НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення для створення інтерактивних ігрових елементів навчання в українській школі

Виконав студент IV курсу, групи ІТ-91
(шифр групи)

Гайова Альона Денисівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент каф. ІПІ ФІОТ, к.т.н., доцент Іванова Л.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації ст. викл. Головченко Максим Михайлович _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент зав. кафедрою Комп'ютерної інженерії ДУТ, к.ф.-м.н., доцент Світлана Поперешняк _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютерних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Гайової Альони Денисівни
(прізвище, ім'я, по батькові)

1. Тема проєкту «Програмне забезпечення для створення інтерактивних ігрових елементів навчання в українській школі»

керівник проєкту Іванова Любов Михайлівна, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31»квітня 2023 р. №2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення.
3) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту, розгортання програмного продукту.

5. Перелік графічного матеріалу
1) Схема структурна варіантів використань
2) Схема бази даних

3) Схема структурна компонентів програмного забезпечення _____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	18.03.2023	
3	Постановка та формалізація задачі	19.03.2023	
4	Розробка інформаційного забезпечення	27.03.2023	
5	Алгоритмізація задачі	3.04.2023	
6	Обґрунтування вибору використаних технічних засобів	6.04.2023	
7	Розробка програмного забезпечення	4.05.2023	
8	Налагодження програми	10.05.2023	
9	Виконання графічних документів	15.05.2023	
10	Оформлення пояснювальної записки	29.05.2023	
11	Подання ДП на попередній захист	02.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Альона ГАЙОВА
(підпис) (ініціали, прізвище)

Керівник _____ Любов ІВАНОВА
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 56 таблиць, 32 рисунків та 19 джерел – загалом 84 сторінки.

Дипломний проєкт присвячений розробці програмного забезпечення для інтерактивних ігрових елементів навчання у школах.

Метою проєкту є підвищення зацікавленості дітей у навчанні шляхом розробки веб-застосунку, який наддасть можливість вчителям доводити матеріал, а учням його засвоювати у ігровій формі.

Об'єкт дослідження: застосунки і методи, які використовуються для навчання в ігровій формі.

Предмет дослідження: програмне забезпечення для створення інтерактивного ігрового навчання.

У першому розділі було описано та проаналізовано предметну область. Було знайдено існуючі та популярні на ринку рішення, недоліки та переваги яких для використання в українській освіті було ретельно досліджено. На основі цієї інформації визначено цілі програмного продукту в рамках дипломного проєкту, описано функціональні та нефункціональні вимоги до програмного забезпечення.

У другому розділі було виконано опис бізнес процесів за допомогою діаграм, проаналізовано різні архітектури програмного забезпечння та доступні інструменти, після чого було обрано і описано архітектуру веб-застосунку із RESTful клієнт серверною архітектурою.

Третій розділ дипломного проєкту охоплює контроль веб застосунку. Для цього було описано процеси тестування, тест-кейси та наведено контрольний приклад для тестування.

У четвертому розділі було виконано опис процесу впровадження та супроводу застосунку, зокрема розгортання програмного забезпечення та описано інструкцію користувача.

КЛЮЧОВІ СЛОВА: ВЕБ ЗАСТОСУНОК, ІНТЕРАКТИВНЕ НАВЧАННЯ, ГРА, ОСВІТА.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 56 tables, 32 figures and 19 sources – in total 84 pages.

This work is dedicated to the development of software for interactive game elements of stydying in schools.

The goal of the project is to increase interest in learning among children through the development of a web application that will enable teachers to demonstrate the material, and students to learn it in a playful way.

Object of research: applications and methods used for learning in a game form.

Research subject: software for creating interactive game lessons.

In the first chapter, the subject area was described and analyzed. Existing and popular solutions on the market were found, the disadvantages and advantages of which were carefully studied for use in Ukrainian education. On the basis of this information, the goals of the software product as part of the diploma project were defined, and the functional and non-functional requirements for the software were described.

In the second chapter, the description of business processes was carried out with the help of diagrams, various software architectures and available tools were analyzed, after which the architecture of a web application with a RESTful client-server architecture was selected and described.

The third section of the diploma project covers web application control. For this, testing processes, test cases and a control example for testing were described.

In the fourth chapter, a description of the process of implementation and maintenance of the application was performed, in particular, the deployment of the software and the user manual was described.

KEYWORDS: WEB APP, INTERACTIVE LAERNING, GAME, EDUCATION.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9106.045440.01.91	Технічне завдання	23	
3	А4	КПІ.ІТ-9106.045440.02.81	Пояснювальна записка	84	
4	А4	КПІ.ІТ-9106.045440.03.12	Текст програми	40	
5	А4	КПІ.ІТ-9106.045440.04.51	Програма та методика тестування	5	
6	А4	КПІ.ІТ-9106.045440.05.34	Керівництво користувача	20	
7	А3	КПІ.ІТ-9106.045440.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІТ-9106.045440.07.99	Схема бази даних	1	
9	А3	КПІ.ІТ-9106.045440.08.99	Схема структурна компонентів	1	
			програмного забезпечення		

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9106.045440.00.90		
Розроб.		Гайова А.Д.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Іванова Л.М.					
Н.контр.		Головченко М.М				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-91	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНИХ ІГРОВИХ ЕЛЕМЕНТІВ НАВЧАННЯ В УКРАЇНСЬКІЙ ШКОЛІ

Технічне завдання

КПІ.ІП-9106.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов ІВАНОВА

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Альона ГАЙОВА

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для створення інтерактивних ігрових елементів навчання в українській школі.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення «Програмне забезпечення для створення інтерактивних ігрових елементів навчання в українській школі» КПІ.ІТ-9106.045440.03.91, котра використовується для створення завдань для учнів у інтерактивній ігровій формі та призначена для зацікавлення дітей у навчання, мотивації учнів на уроках та урізноманітнення навчального процесу.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного забезпечення для створення інтерактивних ігрових елементів навчання в українській школі є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для впровадження завдань в ігровій формі у навчальних закладах.

Метою розробки є підвищення зацікавленості у навчанні серед дітей через створення завдань у ігровій формі.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– До кожної сторінки додати меню вверху екрану з наступними елементами: логотип і назва «Challenger», які є посиланням на домашню сторінку; кнопка «?», яка веде на сторінку «Зв’яжіться з нами»; для неавторизованих користувачів: кнопка «Вхід», яка веде на сторінку логіну; для авторизованих користувачів: кнопка «Мої ігри», яка веде на сторінку з переліком ігор користувача, «Профіль» - на сторінку особистого профіля, «Вихід» - вихід із акаунта користувача.

Це меню можна побачити на всіх екранних формах, представлених нижче на рисунках у розділі 4.1.1.

– Додати домашню сторінку, яка відображатиме інформацію про застосунок. Прототип екранної форми зображено на рисунку 4.1.

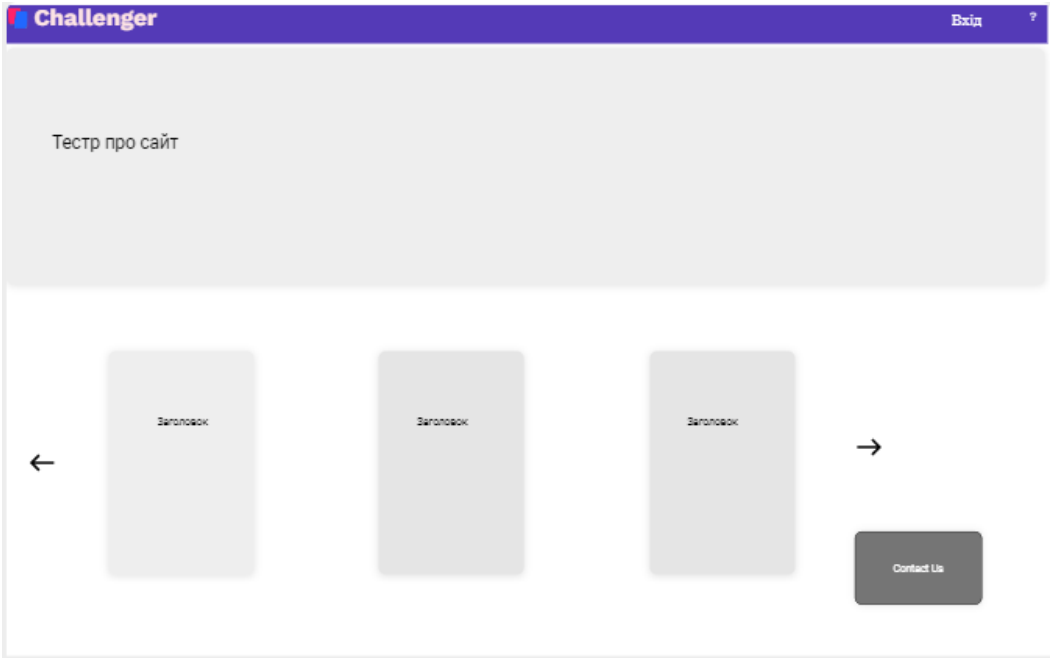


Рисунок 4.1 – Прототип екрану «Домашня сторінка»

– Додати сторінку логіну, де користувач може увійти в існуючий акаунт за допомогою адреси електронної пошти та паролю. Прототип екранної форми зображено на рисунку 4.2.

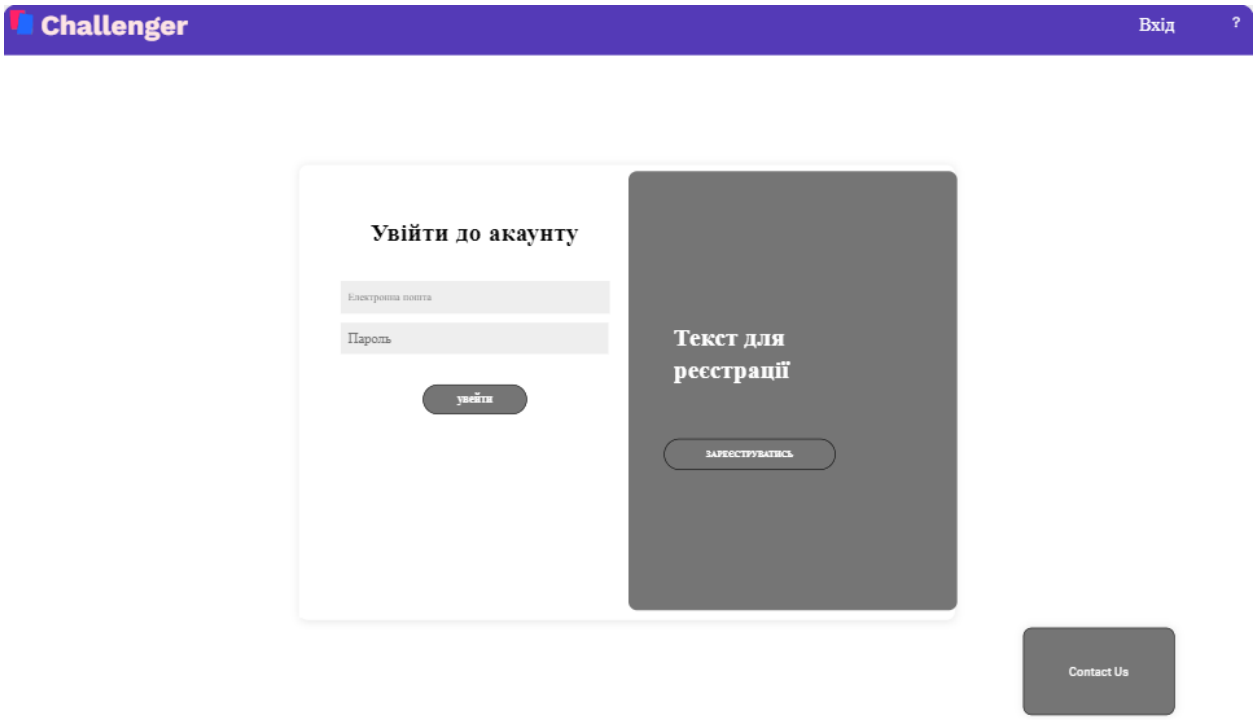


Рисунок 4.2 – Прототип екрану «Логін»

Заповнення полів логіну коректними і існуючими у системі даними і натискання на кнопку «Увійти» авторизує користувача і переведе його на сторінку «Мої ігри». У разі введення некоректних даних цього не відбудеться. Кнопка з надписом «Contact Us» має бути тематично стилізована під конверт і переводити на сторінку «Зв’яжіться з нами». Кнопка зареєструватись переводить екран у інший стан і сторінка стає екраном реєстрації. Цей екран зображено на рисунку 4.3.

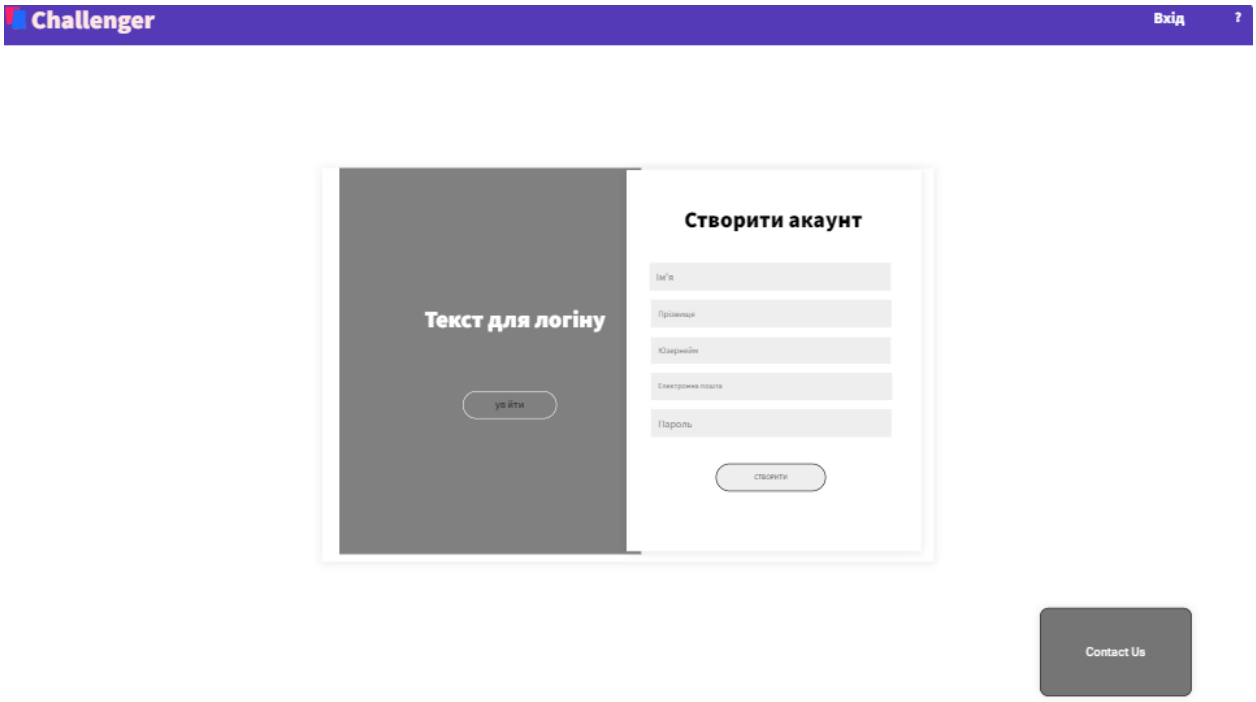


Рисунок 4.3 – Прототип екрану «Реєстрація»

На цьому екрані з метою створення акаунту користувач має ввести коректні значення для полів ім’я (мінімум 3 символи), прізвище (мішімум три символи), юзернейм (мінімум 3 символи), електронна пошта (валідна адреса електронної пошти за шаблоном ^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$), пароль (мінімум 1 велика літера, 1 символ, 1 цифра, не менше 8 символів). Якщо всі дані валідні і ця адреса пошти ще не зайнята, натискання на кнопку «» авторизує користувача і переведе його на сторінку «Мої ігри». У разі введення некоректних даних цього не відбудеться. Кнопка «Увійти» повертає екран у стан логіну.

– Додати сторінку «Мої ігри» з переліком всіх ігор створених користувачем. Ця сторінка має надавати можливість керування іграми (створення, видалення, зміна режиму, стирання прогресу) та можливість запуску гри. Потрапити на цю сторінку можна за допомогою кнопки «Мої ігри» на меню авторизованого користувача чи одразу після авторизації. Прототип екранної форми зображено на рисунку 4.4.

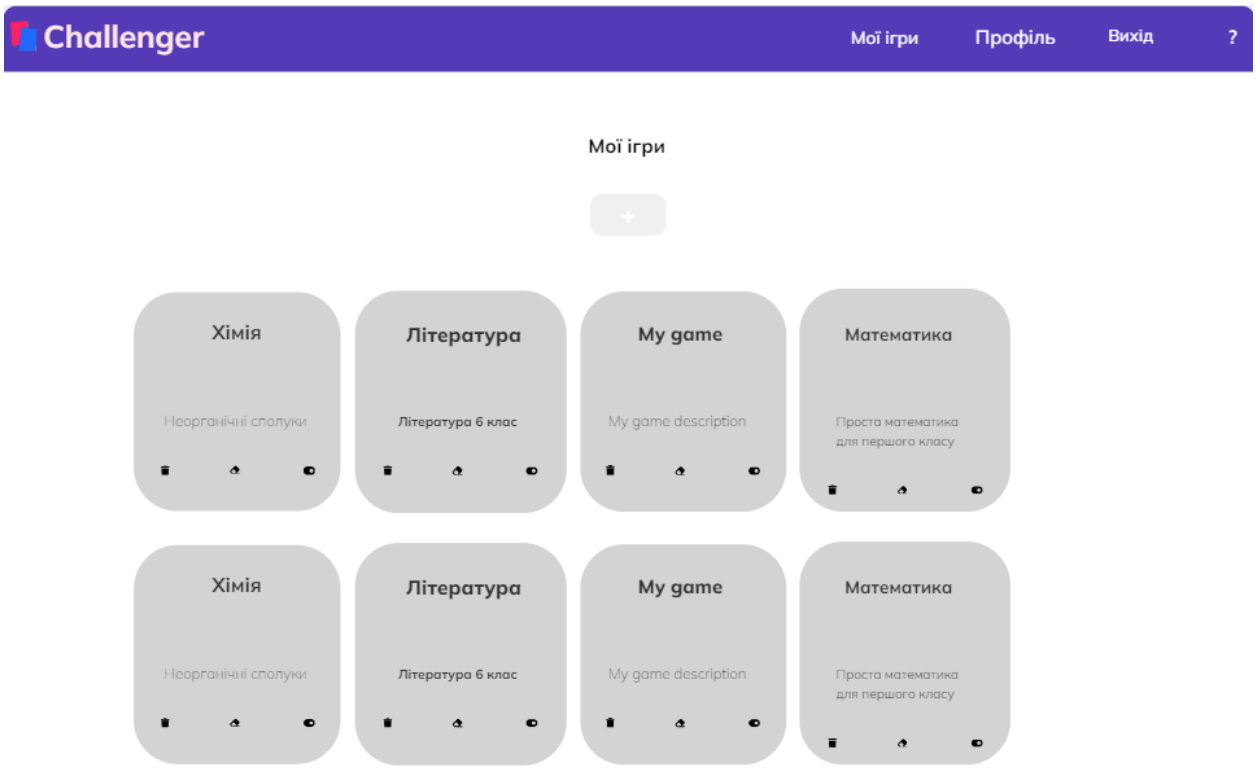


Рисунок 4.4 – Прототип екрану «Мої ігри»

Кожна гра має мати 3 кнопки – кнопку видалення гри у вигляді сміттєвого баку, кнопку стирання прогресу у вигляді ластику і кнопку перемикання режиму гри у вигляді перемикача. Всі три дії, які змінюють стан гри відкривають вікно із підтвердженням дії. Форма для підтвердження видалення зображена на рисунку 4.5. Форма для підтвердження видалення прогресу – на рисунку 4.6. Форма для підтвердження перемикання режиму – на рисунку 4.7.

