

    const handleCreate = async () => {
        await axios.post('http://localhost:5000/api/session/new-session',
{title:title}, {
    headers: {
        Authorization: `Bearer ${token}`
    }
}).then(res => {
    setSessionId(res.data._id)
}).catch(err => {
    console.log(err)
})
    }

    const fetchInvites = async() => {
        await axios.post('http://localhost:5000/api/session/session-invites',
{sessionId: sessionId}, {
    headers: {
        Authorization: `Bearer ${token}`
    }
}

```

		Борчаківський А.С.			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Кониедайло В.В.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }).then(res => {
      setInvites()
    })
  }

  useEffect(() => {
    console.log('xD')
  }, [])

  return (
    <div className='me-3'>
      <button className='btn btn-outline-light' onClick={handleShow}>
        Створити Сесію
      </button>

      <Modal show={show} onHide={handleClose}>
        <Modal.Header>
          {
            !sessionId ? (
              <Modal.Title>
                Створення нової сесії
              </Modal.Title>
            ) : (
              <Modal.Title>
                {title}
              </Modal.Title>
            )
          }

        </Modal.Header>
        {
          !sessionId && (
            <Modal.Body>
              <Form.Group>
                <Form.Label>Назва сесії:</Form.Label>
                <Row className='w-100 d-flex align-items-center'>
                  <Col xs="auto">
                    <Form.Control className='me-2' value={title}
onChange={(e)=>{setTitle(e.target.value)}}/>
                  </Col>
                  <Col xs="auto">
                    <Button variant="dark" size="sm"
onClick={handleCreate}>Створити</Button>
                  </Col>
                </Row>

              </Form.Group>
            </Modal.Body>
          )
        }
        <Modal.Footer>

```

		Борчаківський А.С.			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Кониедайло В.В.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        {
            sessionId && (
                <Row className="w-100">
                    <InviteField sessionId={sessionId}/>
                    <InviteField sessionId={sessionId}/>
                    <InviteField sessionId={sessionId}/>
                    <InviteField sessionId={sessionId}/>
                </Row>
            )
        }

    </Modal.Footer>
</Modal>
</div>
);
}

```

Вміст файлу invitesModal.jsx

```

import React, {useState, useEffect, useContext} from "react"
import { AuthContext } from "../context/AuthContext";
import 'bootstrap/dist/css/bootstrap.min.css';
import { Button, Modal, Pagination } from 'react-bootstrap';
import { Invite } from "../invite";
import { Popup } from "../popup";
import { Link } from "react-router-dom";
import axios from "axios";
import { Character } from "../character";
const PAGE_SIZE = 4;

export const InvitesModal = () => {

    const { token } = useContext(AuthContext);
    const [invites, setInvites] = useState([])
    const [inviteId, setInviteId] = useState()
    const [show, setShow] = useState(false);
    const [notification, setNotification] = useState()
    const [isAccept, setIsAccept] = useState(false)
    const [isLoading, setIsLoading] = useState(false)
    const [characters, setCharacters] = useState([])
    const [currentPage, setCurrentPage] = useState(1);

    const startIndex = (currentPage - 1) * PAGE_SIZE;
    const endIndex = Math.min(startIndex + PAGE_SIZE, characters.length);
    const currentCharacters = characters.slice(startIndex, endIndex);

    const handlePageChange = (page) => {
        setCurrentPage(page);
    };

    const handleClose = () => setShow(false);

```

		Борчаківський А.С.			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Кониедайло В.В.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const handleShow = () => setShow(true);

async function fetchInvites() {

    await axios.get("http://localhost:5000/api/invite/my-invites", {
        headers: {
            Authorization: `Bearer ${token}`
        }
    }).then(res => {
        setInvites(res.data)
    }).catch(err => {
        console.log(err)
    })
}

const fetchCharacters = async () => {
    setIsLoading(true)
    await axios.get("http://localhost:5000/api/character/no_session", {
        headers: {
            Authorization: `Bearer ${token}`
        }
    }).then(res => {
        setCharacters(res.data)
    }).catch(err => {
    })
    setIsLoading(false)
}

useEffect(() => {
    if(isAccept){
        fetchCharacters()
        console.log('fetch')
    }
}, [isAccept])

useEffect(() => {

    if(token){
        fetchInvites()
    }

}, [token])

useEffect(() => {
    const intervalId = setInterval(() => {
        fetchInvites()
        console.log("here")
    }, 5000)

    return () => clearInterval(intervalId)
}, [])

```

		Борчаківський А.С			ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
		Кониедайло В.В				59
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return(
  <>
    <div className="position-relative me-3">
      
      {invites.length > 0 && (
        <span className="position-absolute top-0 start-100 translate-
middle badge rounded-pill bg-info">
          {invites.length}
        </span>
      )}
    </div>

    <Modal show={show} onHide={handleClose}>
      {
        !isAccept ? (
          <>
            <Modal.Header>
              <Modal.Title>
                Мої запрошення
              </Modal.Title>
            </Modal.Header>
            {
              !invites.length > 0 ? (
                <Modal.Body>
                  Активні запрошення відсутні
                </Modal.Body>
              ) : (
                <Modal.Body>
                  {
                    invites.map((invite) => (
                      <Invite invite={invite}
setNotification={setNotification} fetchInvites={fetchInvites}
setIsAccept={setIsAccept} setInviteId={setInviteId}/>
                    ))
                  }
                </Modal.Body>
              )
            }
          </>
        ) : (
          <>
            <Modal.Header>
              <Modal.Title className="d-flex justify-content-
between">
                Оберіть персонажа для гри

```

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ІЗ	Арк.
		Кониедайло В.В				60
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        </Modal.Title>
      </Modal.Header>
      {
        !characters.length > 0 ? (
          <Modal.Body>
            У вас відсутні персонажі. Створіть
персонаж в <Link to="/profile">профілі</Link>
          </Modal.Body>
        ) : (
          <Modal.Body>
            {
              isLoading ? (
                <span>
                  Завантаження...
                </span>
              ) : (
                <>
                  <div className="d-flex">
                    {currentCharacters.map((character
, index) => (
                      <Character
character={character} inviteId={inviteId} setInviteId={setInviteId}
setIsAccept={setIsAccept} setNotification={setNotification}
fetchInvites={fetchInvites}/>
                    )))
                  </div>
                  <div className="d-flex justify-
content-center align-items-center mt-4">
                    {characters.length > PAGE_SIZE &&
(
                      <Pagination>
                        {Array.from(
                          { length:
Math.ceil(characters.length / PAGE_SIZE) },
                          (_, index) => (
                            <Pagination.Item
                              key={index + 1}
                              active={index + 1 ===
currentPage}
                              onClick={() =>
handlePageChange(index + 1)}
                            >
                              {index + 1}
                            </Pagination.Item>
                          )
                        )}
                      </Pagination>
                    )}
                  </div>
                </>
              )
            }
          </Modal.Body>
        )
      }
    </>
  )
}

```

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				61
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        )
    }
    </Modal.Body>
)
}

<Modal.Footer>
    <Button variant="danger" className="btn-sm"
onClick={() => setIsAccept(false)}>
        До запрошень
    </Button>
</Modal.Footer>
</>
)
}

</Modal>
{notification && <Popup message={notification}
setMessage={setNotification}/>}
</>
)
}

```

		Борчаківський А.С			ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
		Кониедайло В.В				62
Змн.	Арк.	№ докум.	Підпис	Дата		