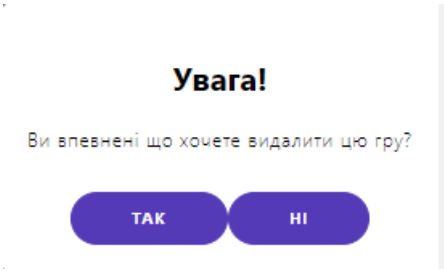


Рисунок 4.5 – Вікно для підтвердження видалення гри

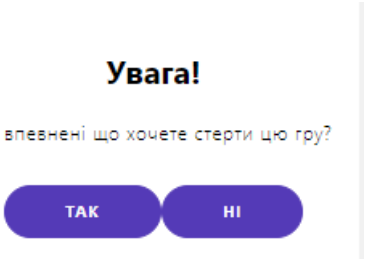


Рисунок 4.6 – Вікно для підтвердження видалення прогресу гри

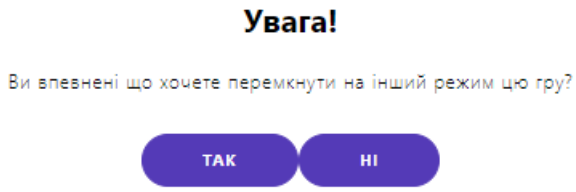


Рисунок 4.7 – Вікно для підтвердження зміни режиму гри

– Додати сторінку для додавання нової гри, яка складатиметься з декількох компонентів, які будуть перераховані нижче у пунктах 1-5. Для переходу на цю сторінку необхідно натиснути кнопку «+» на сторінці «Мої ігри».

Додати компонент для додавання базової інформації про гру: назва (3-20 символів), опис (5-1000 символів), мета гри (число більше 0), чи є гра командною (у вигляді чекбоксу). Якщо всі дані є правильними, то натискання на кнопку «Продовжити» переведе користувача на наступний екран. Прототип першого екрану зображено на рисунку 4.8.

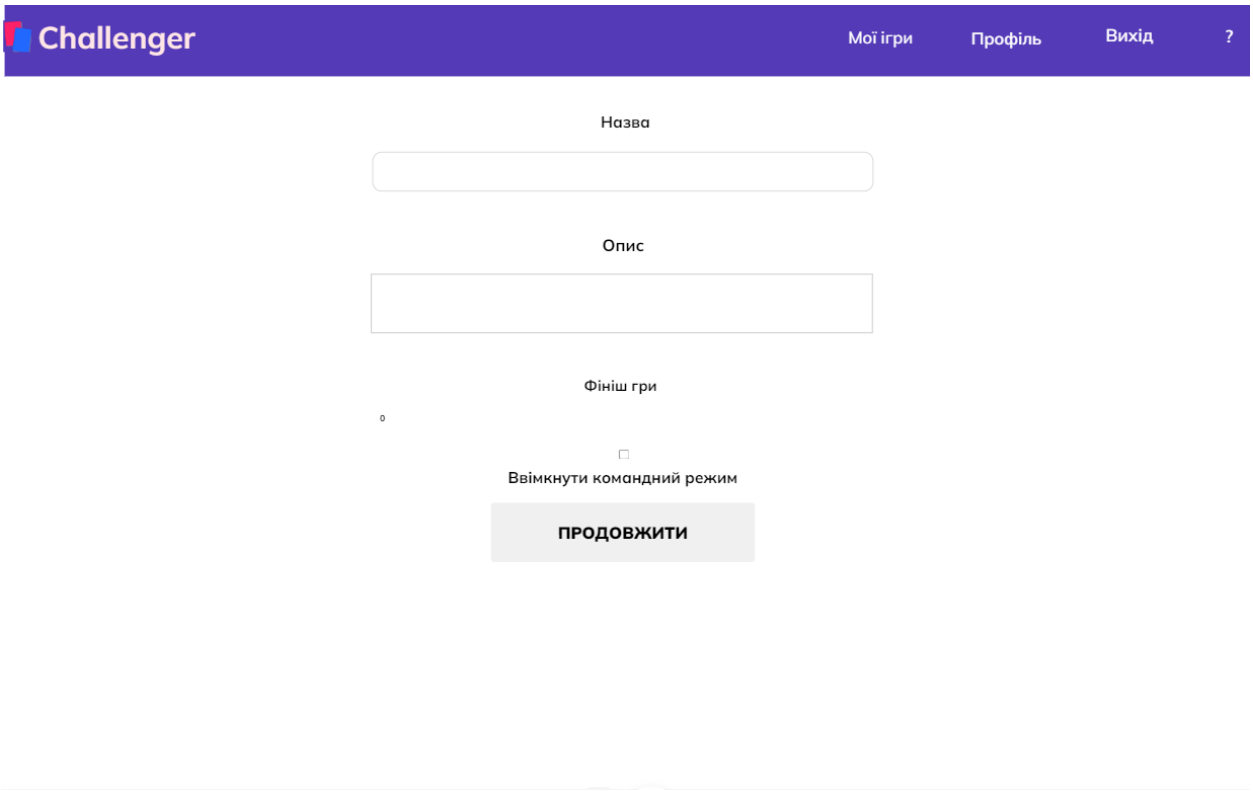


Рисунок 4.8 – Сторінка першого етапу створення гри

Додати компонент для створення категорій гри. Кожна категорія має складатись з назви (3-50 символів) та кольору. Форма має надавати можливість додати декілька категорій (кнопка +), видалення категорії (кнопка видалення) та переглянути список створених категорій. Прототип екрану зображено на рисунку 4.9. Якщо всі дані є правильними, то натискання на кнопку «Продовжити» переведе користувача на наступний екран.

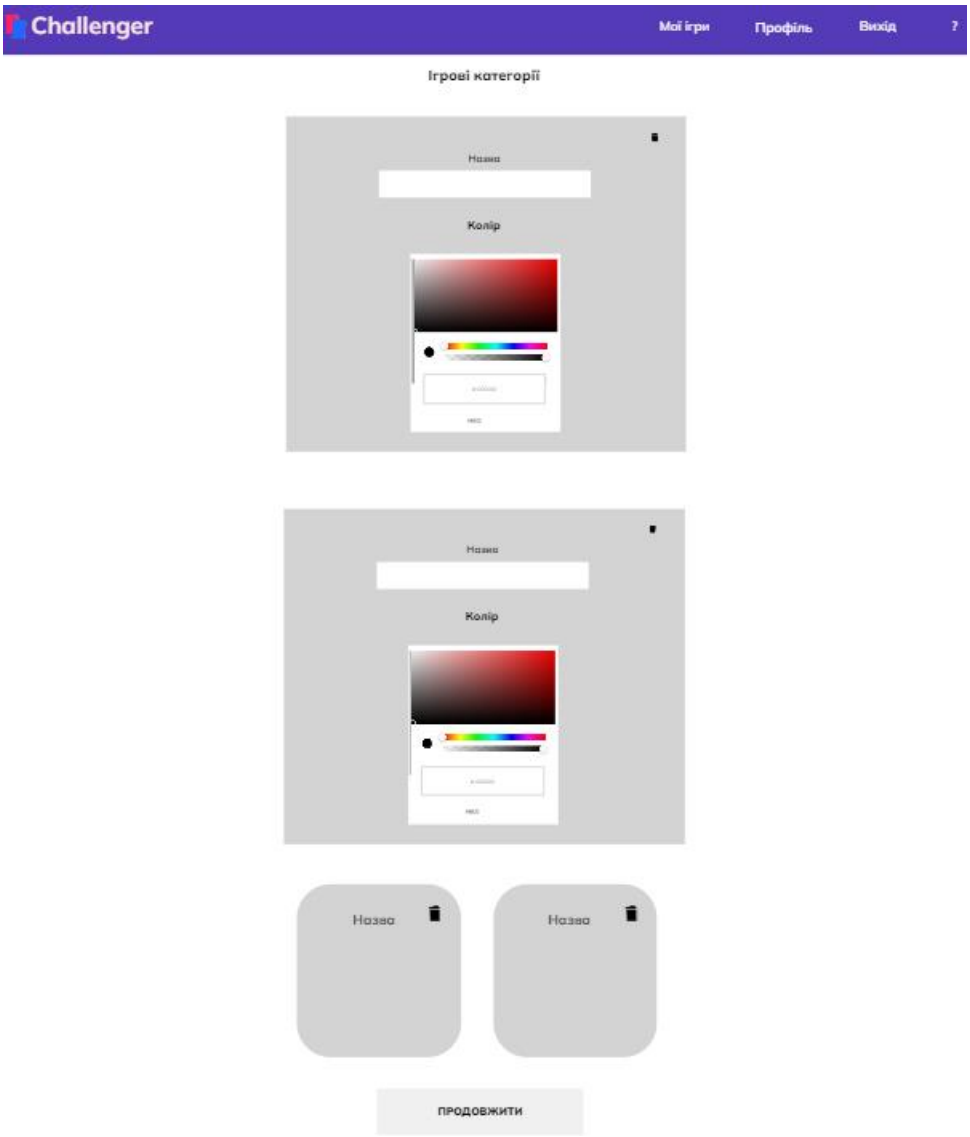


Рисунок 4.9 – Сторінка другого етапу створення гри

Додати компонент для створення ролей гри. Кожна роль має складатись з назви (3-20 символів). Форма має надавати можливість додати декілька ролей (кнопка +), видалення ролі (кнопка видалення) та переглянути список створених ролей. Прототип екрану зображено на рисунку 4.10. Якщо всі дані є правильними, то натискання на кнопку «Продовжити» переведе користувача на наступний екран.

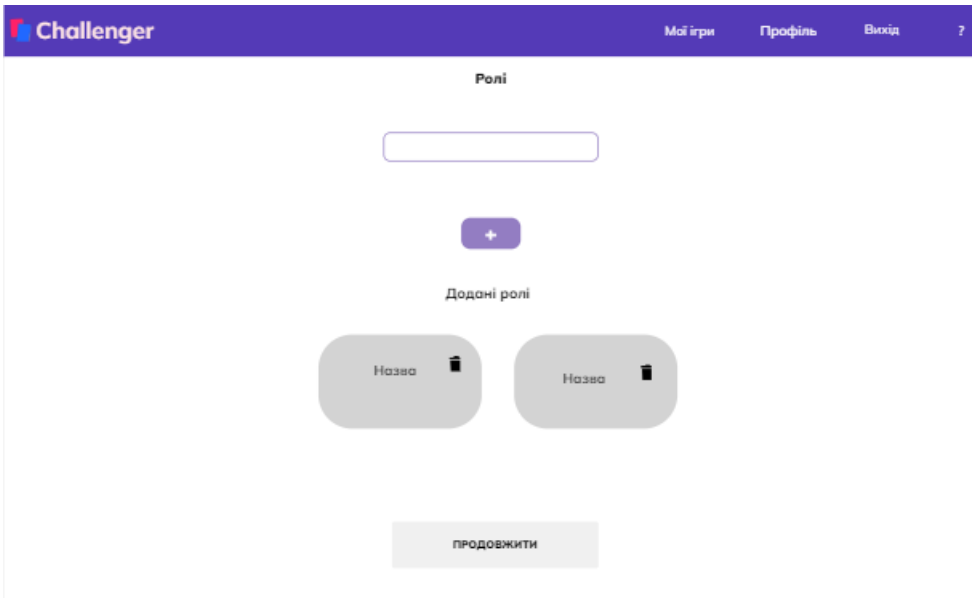


Рисунок 4.10 – Сторінка третього етапу створення гри

Додати компонент для створення карток гри. Кожна картка має складатись з опису (5-200 символів), обраної категорії (із списку категорій гри), кількості балів(число). Форма має надавати можливість додати декілька карток (кнопка +), видалення картки (кнопка видалення) та переглянути список створених карток. Прототип екрану зображено на рисунку 4.11. Якщо всі дані є правильними, то натискання на кнопку «Продовжити» переведе користувача на наступний екран.

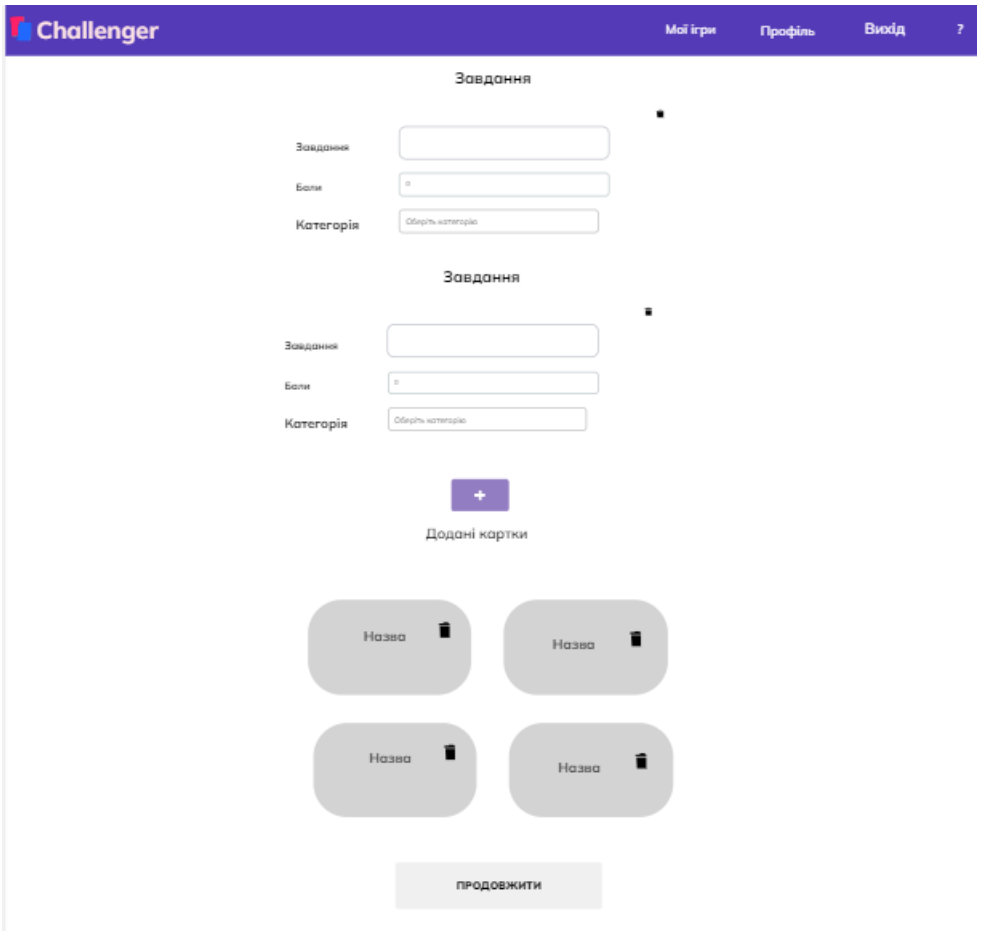


Рисунок 4.11 – Сторінка четвертого етапу створення гри

Додати компонент, який підтверджує успішне додання гри. Прототип цього екрану зображено на рисунку 4.12.

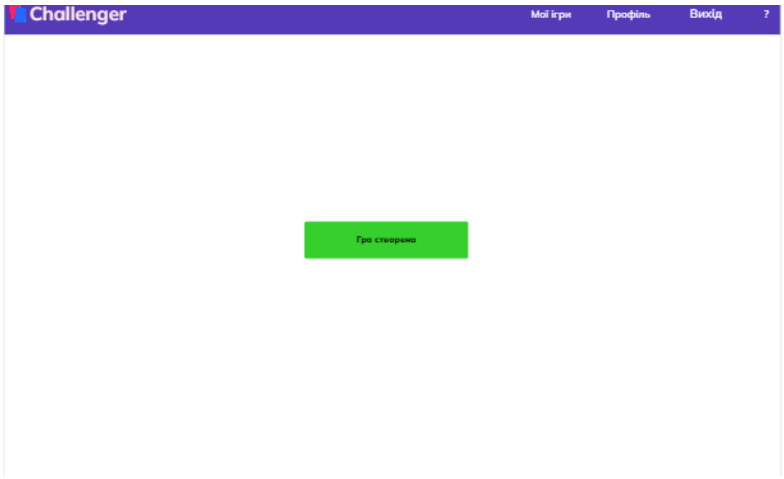


Рисунок 4.12 – Сторінка п’ятого етапу створення гри

– Додати сторінку гри, яка буде головним екраном гри. Сторінка має зображати ігрове поле у вигляді космосу, де кодгий гравець

представлений у вигляді ракети чи команда гравців як одна ракета, які просуваються космосом у процесі проходження гри до мети. Також екран має відображати картку із завданням та таблицю, яка більше детально показує прогрес учасників. Прототип даної сторінки зображено на рисунку 4.13. Перехід на дану сторінку відбувається після натискання на відповідну гру на сторінці «Мої ігри».

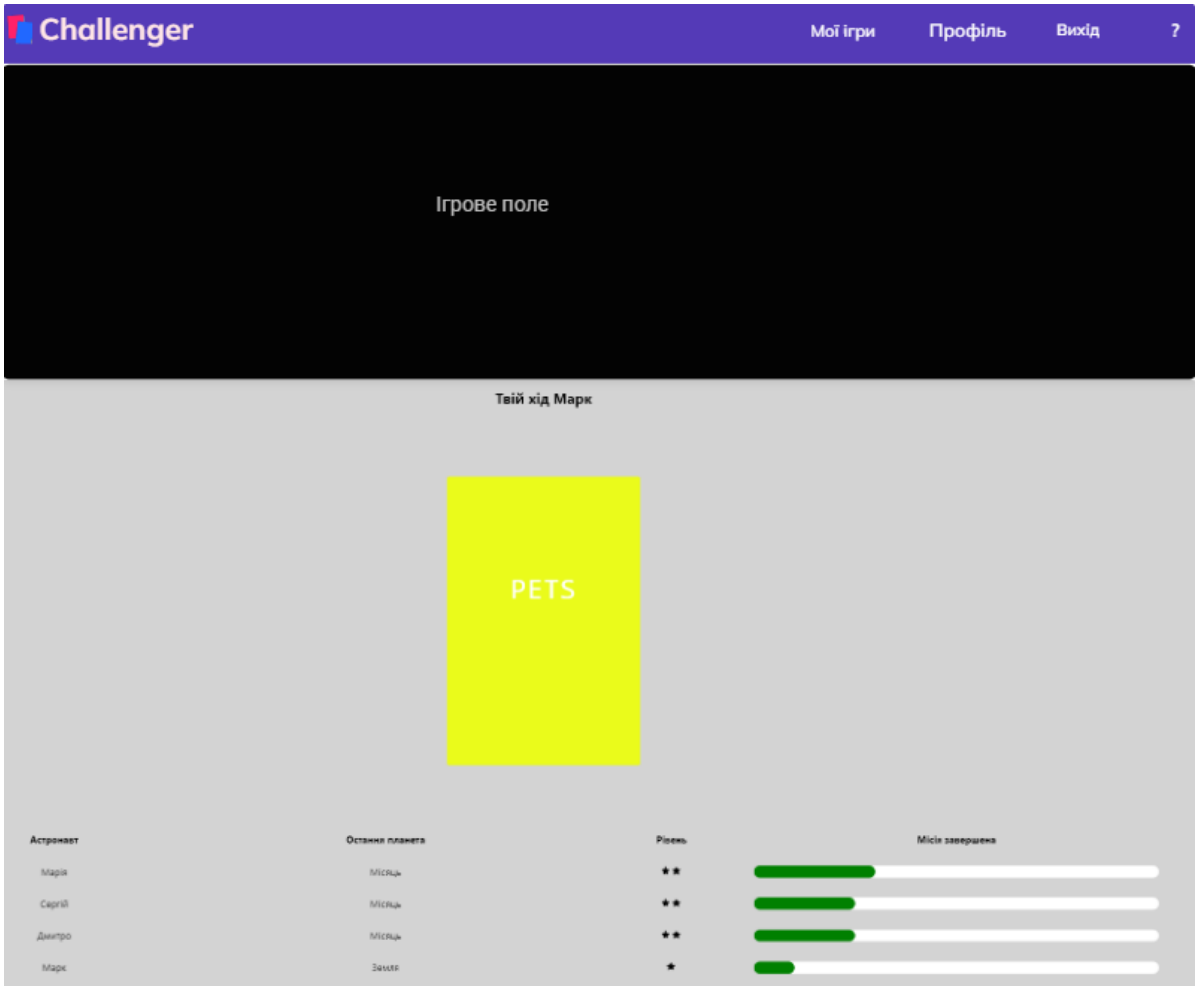


Рисунок 4.13 – Сторінка гри

Після завершення гри у користувача має бути можливість побачити результати. Там має бути відображено хто виконував завдання, його статус, кількість балів за це завдання.

– Додати сторінку профіля користувача, де він може побачити свої дані. Прототип зображено на рисунку 4.14.

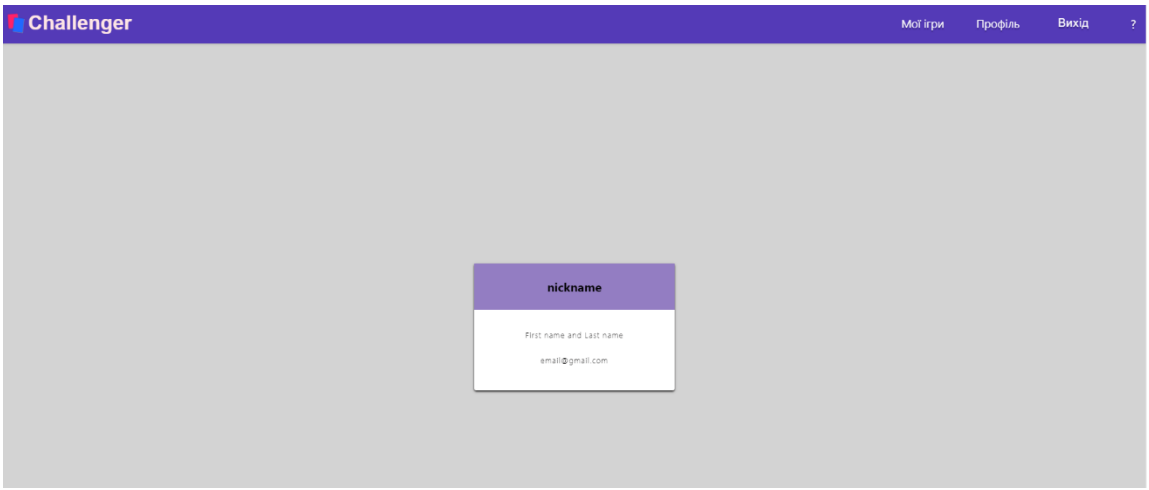


Рисунок 4.14 – Прототип сторінки «Профіль»

– Додати сторінку для зворотнього зв’язку, користувача, розмістити там адресу електронної пошти та відповіді на розповсюджені запитання. Прототип зображено на рисунку 4.15.

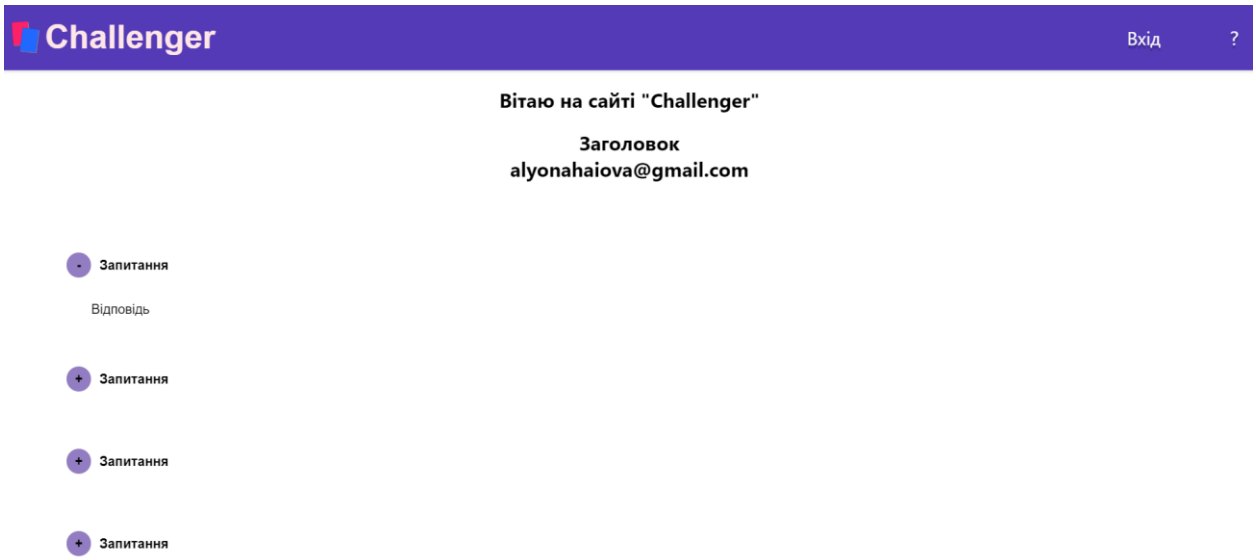


Рисунок 4.15 – Прототип сторінки «Зв’яжіться з нами»

4.1.2 Для користувача:

4.1.2.1 Авторизація користувача

– забезпечити можливість входу до акаунту;

– забезпечити можливість створення нового акаунту;

4.1.2.2 Створення гри

− забезпечити додавання до сутності гри назву, опис, мету гри, режим гри (командний чи змагальний), список ігрових категорій (назва, колір), список штрафних категорій (назва, колір), список ролей гри (ім’я), список карток гри із завданнями (категорія, завдання, бали);

4.1.2.3 Управління сутністю гри

− забезпечити можливість зміни режиму гри;

− забезпечити можливість видалення прогресу гри;

− забезпечити процес видалення гри;

4.1.2.4 Механіка гри

− забезпечення черговості ролей у грі;

− забезпечити можливість отримувати наступну картку;

− забезпечити можливість оновлення картки для поточної ролі;

− забезпечити можливість отримання штрафної картки за невиконане завдання;

− забезпечити можливість додавання балів за виконане завдання;

− забезпечити можливість віднімання балів за невиконане завдання;

− забезпечити можливість досягнення мети гри;

− забезпечити відображення прогресу учасників під час гри;

4.1.2.5 Забезпечення механізму завершення гри

− забезпечити можливість завантаження результатів сесії гри у вигляді текстового файлу;

− забезпечити можливість виводу результатів сесії гри на екран;

− забезпечити можливість продовження гри після досягнення мети;

− забезпечити можливість завершення гри після досягнення мети.

4.1.3 Додаткові вимоги:

− інтуїтивний візуальний інтерфейс;

− простота використання;

− швидка робота;

- безпека даних користувачів;
- гнучкість у використанні;
- розширюваність;
- інтерфейс українською мовою.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних. Зберігати копію бази даних один раз на тиждень на випадок її пошкодження.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.3 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на персональних комп’ютерах з операційною системою Windows, Unix подібною системою або MacOS із доступом до інтернету.

Мінімальна конфігурація технічних засобів:

- тип процесору: Intel Core i5;
- об‘єм ОЗП: 4 Гб;
- підключення до мережі Інтернет зі швидкістю від 20 мегабіт.
- Рекомендована конфігурація технічних засобів :
- тип процесору: Intel Core i7;
- об‘єм ОЗП: 16 Гб;

− підключення до мережі Інтернет зі швидкістю від 80 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows'XP, Windows NT і т.д.) або Unix.

4.5.2 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.3 Вимоги до вихідних даних

Результати завантаження результатів гри повинні бути представлені в наступному форматі: текстовий файл з розширенням .txt і назвою за шаблоном game_yyyy-mm-dd_id.txt.

4.5.4 Вимоги до мови розробки

Розробку виконати на мові програмування Java та JavaScript.

4.5.5 Вимоги до середовища розробки

Розробку виконати на платформі, яка відповідає таким вимогам:

− наявність Java Development Kit (JDK): Для розробки програмного забезпечення на Java рекомендується встановити актуальну версію JDK. Можна використовувати Oracle JDK або OpenJDK згідно з ваших вимог та можливостей.

− Редактор коду: Рекомендується використовувати інтегроване середовище розробки IntelliJ IDEA.

− Засоби для версіонування коду: Для ефективного управління версіями вашого коду рекомендується використовувати систему контролю версій Git.

– Управління залежностями: Для ефективного управління залежностями та підтримки сторонніх бібліотек рекомендується використовувати систему керування залежностями Gradle.

4.5.6 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді репозиторію на GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Розгорнути веб-застосунок на сервері.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- текст програми;
- програма та методика тестування;
- керівництво користувача;

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна варіантів використання;
- схема структурна компонентів програмного забезпечення;
- схема бази даних;

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	10.03	
2.	Розробка технічного завдання	19.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	20.03	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	03.04	Схема структурна програмного забезпечення та специфікація компонентів (схема структурна компонентів, схеми основних бізнес процесів)
5.	Програмна реалізація програмного забезпечення	04.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів графічної частини проекту	15.05	Графічний матеріал проекту
8.	Розробка матеріалів текстової частини проекту	29.05	Пояснювальна записка
9.	Оформлення технічної документації проекту	04.06	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Програмне забезпечення для створення інтерактивних ігрових елементів навчання в українській школі

КПІ.ІТ-9106.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
UI	–	User Interface, користувацький інтерфейс
IT	–	Інформаційні технології
ER	–	Entity-Relation diagram
ОС	–	Операційна система.
БД	–	База даних.

ВСТУП

Якщо ми поглянемо проблеми сучасної системи шкільної освіти, то побачимо серед них нудьгу і небажання вчитись. Це цілком зрозуміло, адже сучасні діти знаходяться під впливом великої кількості розважального контенту, доступного для них цілодобово. До цієї проблеми накладається друга - концентрація при самонавчанні, виконанні домашнього завдання чи підготовки до тесту.[1]

У наші дні з цією проблемою не справляється багато дорослих, що вже казати про дітей та підлітків. Наша держава виказує зацікавленість даною проблемою. З 2018/2019 навчального першокласники по всій Україні почали навчатись за стандартом нової української школи. Але, як завжди буває з новими концепціями, на її адаптацію потрібно багато часу і велика кількість вчителів немає інструментів для ефективної роботи зі своїми учнями, яка б відповідала новим стандартам.[2]

Це все робить дану проблему дуже актуальною. Після введення державою концепції НУШ система освіти все ще потребує нових рішень та інструментів для її втілення.

Існує дуже багато нових застосунків для вивчення конкретних предметів і вони показують дуже хороші результати і викликають неабияку зацікавленість у своїх учнів.[3] Мій застосунок спрямований на створення саме інструменту, який стане в нагоді в навчальному процесі. Таким чином, його можна буде використовувати у поєднанні з будь-якою навчальною програмою, незалежно від віку та рівня учнів, предмету, навчальної програми. Це може призвести до суттєвого покращення концентрації та заглиблення у навчальний процес школярів та студентів, ставши корисним доповненням до освітнього процесу.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Одна з основних проблем освіти – зацікавлення учня, заохочення до навчання. Однак найбільш гостро ця проблема проявляється у дитячих школах. Іноді буває важко пояснити дитині всю важливість навчання та його користь. Люди зазвичай не полюблять робити щось нецікаве якщо у них немає сильної мотивації.[4] У цьому проекті я хочу поглянути на проблему під іншим кутом. Якщо ми не можемо вмотивувати деяких учнів на нецікаве для них навчання, то чому б не зробити його більш привабливим в дитячих очах?

У всі часи хороші педагоги працювали над цією проблемою. Вони вигадували нові і нові способи привернути дітей до знань. Одним з найкращих прикладів я вважаю британського картографа Джона Спілсбері, який вигадав перший пазл. Він зробив його з мапи, мотивуючи дітей вивчати географію. Іноді для того щоб зацікавити учня, достатньо простого і легкого в опануванні методу[5].

Проте сучасний світ сповнений розваг і дитячого контенту, які доступні кожній дитині з телефоном в руках. В таким умовах важко втримати увагу в школі, адже вже у цьому віці вони звикли до різноманіття цікавих та яскравих ігор, зроблених справжніми професіоналами своєї справи. Навіть дуже хорошому педагогу може бути складно конкурувати з індустрією розваг без спеціальних інструментів.

Існують різні рішення, які вирішують цю проблему. Загалом, ринок переповнений рішеннями, які пропонують онлайн уроки. Навчальні відео, інтерактивні посібники, тощо. Вони частково вирішують проблему непривабливості навчального процесу. Вони є хорошим джерелом інформації, але зазвичай мають суттєвий недолік – власний навчальний план. У школах є дуже важливим дотримання навчального плану, затвердженого Міністерством Освіти України.[2]

Також наявно багато рішень, які пропонують створення власних завдань

(Quizlet, Google Classroom). Це надає певну гнучкість у їх використанні. На жаль, ці рішення не мають інтерфейсу гри і не вирішують питання зацікавлення та заохочення учнів.

1.2 Змістовний опис і аналіз предметної області

Зараз все більшої популярності набувають методи навчання, які побудовані на ігрових та інтерактивних механіках. Такі програмні і не тільки рішення стають більш вживаними серед вчителів та учнів. Проте, існують певні недоліки та проблеми у сфері розробки програмного забезпечення для навчальних закладів.[3]

Одним з найбільших недоліків є відсутність гнучкості у багатьох програмних продуктах. Більшість ігор та програм мають обмежену кількість завдань або обмежені можливості для додавання нових завдань. Це може створювати труднощі для вчителів, які прагнуть розробляти власні матеріали для своїх учнів.

Також це є проблемою з точки зору навчальних програм, які є затвердженими Міністерством освіти і науки України.[2] Багато ІТ рішень, націлених на навчання, мають свою власну програму, яка не є затвердженою жодним контролюючим органом і залежить лише від фантазії розробників застосунку.

Ще однією проблемою є складність використання програмного забезпечення для вчителів та учнів. Деякі програми можуть бути занадто складними у використанні для дітей та вчителів, які можуть не мати достатнього досвіду у використанні ІТ-технологій.[6]

Щоб вирішити ці проблеми можна зосередитись на таких якостях нової системи навчання: гнучність, легкість у налаштуванні, зручне та інтуїтивно зрозуміле використання.

В рамках дипломного проєкту було обрано створення гнучкого та простого у використанні застосунку-гри для вчителів та учнів, який дозволить

додавати та змінювати завдання, підлаштовуючись під будь-яку навчальну программу, предмет, рівень вивчення предмету, клас, тощо.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації інтерактивної гри для навчання у школі. Назва проекту – Challenger. В подальшому я буду посилатись на дану розробку саме під такою назвою. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Серед відомих алгоритмічних рішень для інтерактивного навчання можна виділити наступні методи.

Розглянемо створення курсу, повного ігрових завдань. Прикладом такого підходу можу слугувати відомий застосунок для вивчення мов – “Duolingo”. Цей метод є цікавим, але йому не вистачає гнучкості і можливості створювати завдання, націлені на різні навчальні дисципліни.

Розглянемо гру у форматі змагань між учнями. Прикладом такого методу є платформа “Kahoot!”. Бажання стати переможцем і першим дістатись бажаної мети завжди слугували хорошою мотивацією для дітей та дорослих. Було вирішено обрати саме цей формат.[6] Однак, я вважаю, що постійна конкуренція не завжди є хорошим рішенням і що окрім цього треба вчити дітей працювати у команді та підтримувати один одного. Тому також буде додано командний режим, який дозволить згуртувати учнів у команду на шляху до спільної мети.[1]

Розглянемо відомі технічні рішення.

Розглянемо мобільний застосунок для платформ Android та iOS. Більшість застосунків, спрямованих на навчання, використовують саме мобільні платформи. І хоча я вважаю мобільні застосунки дуже зручними, я не

вважаю доречним використовувати дане технічне рішення для шкільного навчання.

Причина полягає у великій спокусі для учня відволіктись від теми уроку і зайти до розважального застосунку. До того ж, телефони здатні відволікати від навчання дуже сильно завдяки різним сповіщенням, повідомленням та нагадуванням, які миттєво перемикають увагу від попередньої роботи.[7]

Розглянемо ігри, розроблені за допомогою ігрового двигуна (Unity, Unreal Engine, тощо). Один з найбільш відомих і широко використовуваних засобів для розробки ігрових додатків є Unity. Unity - це платформа для створення ігор та інших інтерактивних додатків, яка дозволяє розробникам створювати графічні ігри, 3D-моделі та анімацію, а також використовувати сценарії, щоб керувати поведінкою об'єктів в грі. Не менш цікавим є рішення Unreal Engine, який також дозволяє розробникам створювати ігри з високоякісною графікою та повною взаємодією з користувачем. І хоча ці засоби ідеально підходять для створення високоякісної гри, але у Challenger цінується простота, легкість у використанні та доступність. Далеко не кожний учень має свій персональний комп’ютер і навіть не кожний вчитель має доступ до нього у школі. Дуже часто шкільне обладнання використовує застаріле програмне забезпечення, тощо.

Розглянемо гру з використанням технології віртуальної реальності. Технологія віртуальної реальності захоплює і стає новим трендом у розробці ігор. Ця технологія здатна забезпечити найвищий рівень взаємодії з користувачем, відсутність сторонніх сповіщень, дуже велику зацікавленість дітей. Найбільший і дуже суттєвий недолік даного рішення – складність та надвисока ціна обладнання, яку не зможуть собі дозволити школи чи учні.

Розглянемо веб застосунок. Веб застосунок із архітектурою клієнт-сервер є оптимальним технологічним засобом для створення навчальної гри. Веб-застосунки можна запускати на будь-якому пристрої, що підключений до Інтернету, такому як комп'ютер, телефон або планшет. Це означає, що школярі можуть отримати доступ до нього дуже легко. Використання веб технологій

не потребує особливих знань та навичок, є безкоштовним і доступним. Вони можуть містити інтерактивні елементи, такі як відео, аудіо, зображення, анімації, тощо. Це дозволить зробити гру цікавою без використання ігрового двигуна. Клієнт серверна архітектура дозволяє ефективно зберігати матеріалу без ризику втратити їх через змін пристрою.

Обравши веб застосунок, як найбільш оптимальне технічне рішення, розглянемо можливі архітектурні рішення. [8]

Монолітна архітектура - традиційний підхід, де весь застосунок розгортається як єдиний, самодостатній блок. Всі компоненти, такі як фронтенд, бекенд, база даних і інші, об'єднуються в одну єдину програму. Клієнт-серверна архітектура дозволить розподіли застосунок між клієнтом та сервером. Взаємодія відбувається за допомогою API.

Мікросервісна архітектура – підхід, при якому застосунок розбивається на набір невеликих, самостійних сервісів, які працюють разом. Кожен сервіс виконує конкретну функцію і має власну базу даних. Сервіси взаємодіють між собою за допомогою API. Це дозволяє більшу гнучкість та масштабованість, але вимагає складнішого управління. Даний проєкт не є настільки складним і не має багато блоків, які мало б сенс розбити на різні сервіси. Тому дана архітектура не буде застосована для розробки застосунку.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

1.3.2.1 Мова програмування

Опишемо різні мови програмування, які можна використати для розробки веб застосунку та зробимо їх порівняльний аналіз.

Java - це мова програмування, яка є однією з найпопулярніших мов серед програмістів. Вона має дуже велику підтримку, багато сторонніх бібліотек, фреймворків та інструментів розробки. Java є об'єктно-орієнтованою мовою зі статичною типізацією, що дозволяє писати масштабовані, стабільні і безпечні застосунки.[9]

JavaScript - це мова програмування, яка використовується в основному для розробки фронтенду. Вона має динамічну типізацію та є об'єктно-орієнтованою мовою, але також підтримує функціональне програмування. [10] JavaScript має велику кількість сторонніх бібліотек та фреймворків для розробки веб-застосунків.[11]

Python - це мова програмування, яка має простий синтаксис та дуже високий рівень абстракції. Python є динамічною мовою зі збиранням сміття, яка підтримує об'єктно-орієнтоване та функціональне програмування. Вона має велику кількість бібліотек, що дозволяє легко та швидко розробляти різноманітні застосунки.

Тепер можемо порівняти ці мови програмування за наступними критеріями:

Спрощення розробки: Python має простий та лаконічний синтаксис, що дозволяє швидко розробляти застосунки. JavaScript та Java також мають зручний синтаксис, але не такий простий, як у Python.

Швидкодія: Java є найшвидшою мовою серед трьох.

Кросплатформеність: Java є кросплатформеною мовою програмування.

Використання у веб розробці: серед цих трьох мов програмування, JavaScript є стандартним вибором для веб розробки, підтримує багато бібліотек для цього, використовує фреймворки, які є стандартом в індустрії.

Таким чином, було обрано використовувати мову програмування Java для написання бекенду і мову програмування JavaScript для фронтенду.

1.3.2.2 Фрейморки для бекенд розробки мовою програмування Java

Для спрощення розробки можна використовувати Spring Framework, Hibernate, Struts, Play Framework та Dropwizard. Spring Framework має вбудовану систему впровадження залежностей, що дозволяє зменшити кількість написаного коду та прискорити розробку. Крім того, Spring Framework надає безліч інших корисних функцій, таких як обробка HTTP-запитів, підключення до баз даних та інтеграція з іншими сервісами.[12] Hibernate надає можливість простого та зручного взаємодії з базами даних,

Struts забезпечує більшу контрольованість архітектури додатку, Play Framework має вбудований механізм конфігурації та маршрутизації HTTP-запитів, а Dropwizard має набір підключених бібліотек для розробки веб-сервісів.

Таким чином, для спрощення бекенд розробки було обрано Spring Framework, так як саме цей фреймворк є найбільш корисним у даному проекті і може надати багато переваг при розробці.[13]

1.3.2.3 Фрейморки та бібліотеки для фронтенд розробки мовою програмування JavaScript

Для розробки фронтенду використовують React, Angular, Vue.js, Ember.js, Backbone.js. Angular надає широкий функціонал для компонентної розробки, Vue.js має дуже легкий та простий використанні фреймворк, Ember.js надає стандартизований підхід до розробки додатків. React дозволяє розробникам писати код на JSX, що дозволяє поєднувати HTML та JavaScript в одному файлі. Крім того, React має вбудовану систему стану, що дозволяє легко керувати динамічними змінами на стороні клієнта та підвищити продуктивність. [11]

Для розробки було обрано бібліотеку React, так як вона дуже добре підходить для компонентної фронтенд розробки і має високу ефективність.

1.3.2.3 IDE

Інтегроване середовище розробки (IDE) - це програмне забезпечення, що допомагає програмістам створювати, налагоджувати та тестувати програмне забезпечення. Розглянемо декілька популярних IDE для розробки на Java:

Eclipse: безкоштовна та відкрита IDE, яка підтримує Java, а також інші мови програмування. Eclipse має багатий набір функцій, включаючи інтеграцію з системами керування версіями, налагоджувач та засіб профілювання. Він також має велику кількість плагінів, які можуть бути встановлені для підтримки різних типів проектів.

IntelliJ IDEA: це інша популярна IDE для розробки на Java. IntelliJ IDEA забезпечує широкий набір функцій, включаючи автозавершення коду, інтеграцію з системами керування версіями та вбудований налагоджувач.

NetBeans: це безкоштовна та відкрита IDE, яка підтримує Java, має вбудовані засоби для розробки веб-застосунків, таких як HTML та JavaScript, а також має засоби для налагодження та профілювання.

В порівнянні з цими IDE, можна зазначити, що Eclipse має велику кількість плагінів, що дозволяє розширювати його функціональність. IntelliJ IDEA забезпечує велику кількість автоматизованих функцій, які допомагають підвищити продуктивність програмістів. NetBeans є простою та ефектовною IDE, яка є перевірена часом.

Для розробки даного застосунку я скористаюсь IntelliJ IDEA, так як я маю великий досвід роботи саме з цим середовищем розробки.

Для розробки фронтенду я використовую WebStorm – інший продукт від виробника JetBrains, який має дуже широкий функціонал і дуже спрощує розробку веб застосунків. WebStorm є єдиною IDE для веб розробки, з якою мені зручно працювати. Я вважаю роботу з IDE більш зручною ніж із редактором коду, тому обираю її, а не популярну програму – Visual Studio Code.

1.3.2.4 Інші допоміжні засоби розробки

Postman - це інструмент для тестування API, який дозволяє розробникам взаємодіяти з API та перевіряти його відповідь. Postman забезпечує можливість виконання запитів, перегляду відповідей, додавання параметрів, обробки помилок, тощо.

Git - це система контролю версій, яка дозволяє розробникам зберігати код у віддаленому репозиторії та керувати його версіями. [14]

1.3.2.5 Система для авторизації користувачів

Система аутентифікації є доволі складною і потребує особливого захисту. В той час, як великі програмні продукти можуть дозволити собі зробити надійний механізм для цього, для маленьких проектів доцільно буде

скористатись спеціальним вузьконаправленим програмним забезпеченням від третьої особи. Auth0 та Google Auth – це дуже популярні інструменти аутентифікації для веб-додатків. Ось декілька порівняльних характеристик між ними:

Підтримка протоколів аутентифікації: Auth0 підтримує більше протоколів аутентифікації, таких як OAuth, OpenID Connect, SAML, LDAP, і бази даних користувачів. Google Auth підтримує OAuth та OpenID Connect, але не підтримує інші протоколи.[15]

Масштабованість: Auth0 дозволяє розгортання на приватних серверах або у хмарі. Google Auth працює тільки на хмарній платформі Google Cloud.

Легкість використання: Google Auth має дуже простий інтерфейс та легко інтегрується з іншими сервісами Google. Auth0 має більш складний інтерфейс та може потребувати додаткової конфігурації для інтеграції з іншими сервісами.

Вартість: Google Auth зазвичай коштує менше, ніж Auth0, особливо для малих проектів.

Безпека: Обидва сервіси мають добре розроблені механізми захисту від кібератак та інших загроз.

Враховуючи масштаби проекту, я обрала Auth0 для забезпечення аутентифікації користувача.

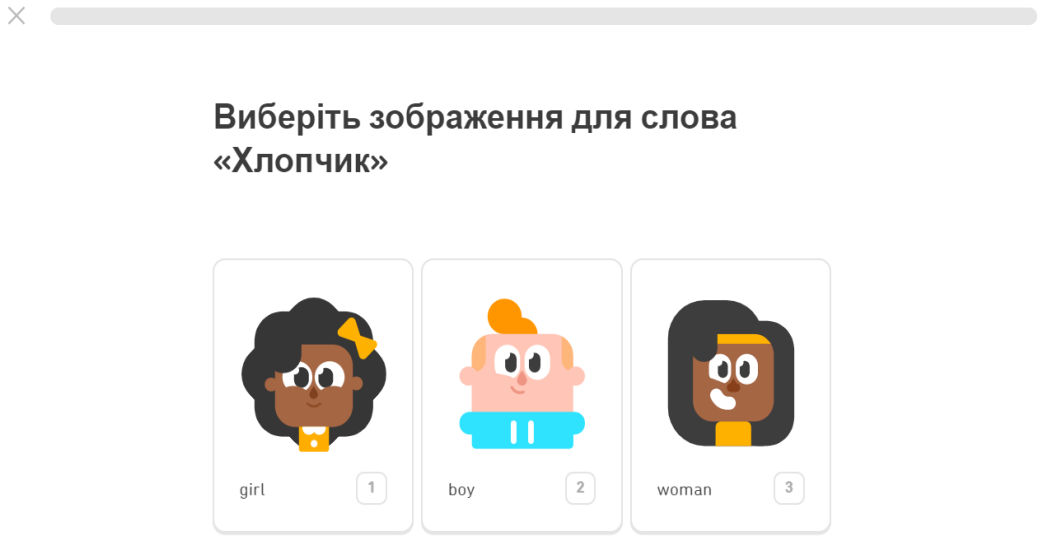
1.3.3 Аналіз відомих програмних продуктів

Duolingo - це додаток для мобільних пристроїв і онлайн-платформа для інтерактивного навчання мов. Додаток має понад 300 мільйонів користувачів у всьому світі і пропонує навчання більше ніж 40 мов.

Оцінюючи Duolingo як додаток для інтерактивного навчання, можна відзначити декілька переваг і недоліків. Цей застосунок є одним з найпопулярніших застосунків для вивчення мов, використовує різноманітні типи завдань, які дозволяють тренувати різні навчички.

Серед найбільших переваг Duolingo, я хочу виділити систему винагород і заохочень. Користувачі збирають різні медалі, нагороди, трофеї, тощо. Також Duolingo намагається тримати користувача у процесі навчання постійно за допомогою відслідковування прогресу учня день за днем, що є хорошим мотивуючим фактором. Перед початком навчання додаток персоналізує план уроків враховуючи ваш рівень знання мови.

Найбільший недолік Duolingo – неможливість налаштування додатку повністю під свої потреби. Навчальний план залежить лише від рівня користувача і не може бути змінений за бажанням.



Риунок. 1.1 – Урок у застосунку Duolingo

Kahoot! - це додаток, який дозволяє створювати та проводити інтерактивні тести, гри та опитування. Користувачі можуть приєднатися до гри, створеної вчителем та проходити опитування. Основною метою додатку Kahoot! є покращення навчання та залучення студентів до процесу навчання. Цей застосунок є інтерактивним, залучає дітей до конкурентного змагання, цікавим. Kahoot! Дозволяє відстежувати прогрес учня та може візуалізувати його.

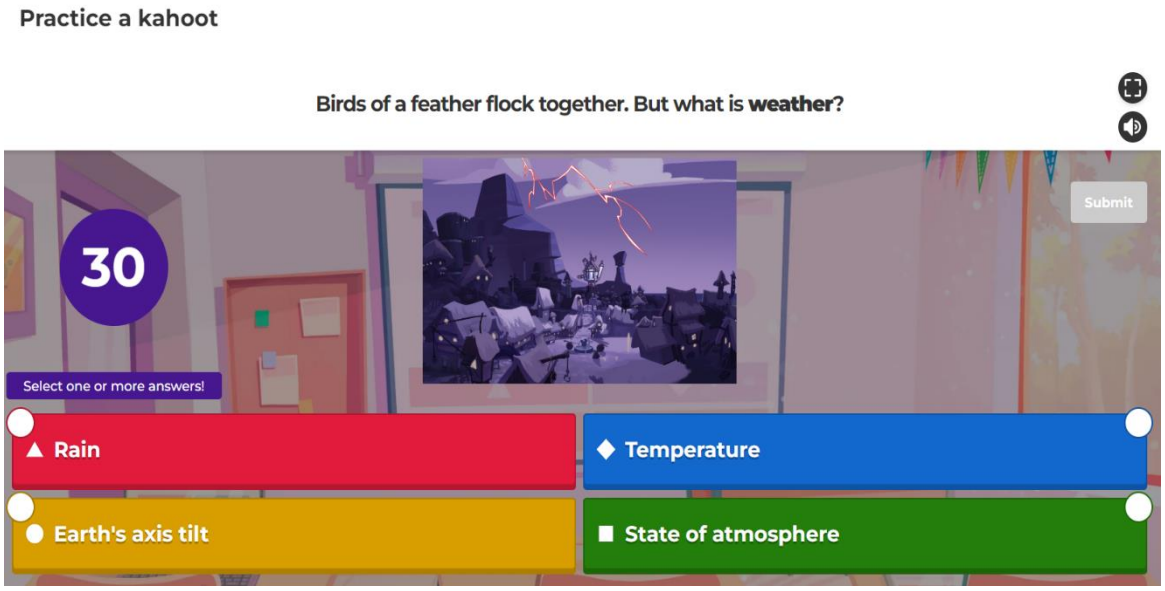


Рисунок 1.2 – Урок у застосунку Kahoot!

Khan Academy - це застосунок, який надає безкоштовні уроки з багатьох предметів. Він використовує відеоуроки для викладання матеріалу. Відеоуроки дозволяють студентам переглядати матеріал у своєму власному темпі. Ще однією корисною функцією Khan Academy є можливість виконувати вправи та тести, які допомагають студентам перевірити своє розуміння матеріалу та отримати зворотний зв'язок про їхні досягнення.

Для порівняння курсової роботи з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Challenger	Duolingo	Kahoot!	Khan Academy	Пояснення
Ігрова механіка	+	+	+	-	Можливість проводити тестування/навчання у ігровій формі
Тестування учнів	+	-	+	+	Можливість оцінювати знання учнів
Вивчення матеріалу	+	+	-	+	Використання для подання нового матеріалу

Продовження таблиці 1.1

Функціонал	Challenger	Duolingo	Kahoot!	Khan Academy	Пояснення
Вивчення різних предметів	+	-	+	Тільки предмети, наявні у додатку	Підходить для вивчення будь-якого предмету
Заохочення прогресу	-	+	+	-	Наявна система винагород і заохочень
Може бути підлаштована під навчальну програму	+	-	+	-	Викладання матеріалу у застосунку залежить лише від педагога
Не потребує додаткових зусиль з боку педагога на підготовку	-	+	-	+	Матеріал вже наявний на платформі і не потребує додаткової роботи з ним

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є запуск, створення та проходження гри. Більше функцій можна побачити на рисунку 1.3.

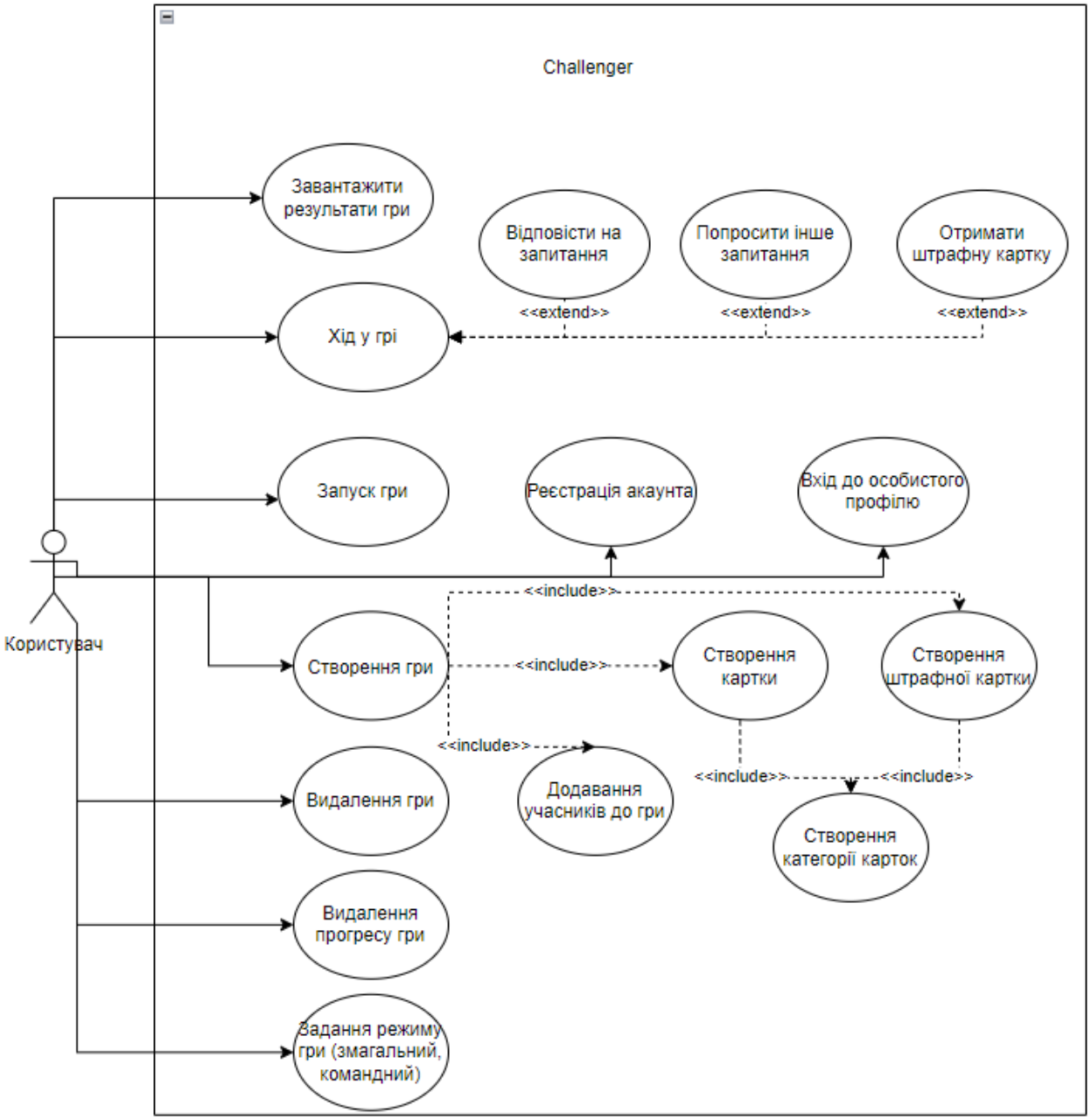


Рисунок 1.3 – Діаграма варіантів використання

В таблицях 1.2 - 1.17 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Реєстрація акаунта
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає зареєструватися

Продовження таблиці 1.2

Pre-conditions	-
Flow of Events	1) Користувач переходить на сторінку реєстрації. 2) В поля для реєстрації вводяться відповідні дані: пошта користувача, ім’я, прізвище, пароль. 3) Користувач натискає кнопку реєстрації. 4) З’являється повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу.
Extension	В випадку введення не коректних даних, кнопка реєстрації стає неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом.
Post-Condition	Створення сторінки користувача, перехід на сторінку входу

Таблиця 1.3 - Варіант використання UC-2

Use case name	Вхід до особистого профілю
Use case ID	UC-02
Goals	Вхід користувача до системи
Actors	Гість (неавторизований користувач)
Trigger	Користувач бажає увійти до системи
Pre-conditions	Користувач зареєстрований
Flow of Events	1) Користувач переходить на сторінку логіну. 2) В поля для логіну вводяться відповідні дані: пошта користувача, пароль. 3) Користувач натискає кнопку входу. 4) Він перенаправляється на сторінку профілю.
Extension	В випадку введення не коректних даних, кнопка входу стає неактивною. Якщо якесь конкретне поле введено некоректно, то воно підсвічується червоним надписом. Якщо користувач ввів некоректні логін чи пароль – буде виведено відповідне повідомлення про помилку авторизації.
Post-Condition	Перехід на сторінку профіля користувача, авторизація

Таблиця 1.4 - Варіант використання UC-3

Use case name	Запуск гри
Use case ID	UC-03
Goals	Почати гру
Actors	Авторизований користувач
Trigger	Користувач натискає на кнопку запуску гри
Pre-conditions	Користувач зареєстрований; гра, яку він хоче запустити, існує
Flow of Events	1) Користувач переходить на сторінку свого акаунту. 2) Користувач натискає на одну із створених ігор. 3) Користувач перенаправляється на сторінку гри.
Extension	-
Post-Condition	Перехід на сторінку гри, початок процесу гри.

Таблиця 1.5 - Варіант використання UC-4

Use case name	Хід у грі
Use case ID	UC-04
Goals	Виконати поточний хід
Actors	Авторизований користувач
Trigger	Користувачу випадає завдання для його ролі
Pre-conditions	Гра запущена
Flow of Events	1) Користувач отримує завдання. 2) Користувач дає відповідь на завдання. 3) Користувач обирає один з трьох варіантів: правильна відповідь, оновити, неправильна відповідь. 4) Користувач натискає відповідну кнопку.
Extension	-
Post-Condition	Перераховуються бали користувача, випадає наступна картка.

Таблиця 1.6 - Варіант використання UC-5

Use case name	Отримання нової картки
Use case ID	UC-05
Goals	Перехід до наступної картки
Actors	Користувач
Trigger	Учаснику гри випадає картка, коли приходить його черга у грі
Pre-conditions	Користувач випало завдання
Flow of Events	1) Якщо завдання не є можливим до виконання з технічних причин – студент натискає на кнопку «оновити» і отримує нову картку. 2) Якщо студент дає правильну відповідь – він натискає на кнопку «продовжити». 3) Якщо студент дає неправильну відповідь – він натискає кнопку «штраф»
Extension	-
Post-Condition	Після натискання кнопки «продовжити» студенту зараховуються бали за виконане завдання, і наступна картка випадає для наступного учня. Після натискання кнопки «штраф» студент отримує картку із категорії, позначеної як штрафне завдання.

Таблиця 1.7 - Варіант використання UC-6

Use case name	Успішна відповідь на завдання картки
Use case ID	UC-06
Goals	Перехід до наступної картки
Actors	Користувач
Trigger	Користувач відповів на завдання правильно.
Pre-conditions	Користувач випало завдання.
Flow of Events	1) Користувач натискає на кнопку «продовжити».
Extension	-

Продовження таблиці 1.7

Post-Condition	Після натискання кнопки «продовжити» студенту зараховуються бали за виконане завдання, і наступна картка випадає для наступного учня.

Таблиця 1.8 - Варіант використання UC-7

Use case name	Отримання штрафної картки
Use case ID	UC-07
Goals	Перехід до наступної картки
Actors	Користувач
Trigger	Користувач відповів на завдання неправильно.
Pre-conditions	Користувач випало завдання
Flow of Events	1) Користувач натискає кнопку «штраф». 2) Користувач отримує картку із категорії, позначеної як штрафна. Користувач виконує завдання штрафної картки.
Extension	-
Post-Condition	Користувачу зараховуються бали за виконану штрафну картку. Ці бали можуть бути від’ємними.

Таблиця 1.9 - Варіант використання UC-8

Use case name	Створення гри
Use case ID	UC-08
Goals	Створення нової гри
Actors	Користувач
Trigger	Користувач хоче додати гру
Pre-conditions	Користувач є авторизованим

Продовження таблиці 1.9

Flow of Events	1) Користувач натискає на кнопку додавання гри. 2) Він заповнює поля: назва гри, опис, кількість балів для перемоги у грі, обирає командний або змагальний режим гри. 3) Він натискає кнопку «Продовжити». 4) Він заповнює поля для додання категорій: назва, колір. 5) За бажанням він редагує наявні штрафні категорії шляхом зміни їх назви чи кольору. 6) Він натискає кнопку «Продовжити». 7) Він додає учасників гри заповнюючи поле «Роль». 8) Він натискає кнопку «Продовжити». 9) Він додає картки із завданнями до гри заповнюючи наступні поля: опис, бали та обирає категорію із випадаючого списку. 10) Він натискає кнопку «Продовжити».
Extension	В випадку введення не коректних даних, користувач побачить повідомлення про помилку. Неможливо додати гру без категорій і карток. Всі поля є обов'язковими до заповнення.
Post-Condition	Користувач бачить повідомлення про успішне додавання гри. Додавання нової гри, гра з'являється на сторінці «Мої ігри».

Таблиця 1.10 - Варіант використання UC-9

Use case name	Видалення гри
Use case ID	UC-09
Goals	Видалення гри
Actors	Користувач
Trigger	Користувач хоче видалити гру
Pre-conditions	Користувач є авторизованим, гра, яку він хоче видалити існує і була створена ним.

Продовження таблиці 1.10

Flow of Events	1) Користувач переходить на сторінку «Мої ігри». 2) Він натискає на кнопку видалення на грі (кнопка у вигляді сміттєвого баку), яку хочу видалити. 3) На екрані з повідомленням, яке попереджає про наслідки цієї дії він натискає на кнопку «Так».
Extension	-
Post-Condition	Видалення гри, гра зникає зі списку на сторінці «Мої ігри».

Таблиця 1.11 - Варіант використання UC-10

Use case name	Видалення прогресу гри
Use case ID	UC-10
Goals	Видалення гри
Actors	Користувач
Trigger	Користувач хоче видалити прогрес гри
Pre-conditions	Користувач є авторизованим, гра, яку він хоче стерти існує і була створена ним.
Flow of Events	1) Користувач переходить на сторінку «Мої ігри». 2) Він натискає на кнопку видалення прогресу (кнопка у вигляді ластику) на грі, яку хочу стерти. 3) На екрані з повідомленням, яке попереджає про наслідки цієї дії він натискає на кнопку «Так».
Extension	-
Post-Condition	Гра повертається до свого початкового стану, весь прогрес і бали учасників видаляються.

Таблиця 1.12 - Варіант використання UC-11

Use case name	Зміна режиму гри
Use case ID	UC-11
Goals	Зміна режиму гри

Продовження таблиці 1.12

Actors	Користувач
Trigger	Користувач хоче змінити режим гри
Pre-conditions	Користувач є авторизованим, гра, яку він хоче змінити існує і була створена ним.
Flow of Events	1) Користувач переходить на сторінку «Мої ігри». 2) Він натискає на кнопку зміни режиму гри (кнопка у вигляді перемикача) на грі, яку хочу змінити. 3) На екрані з повідомленням, яке попереджає про наслідки цієї дії він натискає на кнопку «Так».
Extension	-
Post-Condition	Режим гри перемикається на протилежний (командний режим змінюється на змагальний, а змагальний на командний).

Таблиця 1.13 - Варіант використання UC-12

Use case name	Завантаження результатів гри
Use case ID	UC-12
Goals	Завантажити файл із результатами гри
Actors	Користувач
Trigger	Один із учасників гри досяг фінішу і користувач натиснув кнопку «Завантажити результати».
Pre-conditions	Один із учасників гри досяг фінішу.
Flow of Events	1) Користувач натискає кнопку «Завантажити результати».
Extension	-
Post-Condition	Після натискання кнопки на девайс користувача завантажується текстовий файл із переможцями гри та детальним описом кожного ходу.

Таблиця 1.14 - Варіант використання UC-13

Use case name	Створення картки

Продовження таблиці 1.14

Use case ID	UC-13
Goals	Додати нову картку при створенні гри
Actors	Користувач
Trigger	Користувач натиснув кнопку «Додати картку» на сторінці створення нової гри
Pre-conditions	Користувач знаходиться на етапі створення гри «Додавання карток»
Flow of Events	1) Користувач заповнює наступні поля у формі: опис, бали та обирає категорію із випадаючого списку.
Extension	Якщо якісь поля введені некоректно, тобто опис має менше 5 або більше 1000 символів або категорія не обрана, то виведеться повідомлення про помилку, поки вони не будуть виправлені.
Post-Condition	Нова картка додається до списку карток гри.

Таблиця 1.15- Варіант використання UC-14

Use case name	Створення штрафної картки
Use case ID	UC-14
Goals	Додати нову картку при створенні гри
Actors	Користувач
Trigger	Користувач натиснув кнопку «Додати картку» на сторінці створення нової гри
Pre-conditions	Користувач знаходиться на етапі створення гри «Додавання карток»
Flow of Events	1) Користувач заповнює наступні поля у формі: опис, бали та обирає штрафну категорію із випадаючого списку.
Extension	Якщо якісь поля введені некоректно, тобто опис має менше 5 або більше 1000 символів або категорія не обрана, то виведеться повідомлення про помилку, поки вони не будуть виправлені.
Post-Condition	Нова штрафна картка додається до списку карток гри.

Таблиця 1.16- Варіант використання UC-15

Use case name	Створення категорії карток
Use case ID	UC-15
Goals	Додати нову категорію карток при створенні гри
Actors	Користувач
Trigger	Користувач натиснув кнопку «Додати категорію» на сторінці створення нової гри
Pre-conditions	Користувач знаходиться на етапі створення гри «Додавання категорій карток»
Flow of Events	1) Користувач заповнює наступні поля у формі: назва категорії. 2) Користувач обирає колір категорії у полі вибору кольору.
Extension	Якщо якісь поля введені некоректно, тобто опис має менше 3 або більше 50 символів, то виведеться повідомлення про помилку, поки вони не будуть виправлені.
Post-Condition	Нова категорія додається до списку категорій гри.

Таблиця 1.17- Варіант використання UC-16

Use case name	Додавання учасників до гри
Use case ID	UC-16
Goals	Додати учасників до гри
Actors	Користувач
Trigger	Користувач натиснув кнопку «Додати учасника» на сторінці створення нової гри
Pre-conditions	Користувач знаходиться на етапі створення гри «Додавання учасників»
Flow of Events	1) Користувач заповнює поле «назва» у формі. 2) Користувач натискає кнопку «Далі».
Extension	Якщо якісь поля введені некоректно, тобто назва має менше 3 або більше 20 символів, то виведеться повідомлення про помилку, поки вони не будуть виправлені.

Продовження таблиці 1.17

Post-Condition	До гри додаються учасники.

1.4.1 Розроблення функціональних вимог

Розробимо модель вимог до програмного забезпечення. Модель представлена на рисунках 1.4 – 1.6 по частинах.

#	Id	Name	Text	Risk
1	1	1 Авторизація користувача		Low
2	1.2	1.2 Створення нового акаунту	Створення нового акаунту за допомогою імені, прізвища, нікнейма, електронної пошти та паролю	Medium
3	1.2.1	1.2.1 Введення імені		Low
4	1.2.1.1	1.2.1.1 Первірка на довжину (від 3 до 100 символів)		Low
5	1.2.2	1.2.2 Введення прізвища		Low
6	1.2.2.1	1.2.2.1 Перевірка на довжину (від 3 до 100 символів)		Low
7	1.2.3	1.2.3 Введення нікнейму		Low
8	1.2.3.1	1.2.3.1 Перевірка на довжину (від 3 до 100 символів)		Low
9	1.2.4	1.2.4 Введення адреси електронної пошти		Low
10	1.2.4.1	1.2.4.1 Перевірка на довжину (від 1 до 50 символів)		Low
11	1.2.4.2	1.2.4.2 Перевірка на валідність адреси едектронної пошти	Перевірка, що адреса відповідає дійсній адресі електронної пошти	Low
12	1.2.5	1.2.5 Введення паролю	Перевірка складності паролю	Low
13	1.2.5.1	1.2.5.1 Перевірка на довжину (мінімум 8 символів)		Low
14	1.2.5.2	1.2.5.2 Перевірка наявності спеціального символу		Low
15	1.2.5.3	1.2.5.3 Перевірка наявності цифри		Low
16	1.2.5.4	1.2.5.4 Перевірка наявності великої літери		Low
17	1.2.6	1.2.6 Додавання користувача у Auth0		Medium
18	1.1	1.1 Вхід до акаунту	Вхід до акаунту за допомогою електронної пошти і паролю	High
19	1.1.1	1.1.1 Введення даних для входу (логін і пароль)		Low
20	1.1.1.1	1.1.1.1 Перевірка на існування облікового запису	Якщо в системі є користувач з таким логіном і паролем - користувача буде авторизовано	Medium
21	2	2 Створення гри		Low
22	2.1	2.1 Додавання базової інформації про гру	Додавання назви гри, опису, режиму та мети гри	Low
23	2.1.1	2.1.1 Додавання назви гри		Low
24	2.1.1.1	2.1.1.1 Перевірка на довжину (від 3 до 20 символів)		Low
25	2.1.2	2.1.2 Додавання опису гри		Low
26	2.1.2.1	2.1.2.1 Перевірка на довжину (від 5 до 1000 символів)		Low
27	2.1.3	2.1.3 Додавання мети гри		Low
28	2.1.3.1	2.1.3.1 Перевірка, що мета більше ніж нуль		Low
29	2.1.4	2.1.4 Вибір режиму гри (командний, змагальний)		Low
30	2.2	2.2 Додавання категорій карток	Додавання списку категорій, які складаються з назви, кольору та типу (ігрові та штрафні)	Low
31	2.2.1	2.2.1 Додавання назви категорії		Low
32	2.2.1.1	2.2.1.1 Перевірка на довжину (від 3 до 50 символів)		Low
33	2.2.2	2.2.2 Додавання кольору категорії		Low

Рисунок 1.4 – Модель вимог ч. 1

34	2.2.3	2.2.3 Редагування штрафних категорій		Low
35	2.2.4	2.2.4 Видалення категорії		Low
36	2.3	2.3 Додавання карток	Додавання карток, які складаються з опису, категорії, кількості балів	Low
37	2.3.1	2.3.1 Додавання опису картки		Low
38	2.3.1.1	2.3.1.1 Перевірка довжини (від 5 до 200 символів)		Low
39	2.3.2	2.3.2 Вибір категорії картки із списку доданих категорій		Low
40	2.3.2.1	2.3.2.1 Перевірка того, що існуюча категрія обрана		Low
41	2.3.3	2.3.3 Додавання кількості балів за картку		Low
42	2.3.4	2.3.4 Видалення картки		Low
43	2.4	2.4 Додавання ролей	Додавання ролей, які складаються з назви	Low
44	2.4.1	2.4.1 Додавання назви ролі		Low
45	2.4.1.1	2.4.1.1 Перевірка на довжину (від 3 до 20 символів)		Low
46	2.4.2	2.4.2 Видалення ролі		Low
47	3	3 Управління сутністю гри		Low
48	3.1	3.1 Зміна режиму гри	Перемикання режиму від комендного до змагального і навпаки	Low
49	3.1.1	3.1.1 Отримання підтвердження дії від користувача		Low
50	3.2	3.2 Видалення прогресу гри	Видаленння балів всіх ролей гри	Low
51	3.2.1	3.2.1 Отримання підтвердження дії від користувача		Low
52	3.3	3.3 Видалення гри	Видалення гри	Low
53	3.3.1	3.3.1 Отримання підтвердження дії від користувача		Low
54	4	4 Механіка гри		Low
55	4.1	4.1 Отримання наступної картки	Отримання наступної картки для наступної ролі у разі правильної відповіді	Low
56	4.1.1	4.1.1 Зарахування балів поточній ролі за останню картку		Low
57	4.1.2	4.1.2 Запис ходу до статистики		Low
58	4.1.3	4.1.3 Зміна поточної ролі на наступну у черзі		Low
59	4.1.4	4.1.4 Видача нової картки для поточної ролі		Low
60	4.1.5	4.1.5 Відображення зміни прогресу ролі у графічному інтерфейсі		Low
61	4.2	4.2 Оновлення картки для поточної ролі	Оновлення картки для поточної ролі	Low
62	4.2.1	4.2.1 Запис ходу до статистики		Low
63	4.2.2	4.2.2 Видача нової картки для поточної ролі		Low

Рисунок 1.5 – Модель вимог ч. 2

64	4.3	4.3 Отримання штрафної картки	Отримання наступної штрафної картки для наступної ролі у разі неправильної відповіді	Low
65	4.3.1	4.3.1 Зарахування балів поточній ролі за останню картку		Low
66	4.3.2	4.3.2 Запис ходу до статистики		Low
67	4.3.3	4.3.3 Видача нової картки для поточної ролі		Low
68	4.3.4	4.3.4 Відображення зміни прогресу ролі у графічному інтерфейсі		Low
69	4.4	4.4 Керування балами	Додавання чи віднімання балів, в залежності від балів у картці	Low
70	4.5	4.5 Відображення прогресу гри	Відображення прогресу візуально на шкалі від 0% до 100%	Low
71	4.5.1	4.5.1 Відображення прогресу у вигляді ракети на ігровому ролі		Low
72	4.5.2	4.5.2 Відображення прогресу у вигляді шкали	Шкала заповнюєть у відсотковому співвідношенні мети гри до кількості балів набраних учасником	Low
73	4.5.3	4.5.3 Відображення прогресу у вигляді останньої планети		Low
74	4.5.4	4.5.4 Відображення прогресу у вигляді статусу учасника	Статус учасника позначається зірками (від 1 до 5)	Low
75	5	5 Завершення гри		Low
76	5.1	5.1 Завантаження результатів сесії гри	Завантаження результатів у текстовий файл	Low
77	5.2	5.2 Виведення результатів сесії гри на екран	Виведення таблиці з результатами гри на екран	Low
78	5.3	5.3 Продовження гри		Low
79	5.4	5.4 Завершення гри		Low

Рисунок 1.6 – Модель вимог ч. 3

Базуючись на складених варіантах використання в пункті 1.4 дипломної роботи та моделі вимог, можна сформулювати ряд функціональних вимог до програмного забезпечення, що розробляється. В таблицях 1.18 – 1.33 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.7.

Таблиця 1.18– Функціональна вимога FR-1

Назва	Реєстрація користувача
Опис	Система повинна надавати можливість реєстрації користувачеві шляхом введення пошти, паролю, ПІБ.

Таблиця 1.19– Функціональна вимога FR-2

Назва	Вхід до акаунту користувача
Опис	Система повинна надавати можливість логіну користувачеві шляхом введення пошти та паролю.

Таблиця 1.20– Функціональна вимога FR-3

Назва	Запуск гри у командному режимі
Опис	Система повинна надавати запускати гру у режимі, коли всі учні набирають бали разом і їх прогрес складається разом для досягнення фінішу.

Таблиця 1.21 – Функціональна вимога FR-4

Назва	Запуск гри у змагальному режимі
Опис	Система повинна надавати запускати гру у режимі, коли всі учні набирають свої бали і змагаються для досягнення фінішу першими. У цьому режимі і грі можуть бути переможці, які посідають призові місця.

Таблиця 1.22– Функціональна вимога FR-5

Назва	Отримання нової картки
Опис	Система повинна надавати можливість студенту отримати нову картку не штрафної категорії у разі, якщо завдання не є технічно можливим на даний момент.

Таблиця 1.23– Функціональна вимога FR-6

Назва	Успішна відповідь на завдання
Опис	Система повинна надавати можливість студенту отримати бали і завершити питання шляхом натискання відповідної кнопки.

Таблиця 1.24– Функціональна вимога FR-7

Назва	Отримання штрафної картки.
Опис	Система повинна надавати можливість студенту отримати картку із штрафної категорії шляхом натискання відповідної кнопки. Штраф може бути додатковим завданням, втратою балів, тощо.

Таблиця 1.25– Функціональна вимога FR-8

Назва	Створення гри
Опис	Система повинна надавати можливість створення гри користувачем шляхом заповнення назви, опису, мети, режиму, категорій, карток та учнів.

Таблиця 1.26– Функціональна вимога FR-9

Назва	Видалення гри
Опис	Система повинна надавати можливість видалення гри користувачем шляхом натискання відповідної кнопки.

Таблиця 1.27– Функціональна вимога FR-10

Назва	Видалення прогресу гри
Опис	Система повинна надавати можливість стирання прогресу гри, залишаючи її у тому стані, у якому вона була створена.

Таблиця 1.28– Функціональна вимога FR-11

Назва	Зміна режиму гри

Продовження таблиці 1.28

Опис	Система повинна надавати можливість зміни режиму гри з командного на змагальний і навпаки. Різниця у режимах гри має бути позначена у інтерфейсі, щоб користувач розумів який режим у гри зараз.

Таблиця 1.28– Функціональна вимога FR-12

Назва	Перегляд результатів гри
Опис	Система повинна надавати можливість переглянути переможців гри і запис кожного ходу.

Таблиця 1.29– Функціональна вимога FR-13

Назва	Завантаження результатів гри
Опис	Система повинна надавати можливість переглянути переможців гри і запис кожного ходу у автоматично згенерованому текстовому файлі, який можна завантажити.

Таблиця 1.30– Функціональна вимога FR-14

Назва	Створення картки
Опис	Система повинна надавати можливість створення картки шляхом заповнення назви, опису, категорії, кількості балів.

Таблиця 1.31– Функціональна вимога FR-15

Назва	Створення категорії
Опис	Система повинна надавати можливість створення категорії шляхом заповнення назви та кольору.

Таблиця 1.32– Функціональна вимога FR-16

Назва	Додавання учнів до гри

Продовження таблиці 1.32

Опис	Система повинна надавати можливість створення ролей для гри шляхом заповнення поля «Роль».

На рисунку 1.7 зображено матрицю трасування вимог.

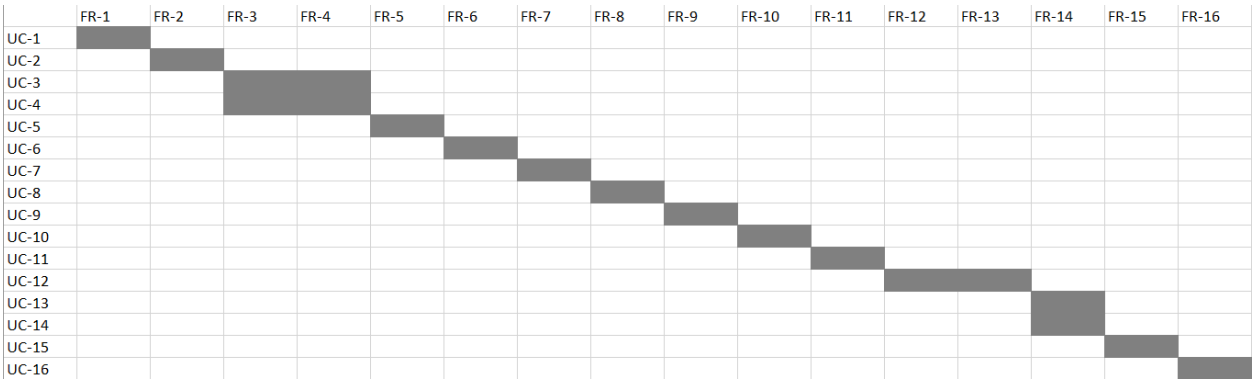


Рисунок 1.7 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Нефункціональними вимогами до даного застосунку є:

- інтуїтивний візуальний інтерфейс;
- простота використання;
- швидка робота;
- безпека даних користувачів;
- гнучкість у використанні;
- розширюваність;
- інтерфейс українською мовою.

1.5 Постановка задачі

Метою даної розробки є підвищення зацікавленості учнів у навчання шляхом створення веб застосунку, який буде корисним для інтерактивного навчання у цікавій для дітей формі. Для цього в застосунку має бути можливість створювати цікаві, яскраві ігри з системою нагороди і покарання.

Було розроблено основну тематику гри – космічні подорожі. В залежності від режиму гри, учасники або будуть змагатись у швидкості свої

космічних кораблів, або будуть екіпажом спільного корабля. Метою гри буде якнайшвидше дістатись від Землі до Сатурна, паралельно підвищуючи рівень досягнень свого персонажу.

Проаналізувавши варіанти використання, функціональні та нефункціональні вимоги, що було розроблено, було сформульовано наступні задачі для розробки продукту:

- авторизація користувача;
- додавання ігор;
- додавання карток із завданнями до ігор, включаючи звичайні та штрафні картки.
- створення черги з учнів, картки випадають згідно цієї черги;
- керування акаунтом користувача (зміна, видалення акаунту)
- керування всіма елементами гри (гра, категорія, картка, роль);
- реалізація ігрової механіки.

1.5 Висновки до розділу

У першому розділі дипломного проєкту було проаналізовано наявні підходи до інтерактивного навчання та програмні продукти, що вже представлені на ринку та завоювали велику популярність. Було проаналізовано сильні та слабкі сторони аналогових застосунків та підходів до розробки ПЗ, що дозволило сформулювати оптимальні вимоги для даного проєкту.

Головна відмінність цього проєкту від інших рішень – чітка орієнтація на використання у навчальних закладах.

Відповідно до цих даних було визначено варіанти використання, а також функціональні та нефункціональні вимоги, на базі яких було сформовано план розробки програмного забезпечення та які будуть враховані при виборі архітектури програмного забезпечення. Визначений комплекс задач, які потребують вирішення при розробці даного програмного забезпечення.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

У цьому розділі розглянемо варіанти поведінки користувачів у системі, опишемо детально кожен сценарій. Для опису бізнес процесів використовується BPMN модель.

Основний бізнес процес програмного забезпечення – хід у грі. Його опис зображений на рисунку 2.1.

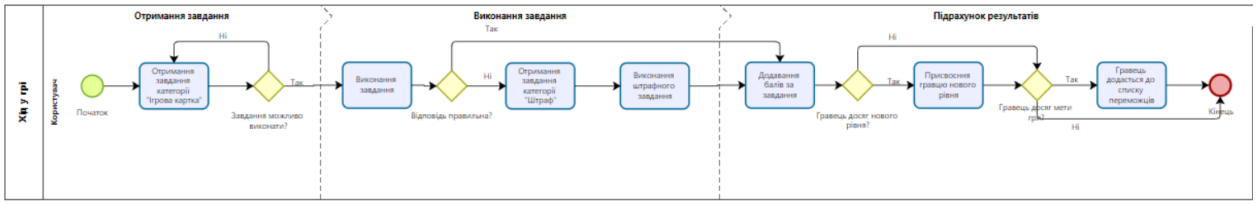


Рисунок 2.1 – Схема бізнес-процесу «Хід у грі»

Опис послідовності проходження гри:

Кожен хід виконує один учасник гри, чиє ім’я написано на сторінці. Всі учасники відповідають по черзі. Хід починається випадінням картки одного учасника і завершується закінченням циклу його завдань, що призведе до випадіння картки наступного учасника. У кожної картки є кількість балів за виконання завдання. Це число може бути як додатнім, так і від’ємним (має сенс використовувати разом із штрафними картками). У разі виконання завдання, ці бали додаються поточному учаснику, що наближає його до фінішу гри або віддаляє від нього. Після зарахування балів, учаснику можуть призначити новий рівень розвитку чи нову планету, до якої вони долетіли. У разі якщо учасник досяг фінішу, він записується як переможець і гра може бути завершена.

а) Користувач отримує картку із завданням, обрану випадково.

б) Користувач оцінює чи можливо виконати завдання (наприклад, якісь завдання можуть потребувати спеціального обладнання, погодних умов, тощо)

в) Якщо завдання неможливо виконати – користувач отримує нову картку на своє ім’я.

г) Користувач дає відповідь на завдання у картці і вчитель визначає його правильність. Якщо відповідь правильна:

1) Користувач натискає кнопку «Продовжити».

2) Користувачу зараховуються бали за виконане завдання.

Якщо учень відповів неправильно:

1) Учень натискає кнопку «Штраф».

2) Учень отримує нову картку із категорії «Штраф»

3) Учень виконує завдання із нової картки.

4) Користувачу зараховуються бали за виконане завдання.

ґ) Проводиться порівняння балів користувача із метою гри. У разі, якщо користувач доходить до фінішу – він займає місце серед переможців.

д) Завершення ходу.

Другий бізнес процес – створення гри. Це – багатоетапний процес, оскільки гра потребує власних категорій, карток, додавання до неї учасників, тощо. Процес створення гри описаний на рисунку 2.2.

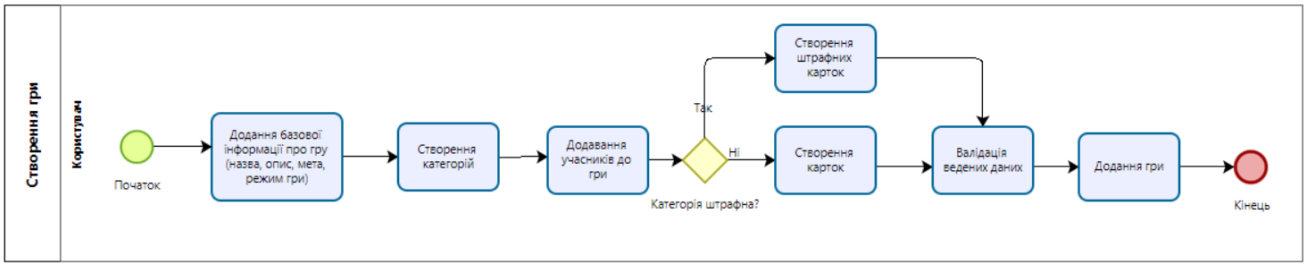


Рисунок 2.2 – Схема бізнес-процесу «Створення гри»

Опис послідовності створення гри:

− Користувач переходить на сторінку додавання гри.

− Користувач заповнює форму із наступними полями: назва гри, опис гри, мета, чи це команда гра.

– Користувач додає категорії. Кожна категорія має назву, опис, колір (те, як вона буде відображатись для учнів).

– Штрафні категорії додаються автоматично, користувач може їх редагувати.

– Додавання учасників до гри.

– Створення карток для різних категорій. Це включає назву, опис, кількість балів, вибір категорії.

– Валідація введених даних. Форма вважатиметься валідною за наступних умов: всі поля заповнені; є хоча б одна ігрова категорія, є хоча б одна штрафна категорія, у грі є хоча б одна роль.

– Якщо валідація пройшла успішно, користувач натискає кнопку додавання гри і гра додається до бази даних.

Третій бізнес процес - реєстрація облікового запису користувача.

Бізнес процес – реєстрація облікового запису користувача зображений на рисунку 2.3

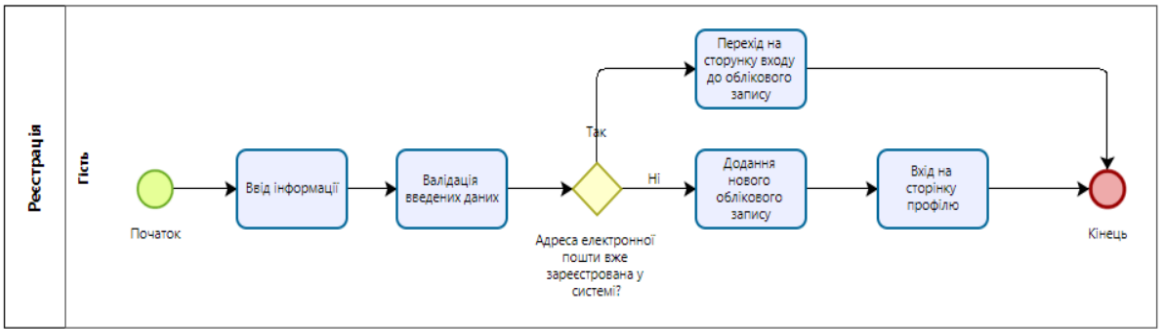


Рисунок 2.3 – Схема бізнес-процесу «Реєстрація облікового запису користувача»

Опис послідовності створення облікового запису користувача:

– Користувач переходить на сторінку реєстрації.

– Користувач заповнює поля реєстрації.

– Якщо введені поля, не відповідають шаблону заповнення на клієнтській стороні, відповідні поля підсвічуються помилкою.

– Якщо у системі вже існує акаунт з такою електронною адресою, користувач бачить повідомлення про те, що акаунт вже є зареєстрований і переходить на сторінку входу до облікового запису.

– Якщо електронна адреса є унікальною, користувач успішно додається до системи і автоматично авторизується.

2.2 Архітектура програмного забезпечення

Для даної системи я обрала клієнт-серверний стиль архітеутури. Дана система буде мати вигляд RESTful веб застосунку. В першу чергу це зумовлено такими вимогами як гнучкість та розширюваність. У такому вигляді допрацювання застоснку та додання нового функціоналу може проходити дуже легко. Ця архітектура є простою в опануванні і написанні додатку і водночас має дуже багато переваг.[8]

Для більш точного зображення архітектури застосунку на різних рівнях масштабування скористаємося моделлю C4. Вона передбачає будування діаграм для різних рівнів – діаграма контексту системи, діаграма контейнеру, діаграма окремих компонентів, тощо. Таким чином, ми почнемо з більш загального опису системи і поступово заглибимось у деталі реалізації.

Діаграма контексту системи, відома як System Context Diagram.

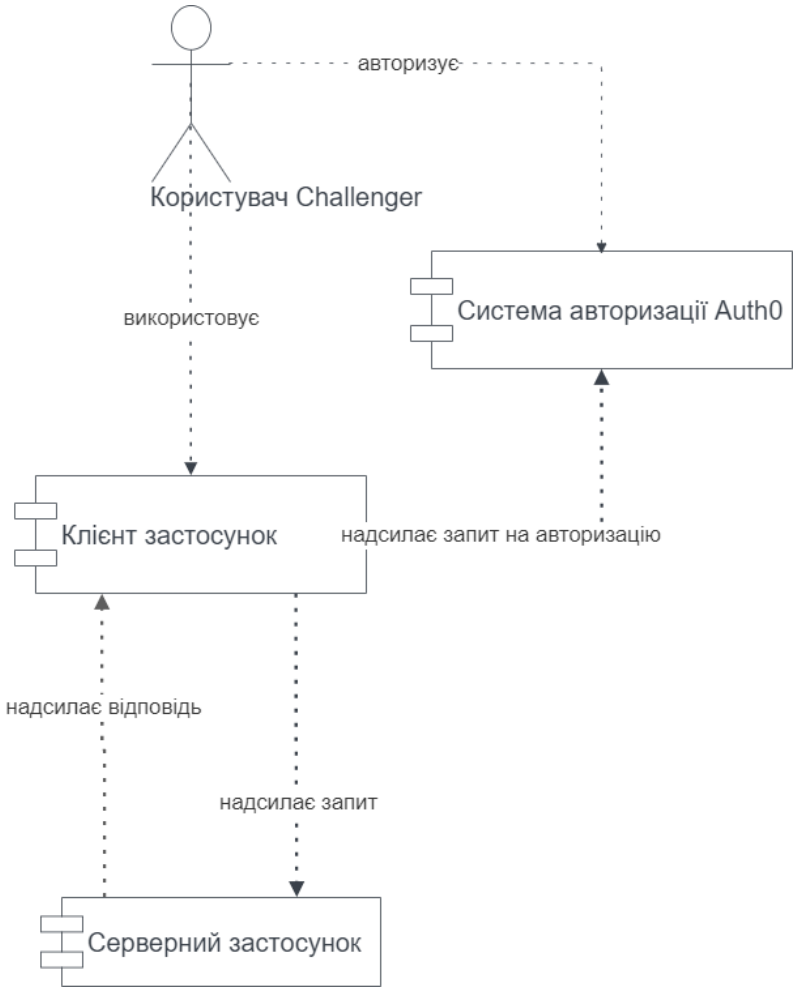


Рисунок 2.4 – Діаграма контексту системи

Розглянемо систему трохи більш детально за допомогою діаграми контейнерів, де контейнер — це окремо запущений/розгорнутий блок (наприклад, окремий простір процесу), який виконує код або зберігає дані. Вона зображена на рисунку 2.5.

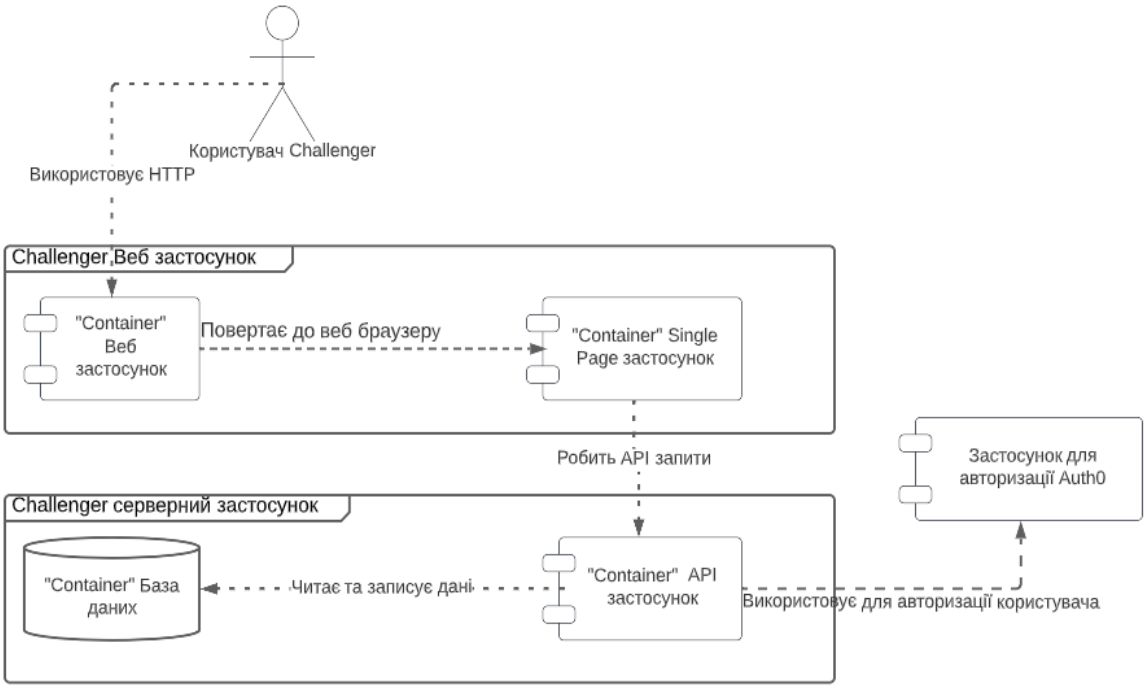


Рисунок 2.5 – Діаграма контейнерів

Діаграма компонентів показує, як контейнер складається з кількох «компонентів», їхні обов'язки та деталі реалізації. На рисунку 2.6 наведена діаграма компонентів. Як видно з даної діаграми, у API застосунку є декілька контролерів, кожен з яких відповідає за певну групу запитів (наприклад запити стосовно гри чи авторизації). Уся бізнес логіка та обробка даних відбувається у сервісах, кожен з яких відповідає за взаємодію з репозиторіями. Репозиторії є шаром доступу до сховища даних.

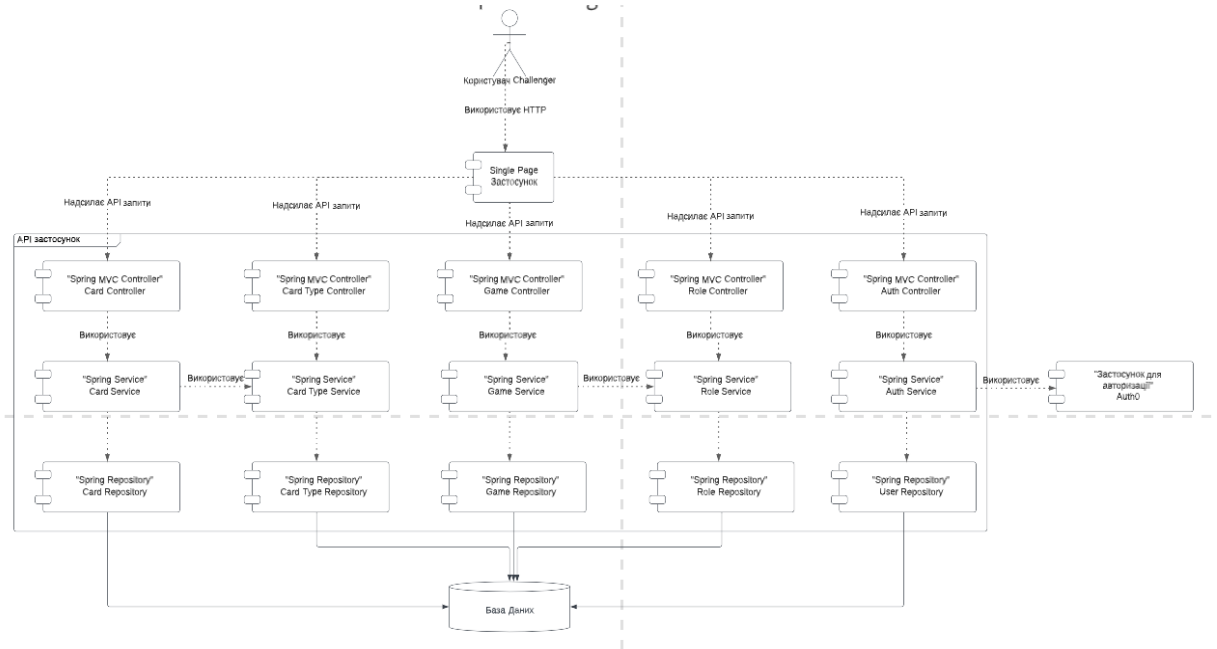


Рисунок 2.6 – Діаграма компонентів

На рисунку 2.7 наведена діаграма класів шару бізнес логіки та шару доступу до даних серверного застосунку.

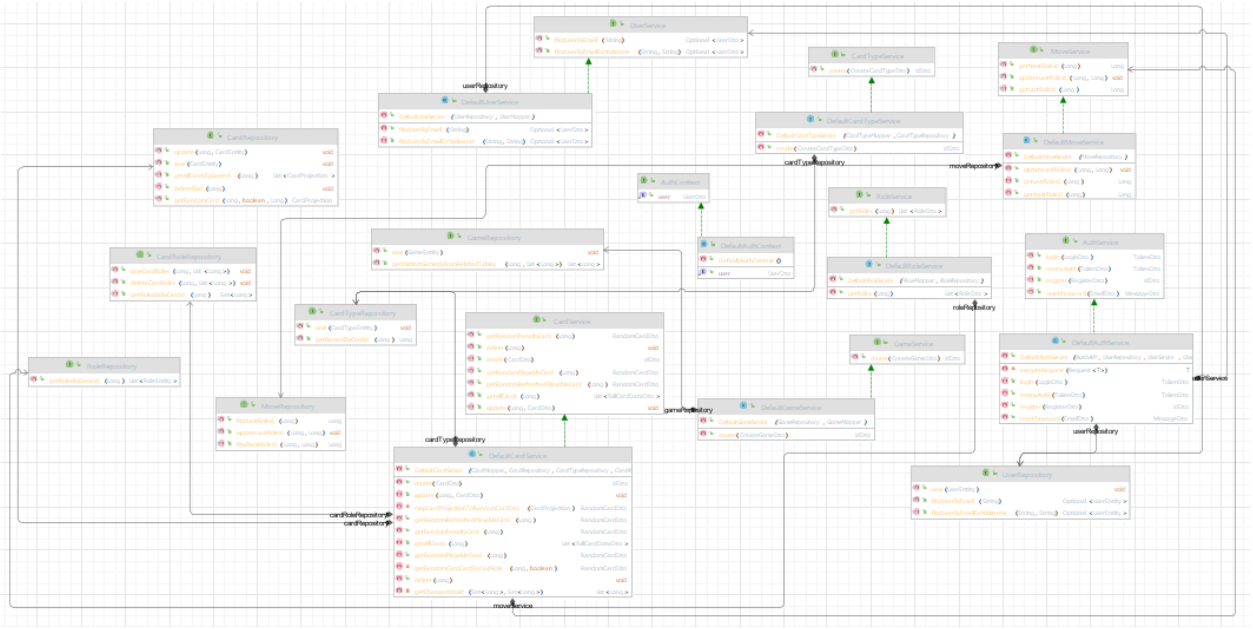


Рисунок 2.7 – Схема структурна класів

2.3 Конструювання програмного забезпечення

Опишемо детально роботу окремих модулів у таблиці 2.1.

Таблиця 2.1 - Модулі

Модуль	Призначення модуля
Single Page веб застосунок	Виконує роль користувацького інтерфейсу, формує та надсилає до API застосунку HTTP запити.
Card Controller	Приймає запити на створення, редагування, видалення карток.
Card Type Controller	Приймає запити на створення, редагування, видалення категорій карток.
Game Controller	Приймає запити на створення, редагування, видалення ігор. Приймає запити на контролювання проходження гри (наступна картка, картка з штрафом, оновити картку)
Role Controller	Приймає запити на створення, редагування, видалення ролей. Приймає запити на присвоєння ролей до карток.

Продовження таблиці 2.1

Auth Controller	Приймає запити на всі дії пов’язані із аутентифікацією та авторизацією користувача (логін, реєстрація, оновлення паролю, видалення акаунту).
Card Service	Містить методи для виконання створення, видалення та редагування картки у системі.
Card Type Service	Містить методи для виконання створення, видалення та редагування категорій карток у системі.
Game Service	Містить методи для виконання створення, видалення та редагування ігор у системі. Містить алгоритм для контролю ходу у грі.
Role Service	Містить методи для виконання створення, видалення та редагування ролей у системі.
Auth Service	Містить методи для дій пов’язані із аутентифікацією та авторизацією користувача. Ці методи формують і надсилають запити до API Auth0.
Застосунок Auth0	Є засобом аутентифікації користувача. Саму у цьому додатку зберігаються паролі користувачі, не у базі даних. Auth0 забезпечує надійну аутентифікацію за допомогою токенів, що гарантує безпеку особистих даних користувачів і застосунку.
Card Repository	Взаємодія з таблицею бази даних cards.
Card Type Repository	Взаємодія з таблицею бази даних card_types.
Game Repository	Взаємодія з таблицею бази даних games.
Role Repository	Взаємодія з таблицею бази даних roles.
User Repository	Взаємодія з таблицею бази даних users.
База даних	Реляційна база даних, призначена для структурованого збереження даних застосунку.

Наведемо діаграму сутностей застосунку. Основними сутностями є Користувач (User), Гра (Game) , Категорія карток (Card Type), Картка (Card), Роль (Role), Хід (Move). Кожна гра має користувача, який є її автором. Категорій карток належать до конкретної гри. Так само ролі належать до гри. Картки в свою чергу належать до певної категорії, тобто вони пов’язані з грою через категорії. Хід дозволяє контролювати черговість учасників у грі і містить записи про останній зроблений хід за допомогою ідентифікатора ролі.

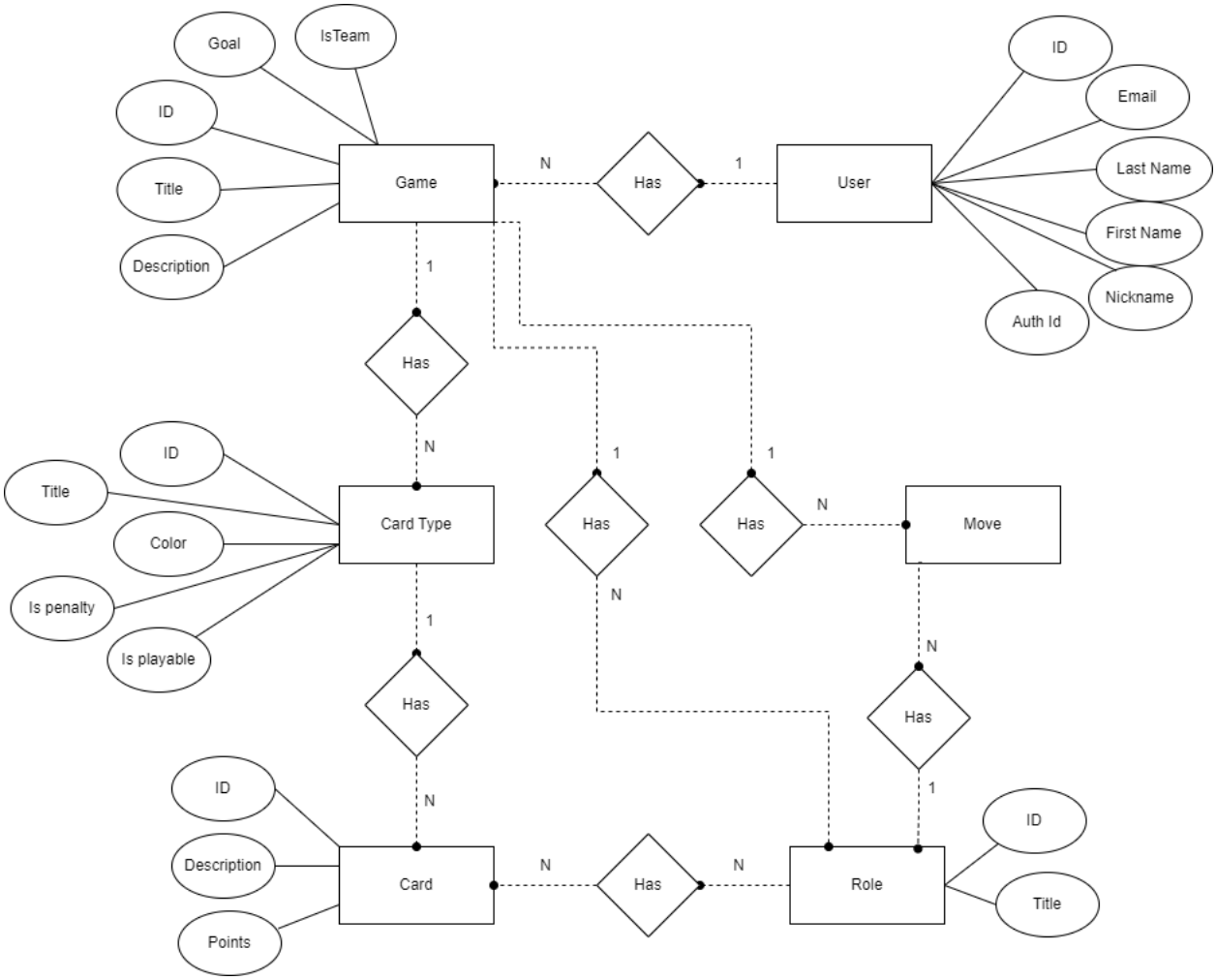


Рисунок 2.8 – ER діаграма сутностей User, Game, Card Type, Card, Role, Move

В якості системи управління базами даних використовується Postgres.[17] База даних серверу призначена для зберігання користувачів, а також даних про створені ігри. Опис таблиць бази даних наведено у таблицях 2.2 - 2.8. Модель бази даних наведена на рисунку 2.9.

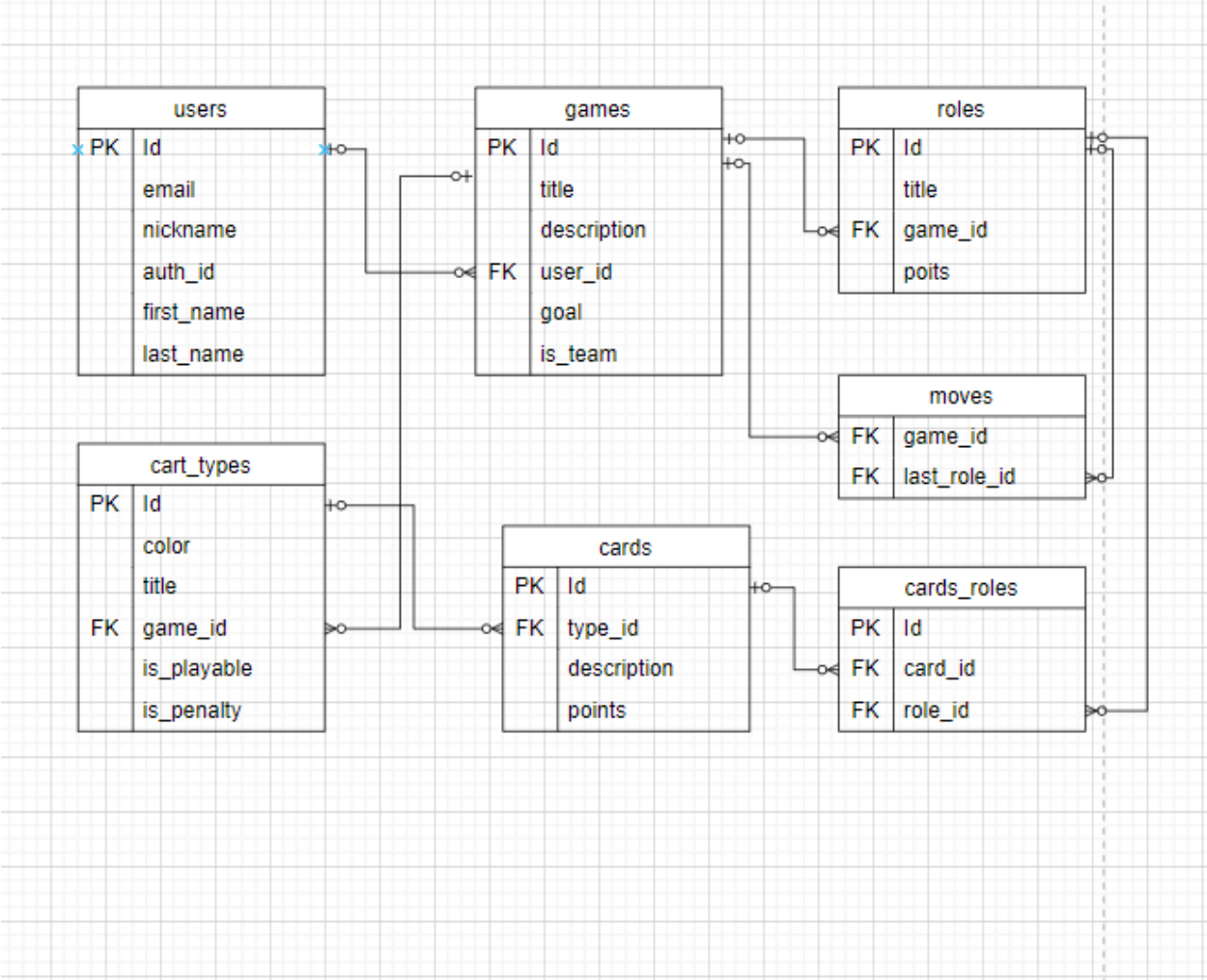


Рисунок 2.9 – Схема бази даних

Таблиця 2.2 - Опис таблиці users

Таблиця	Назва поля	Тип даних	Опис
users	id	bigint	Ідентифікаційний номер користувача
	email	varchar(50)	Електронна пошта користувача
	auth_id	varchar(32)	Ідентифікатор користувача у системі Auth0

Продовження таблиці 2.2

	nickname	varchar(100)	Нікнейм коритсувача
	first_name	varchar(50)	Ім’я користувача
	last_name	varchar(50)	Прізвище користувача

Таблиця 2.3 - Опис таблиці games

Таблиця	Назва поля	Тип даних	Опис
games	id	bigint	Ідентифікаційний номер гри
	user_id	bigint	Посилання на ідентифікатор у таблиці users на автора гри
	title	varchar(20)	Назва гри
	description	varchar(1000)	Короткий опис гри
	goal	integer	Мета гри, фініш.
	is_team	boolean	Прапор, який вказує на те, що гра знаходиться у командному режимі

Таблиця 2.4 - Опис таблиці card_types

Таблиця	Назва поля	Тип даних	Опис
card_types	id	bigint	Ідентифікаційний номер категорії

Продовження таблиці 2.4

	game_id	bigint	Посилання на ідентифікатор у таблиці games на гру, до якої належить категорія
	title	varchar(20)	Назва категорії
	color	varchar(15)	Колір, який може бути заданий у читаємому для html варіанті (‘red’або ‘#FF0000’). Дає можливість виділити різні категорії різними кольорами для зручності.
	Is_playable	boolean	Прапор, який вказує що категорія призначена для нештрафного ігрового процесу
	Is_penalty	boolean	Прапор, який вказує що категорія призначена для штрафу

Таблиця 2.5 - Опис таблиці cards

Таблиця	Назва поля	Тип даних	Опис
cards	id	bigint	Ідентифікаційний номер картки
	type_id	bigint	Посилання на ідентифікатор у таблиці card_types на категорію, до якої належить картка
	description	varchar(200)	Опис картки (завдання, умови, тощо)

Продовження таблиці 2.5

	points	integer	Кількість балів за успішне виконання завдання

Таблиця 2.6 - Опис таблиці roles

Таблиця	Назва поля	Тип даних	Опис
roles	id	bigint	Ідентифікаційний номер ролі
	game_id	bigint	Посилання на ідентифікатор у таблиці games на гру, до якої належить роль
	title	varchar(20)	Ім’я ролі

Опис таблиці cards_roles представлений у таблиці 2.7. Ця таблиця потрібна для забезпечення зв’язку типу «багато до багатьох» між таблицями cards та roles.

Таблиця 2.7 – Опис таблиці cards_roles

Таблиця	Назва поля	Тип даних	Опис
cards_roles	id	bigint	Ідентифікаційний номер
	card_id	bigint	Посилання на ідентифікатор у cards
	role_id	bigint	Посилання на ідентифікатор у таблиці roles

Опис таблиці moves представлений у таблиці 2.8. У цю таблицю записується останній хід (ідентифікатор ролі учасника, який виконав останній

хід) та ідентифікатор гри (для полегшення пошуку). Це поле перезаписується з кожним ходом, дозволяючи відслідковувати черговість учасників.

Таблиця 2.8 – Опис таблиці moves

Таблиця	Назва поля	Тип даних	Опис
moves	game_id	bigint	Посилання на ідентифікатор у таблиці games
	role_id	bigint	Посилання на ідентифікатор у таблиці roles останнього учасника ходу

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.22.

Таблиця 2.9 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	IntelliJ IDEA	Головне середовище розробки програмного забезпечення серверної частини курсової роботи.
2	Web Storm	Головне середовище розробки програмного забезпечення веб частини курсової роботи.
3	Postman	Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.

Продовження таблиці 2.9

4	PgAdmin 4	Програмне забезпечення яке надає легкий графічний інтерфейс для доступу до бази даних Postgres.
5	Auth0	Програмне забезпечення для забезпечення аутентифікації користувача.

2.4 Аналіз безпеки даних

В сучасному цифровому світі безпека даних є надзвичайно важливою. Застосунок треба проектувати враховуючи велику кількості кібератак та крадіжок конфіденційної інформації. У цьому розділі проведено аналіз безпеки даних та визначено заходи для забезпечення їх найвищого рівня захисту. Для забезпечення цієї безпеки потрібно дотримуватись рекомендацій Загального регламенту з охорони персональних даних.[16]

Згідно з правилами щодо обробки персональних даних, застосунок повинен забезпечити технічні та організаційні заходи для забезпечення безпеки персональних даних та запобігання несанкціонованому доступу, витоку або втраті цих даних. Інформація про користувачів, така як паролі, повинна бути збережена у безпечному, шифрованому вигляді.

Замість безпосереднього зберігання паролю користувачів у базі даних, для авторизації використовується сервіс Auth0, який допомагає забезпечити безпеку процесу входу в систему. Він використовує такі технології для забезпечення безпеки як шифрування даних, використання токенів для авторизації та аутентифікації, захист від атак переповнення буфера, тощо. Крім того, Auth0 надає інструменти для управління доступом та керування дозволами користувачів, що сприяє виконанню прав користувачів, як вимагає GDPR.

Запити користувачів захищені відповідно до мережевих сертифікатів та протоколів. Дані передаються через мережу в зашифрованому вигляді.

Всі поля вводу захищені від SQL ін’єкцій та не є загрозою для особистих даних користувачів.

Висновки до розділу

В даному розділі було описано процес моделювання та загальну архітектуру розробленого програмного забезпечення. Архітектура була детально описана за моделлю C4.

Були описані основні бізнес процеси та алгоритми.

Також було описано базу даних та детально проаналізовано всі її таблиці. Було проаналізовано безпеку збереження інформації та самого застосунку.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Протягом розробки програмного забезпечення різні дефекти та помилки виникають на кожній стадії від написання документації до підтримки. Вирішити цю проблему можна за допомогою вичерпного тестування програмного забезпечення. Цей етап є важливим у життєвому циклі розробки програмного забезпечення, оскільки він дає можливість контролювати якість готового продукту, продуктивність системи, відповідність готового продукту заявленим вимогам.

Основні причини виникнення дефектів:

- неправильно побудована архітектура ПЗ;
- помилки при реалізації ПЗ;
- неналежна експлуатація ПЗ;
- відсутність контролю якості, який дозволив би виявити ці дефекти.

При тестуванні треба ретельно перевірити наступні пункти:

- правильне зображення елементів інтерфейсу;
- правильне створення та зображення об’єктів після виконання користувачем певних дій;
- ефективність програми;
- провести інтеграційні тести для виявлення можливих проблем із сторонніми сервісами (Auth0).

Опишемо деякі метрики коду. Опишемо метрики LOC та NLOC, що означає Lines of Code та Non-commented Lines of Code. Таблиця з описом даних метрик для всіх типів файлів представлена на рисунку 3.1.

file type	LOC	NCLOC
.gitignore (GitIgnore)	34	34
Dockerfile	9	9
File type auto-detected by file cor	5	5
Files supported via TextMate bund	68	68
Groovy	51	51
Java	1 671	1 671
JSON	178	178
Properties	17	17
Shell script	207	207
SQL	146	140
Text	15	
XML	305	305
Total	**2 706**	**2 685**
Average	225,50	244,09

Рисунок 3.1 – Метрика коду LOC, NLOC

Для тестування буде використано стандарт ISO-9126, який визначає основні характеристики програмного забезпечення.[18] Характеристики, які визначає даний стандарт перераховані у списку:

– функціональність;

– надійність;

– зручність;

– ефективність;

– супровід;

– портативність.

3.2 Опис процесів тестування

Під час тестування програмного забезпечення будуть проводитись наступні функціональні типи тестів:

– тестування всіх функціональних вимог;

– тестування авторизації користувача за допомогою Auth0.

Також під час тестування проводитиметься перевірка нефункціональних

типи тестів:

– статичне тестування коду за допомогою статичного аналізатора;

– тестування роботи інтерфейсу користувача;

– навантажувальне тестування сервера по обробці даних.

Для статичного аналізу коду було обрано PVS Studio. Цей інструмент дозволяє перевірити код на можливі помилки та вразливості. Звіт PVS Studio представлений на рисунку 3.2.

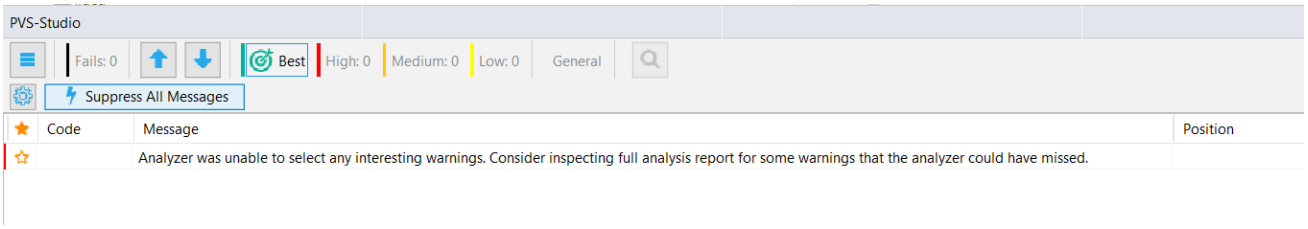


Рисунок 3.2 – Результат статичного аналізу коду

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.1 – 3.12.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Ім’я, прізвище, нікнейм, електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: ім’я та прізвище користувача, які мають містити літери і бути не коротше трьох символів, нікнейм (не коротше трьох символів), коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 8 символів, який містить хоча б з одну англійську літеру, одне число і один спеціальний символ. Після цього натискається кнопка підтвердження реєстрації.

Продовження таблиці 3.1

Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку свого акаунту. Користувач додається до Auth0.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку свого акаунту. Користувач додається до Auth0.

Таблиця 3.2 – Тест 1.2

Тест	Вхід до акаунту користувача з правильними даними
Модуль	Логін
Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці логіну
Вхідні данні	Електронна пошта, пароль (відповідають існуючому акаунту)
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка є зареєстрованою у системі, та пароль, який співпадає із паролем даного акаунту. Після цього натискається кнопка логіну.
Очікуваний результат	Оскільки пароль та пошта відповідають існуючому акаунту, то вхід проходить успішно, користувач перенаправляється на сторінку свого акаунту. Якщо пароль та пошта не відповідають існуючому акаунту, то виводиться повідомлення про помилку, користувач залишається на сторінці входу.

Продовження таблиці 3.2

Фактичний результат	Оскільки пароль та пошта відповідають існуючому акаунту, то вхід проходить успішно, користувач перенаправляється на сторінку свого акаунту. Якщо пароль та пошта не відповідають існуючому акаунту, то виводиться повідомлення про помилку, користувач залишається на сторінці входу.

Таблиця 3.3 – Тест 1.3

Тест	Вхід до акаунту користувача з неправильними даними
Модуль	Логін
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на сторінці логіну
Вхідні данні	Електронна пошта, пароль (не відповідають існуючому акаунту)
Опис проведення тесту	У відповідні поля вводяться: електронна пошта, яка є зареєстрованою у системі, та пароль, які є некоректними та не відповідають існуючому акаунту. Після цього натискається кнопка логіну.
Очікуваний результат	Виводиться повідомлення про помилку, користувач залишається на сторінці входу.
Фактичний результат	Виводиться повідомлення про помилку, користувач залишається на сторінці входу.

Таблиця 3.4 – Тест 1.4

Тест	Створення гри
Модуль	Гра
Номер тесту	1.4

Продовження таблиці 3.4

Початковий стан системи	Користувач увійшов у систему і знаходиться на сторінці створення гри.
Вхідні данні	Назва гри, опис гри, назва категорій гри, кольори категорій, функціонал категорій, опис карток, кількість балів, учасники.
Опис проведення тесту	У відповідні поля вводяться: невалідні назва гри та опис гри, користувач натискає кнопку «Далі». Неправильно заповнені поля підсвітяться червоним і користувач їх виправить (назва має бути не менше трьох симовлів). Користувач знов натискає «Далі». Користувач натискає на кнопку «Додати категорію» і вводить в поля назву, колір та призначення (ігрова, штрафна) категорії. Якщо не всі поля заповнені, то при натисканні кнопки «Далі» порожні поля будуть виділені червоним. Після заповнення всіх полів натискання кнопки «Далі» відкриє форму для додання карток. Після додання всіх необхідних категорій користувач натискає на кнопку «Додати картку» і заповнює відповідні поля: назва картки, опис, кількість балів (має бути числом, не обов'язково позитивним) та обирає у випадаючому списку категорію. . Якщо не всі поля заповнені, то при натисканні кнопки «Далі» порожні поля будуть виділені червоним. Після заповнення всіх полів натискання кнопки «Далі» відкриє форму для додання учасників. Користувач заповнює поле нікнейм, після чого відмітить галочками всі картки, які може виконувати даний гравець або натисне кнопку «Додати до всіх карток». Користувач натискає кнопку «Додати гру».

Продовження таблиці 3.4

Очікуваний результат	Створюється нова гра, разом з категоріями та картками. Гра з'являється у розділі «Мої ігри» на головній сторінці. Якщо запустити гру, то в ній будуть наявні всі дані занесені при її створенні.
Фактичний результат	Створюється нова гра, разом з категоріями та картками. Гра з'являється у розділі «Мої ігри» на головній сторінці. Якщо запустити гру, то в ній будуть наявні всі дані занесені при її створенні.

Таблиця 3.5 – Тест 1.5

Тест	Видалення прогресу гри
Модуль	Гра
Номер тесту	1.5
Початковий стан системи	Користувач увійшов у систему і на сторінці «Мої ігри».
Вхідні данні	-
Опис проведення тесту	Користувач натискає кнопку «Видалити прогрес гри» (кнопка із зображенням ластику). Після цього користувач переходить до цієї гри, щоб переконатись що її прогрес очищено.
Очікуваний результат	Дані гри редагуються до стану щойно створеної ігри. На сторінці гри прогрес всіх учасників (або команди) дорівнює нулю.
Фактичний результат	Дані гри редагуються до стану щойно створеної ігри. На сторінці гри прогрес всіх учасників (або команди) дорівнює нулю.

Таблиця 3.6 – Тест 1.6

Тест	Видалення гри

Продовження таблиці 3.6

Модуль	Гра
Номер тесту	1.6
Початковий стан системи	Користувач увійшов у систему і на сторінці «Мої ігри».
Вхідні данні	-
Опис проведення тесту	Користувач натискає кнопку «Видалити гру» (кнопка із зображенням сміттєвого баку). Користувач залишається на сторінці, щоб переконатись, що гра зникла зі списку.
Очікуваний результат	Гра видаляється і зникає з розділу «Мої ігри».
Фактичний результат	Гра видаляється і зникає з розділу «Мої ігри».

Таблиця 3.7 – Тест 1.7

Тест	Зміна режиму гри з змагального на командний
Модуль	Гра
Номер тесту	1.7
Початковий стан системи	Користувач увійшов у систему і знаходиться на сторінці «Мої ігри». Він обрав одну із ігор зі змагальним режимом (які позначені зеленим кольором).
Вхідні данні	Гра зі змагальним режимом

Продовження таблиці 3.7

Опис проведення тесту	Користувач переходить на сторінку гри, щоб переконатись що вона у змагальному режимі. Це можна ідентифікувати за наявністю на ігровому полі декількох ракет, а також декількох записів у таблиці із результатами. Шкала прогресу та ракета будуть просуватись індивідуально для кожного учасника. Користувач повертається назад до списку ігор. Користувач натискає кнопку «Перемкнути режим» (кнопка із зображенням перемикача). Користувач переходить на сторінку редагованої гри, щоб переконатись що сторінка змінила режим.
Очікуваний результат	Гра відкривається у командному режимі. Це можна ідентифікувати по ігровому полю у вигляді космосу, де буде всього одна ракета із підписом «Екіпаж» і таблиця із прогресом учасників показуватиме лише одну шкалу – прогресу, яка буде просуватись із відповіддю будь-якого учасника. За замовчуванням, перемикання прогресу не стирає прогрес гри, тому прогрес команди почнеться з суми прогресу кожного учасника. Для початку гри наново необхідно стерти прогрес.
Фактичний результат	Гра відкривається у командному режимі. Це можна ідентифікувати по ігровому полю у вигляді космосу, де буде всього одна ракета із підписом «Екіпаж» і таблиця із прогресом учасників показуватиме лише одну шкалу – прогресу, яка буде просуватись із відповіддю будь-якого учасника. За замовчуванням, перемикання прогресу не стирає прогрес гри, тому прогрес команди почнеться з суми прогресу кожного учасника. Для початку гри наново необхідно стерти прогрес.

Таблиця 3.8 – Тест 1.8

Тест	Зміна режиму гри з командного на змагальний
Модуль	Гра
Номер тесту	1.8
Початковий стан системи	Користувач увійшов у систему і знаходиться на сторінці «Мої ігри». Він обрав одну із ігор з командним режимом (які позначені жовтим кольором).
Вхідні данні	Гра з командним режимом
Опис проведення тесту	Користувач переходить на сторінку гри, щоб переконатись що вона у командному режимі. Це можна ідентифікувати по ігровому полю у вигляді космосу, де буде всього одна ракета із підписом «Екіпаж» і таблиця із прогресом учасників показуватиме лише одну шкалу – прогресу, яка буде просуватись із відповіддю будь-якого учасника. Користувач повертається назад до списку ігор. Користувач натискає кнопку «Перемкнути режим» (кнопка із зображенням перемикача). Користувач переходить на сторінку редагованої гри, щоб переконатись що сторінка змінила режим.
Очікуваний результат	Гра відкривається у змагальному режимі. Це можна ідентифікувати за наявністю на ігровому полі декількох ракет, а також декількох записів у таблиці із результатами. Шкала прогресу та ракета будуть просуватись індивідуально для кожного учасника. За замовчуванням, перемикання прогресу не стирає прогрес гри, тому прогрес учасників зберігається. Для початку гри наново необхідно стерти прогрес.

Продовження таблиці 3.8

Фактичний результат	Гра відкривається у змагальному режимі. Це можна ідентифікувати за наявністю на ігровому полі декількох ракет, а також декількох записів у таблиці із результатами. Шкала прогресу та ракета будуть просуватись індивідуально для кожного учасника. За замовчуванням, перемикання прогресу не стирає прогрес гри, тому прогрес учасників зберігається. Для початку гри наново необхідно стерти прогрес.

Таблиця 3.9 – Тест 1.9

Тест	Запуск гри у змагальному режимі
Модуль	Гра
Номер тесту	1.9
Початковий стан системи	Користувач запустив гру у змагальному режимі.
Вхідні данні	Мета гри – 100 балів, бали за першу картку – 5 балів, за другу – 10 балів, за штрафну – (-10) балів.
Опис проведення тесту	Користувач натискає кнопку «Правильна відповідь» для різних учасників. Користувач натискає кнопку «Оновити питання». Користувач натискає послідовно кнопки «Штраф» і «Правильна відповідь» на картці з від’ємною кількістю балів.

Продовження таблиці 3.9

Очікуваний результат	При натисканні кнопки «Правильна відповідь» для картки на 5 балів для учасника прогрес учасника у таблиці і ракета учасника просуваються вперед на 5% ((5/100 * 100) %). При натисканні тієї ж кнопки для другого учасника для картки на 10 балів – на 10 %. Натискання кнопки «Оновити питання» видає нову картку для того самого учасника, але ніяк не зачіпає прогрес. Клік на кнопку на картці зі штрафом рухає ракету і прогрес відповідного учасника на 10 % назад.
Фактичний результат	При натисканні кнопки «Правильна відповідь» для картки на 5 балів для учасника прогрес учасника у таблиці і ракета учасника просуваються вперед на 5% ((5/100 * 100) %). При натисканні тієї ж кнопки для другого учасника для картки на 10 балів – на 10 %. Натискання кнопки «Оновити питання» видає нову картку для того самого учасника, але ніяк не зачіпає прогрес. Клік на кнопку на картці зі штрафом рухає ракету і прогрес відповідного учасника на 10 % назад.

Таблиця 3.10 – Тест 1.10

Тест	Запуск гри у командному режимі
Модуль	Гра
Номер тесту	1.10
Початковий стан системи	Користувач запустив гру у командному режимі.
Вхідні данні	Мета гри – 100 балів, бали за першу картку – 5 балів, за штрафну – (-10) балів.

Продовження таблиці 3.10

Опис проведення тесту	Користувач натискає кнопку «Правильна відповідь» для різних учасників. Користувач натискає кнопку «Оновити питання». Користувач натискає послідовно кнопки «Штраф» і «Правильна відповідь» на картці з від'ємною кількістю балів.
Очікуваний результат	При натисканні кнопки «Правильна відповідь» для картки на 5 балів для прогрес команди і ракета просуваються вперед на 5%. Натискання кнопки «Оновити питання» видає нову картку для того самого учасника, але ніяк не зачіпає прогрес команди. Клік на кнопку на картці зі штрафом рухає ракету і прогрес всієї команди на 10 % назад.
Фактичний результат	При натисканні кнопки «Правильна відповідь» для картки на 5 балів для прогрес команди і ракета просуваються вперед на 5%. Натискання кнопки «Оновити питання» видає нову картку для того самого учасника, але ніяк не зачіпає прогрес команди. Клік на кнопку на картці зі штрафом рухає ракету і прогрес всієї команди на 10 % назад.

Таблиця 3.11 – Тест 1.11

Тест	Досягнення учасником нового рівня і перемога учасника
Модуль	Гра
Номер тесту	1.11
Початковий стан системи	Користувач запустив гру у змагальному режимі. Прогрес учасника – 0. Кількість зірок у таблиці для цього гравця – 1.

Продовження таблиці 3.11

Вхідні данні	Мета гри – 100 балів, кількість балів для досягнення першого рівня – 0, для другого – 25% від мети, для третього – 50% від мети, для третього – 75%, для четвертого – 90%.
Опис проведення тесту	Користувач за допомогою правильних відповідей доводить кількість балів учасника до 25% від мети (25 балів). Потім до 50, 75 та 90 відповідно. Потім користувач доводить учасника до мети гри.
Очікуваний результат	На початку гри у таблиці учасників рівень гравця позначений 1 зіркою. На кожному з цих моментів (25, 50, 75, 90) додається по одній зірці. В кінці гри у учасника має бути 5 зірок. При доходженні до мети з'являється вікно із привітанням і варіантами : побачити результати, завантажити результати, продовжити гру, завершити гру.
Фактичний результат	На початку гри у таблиці учасників рівень гравця позначений 1 зіркою. На кожному з цих моментів (25, 50, 75, 90) додається по одній зірці. В кінці гри у учасника має бути 5 зірок. При доходженні до мети з'являється вікно із привітанням і варіантами : побачити результати, завантажити результати, продовжити гру, завершити гру.

Таблиця 3.12 – Тест 1.12

Тест	Завантаження результатів
Модуль	Гра
Номер тесту	1.12
Початковий стан системи	Користувач дійшов до мети гри і натиснув кнопку «Завантажити результати».
Вхідні данні	Ходи всіх учасників

Продовження таблиці 3.12

Опис проведення тесту	Користувач натискає на кнопку. Користувач переходить у папку «Downloads» на своєму пристрої і переглядає файл з назвою game_*.txt.
Очікуваний результат	Користувач бачить у файлі таблицю з усіма переможцями гри (якщо у змагальному режимі) у форматі: Під ним він бачить перелік усіх ходів у форматі: 1 – ім’я першого переможця 2 - ім’я другого переможця 3 - ім’я третього переможця Завдання: текст завдання картки Роль: ім’я учасника Статус: Виконано / Оновлено / Не виконано Бали: кількість балів за завдання Бали всього: сумарні бали учасника після виконання завдання
Фактичний результат	Користувач бачить у файлі таблицю з усіма переможцями гри (якщо у змагальному режимі) у форматі: Під ним він бачить перелік усіх ходів у форматі: 1 – ім’я першого переможця 2 - ім’я другого переможця 3 - ім’я третього переможця Завдання: текст завдання картки Роль: ім’я учасника Статус: Виконано / Оновлено / Не виконано Бали: кількість балів за завдання Бали всього: сумарні бали учасника після виконання завдання

3.3 Опис контрольного прикладу

Відкривши застосунок користувач знаходиться на домашній сторінці застосунку.

Для роботи з системою користувач має увійти до неї. Для цього він натискає на кнопку «Вхід» на меню зверху.

Якщо користувач вже має акаунт він має увійти в систему за допомогою логіна та паролю. Сторінка входу зображена на рисунку 3.1. Якщо користувач не має акаунту він має натиснути на кнопку «Зареєструватись» і заповнити реєстраційну форму відповідними валідними даними (ім’я, прізвище та нікнейм більше 2 символів, валідна пошта, надійний пароль). Сторінка реєстрації зображена на рисунку 3.2.

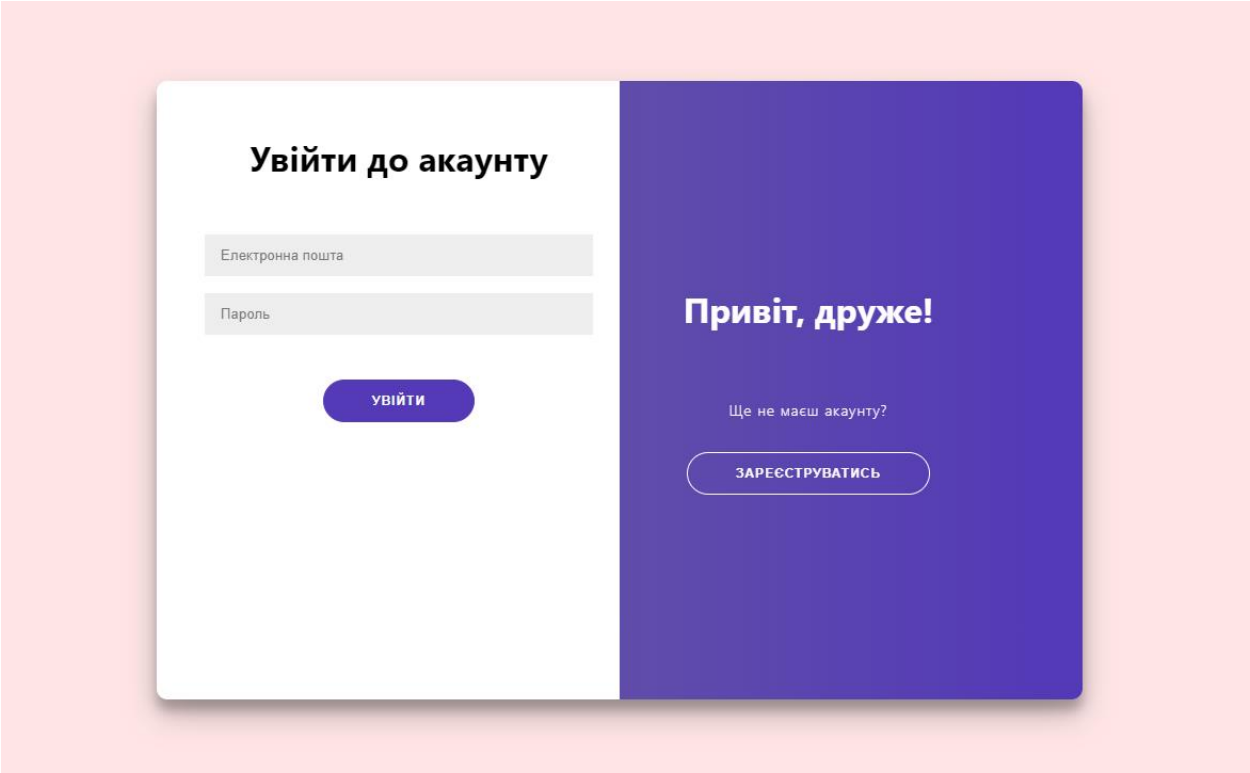


Рисунок 3.1 – Логін

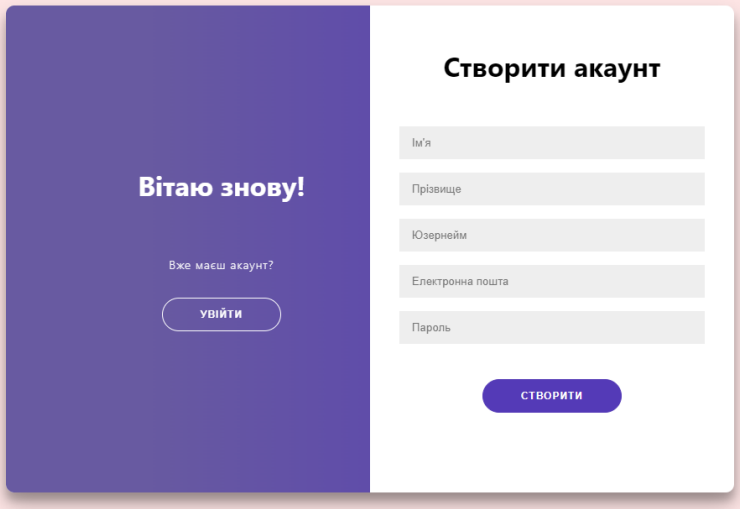


Рисунок 3.2 – Реєстрація

Після успішного входу або реєстрації користувач опиняється на сторінці свого акаунту – «Мої ігри». На ній користувач бачить всі додані ним ігри. Кожна гра відображає назву, опис, режим гри (зелені – змагальні, жовті – командні). Для кожної гри є кнопки – видалення гри, видалення прогресу гри, зміна режиму гри. Також тут наявна кнопка додавання нової гри.

З цієї сторінки у користувача є доступ до більшості функцій застосунку. Перевіримо як працює запуск гри у змагальному режимі. Користувач створює нову гру, запускаючи це процес кнопкою «+». Всі етапи створення гри можна побачити на рисунках 3.3-3.6

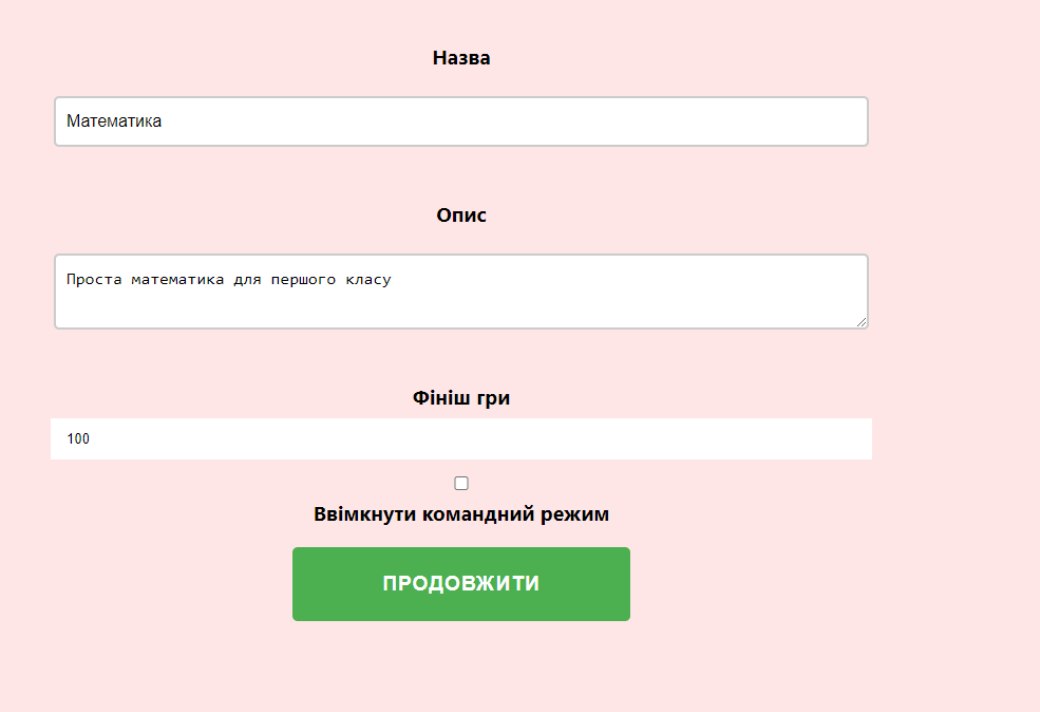


Рисунок 3.3 – Додавання базової інформації про гру

Створення категорій. Штрафні категорії додаються автоматично, але можуть бути редаговані.

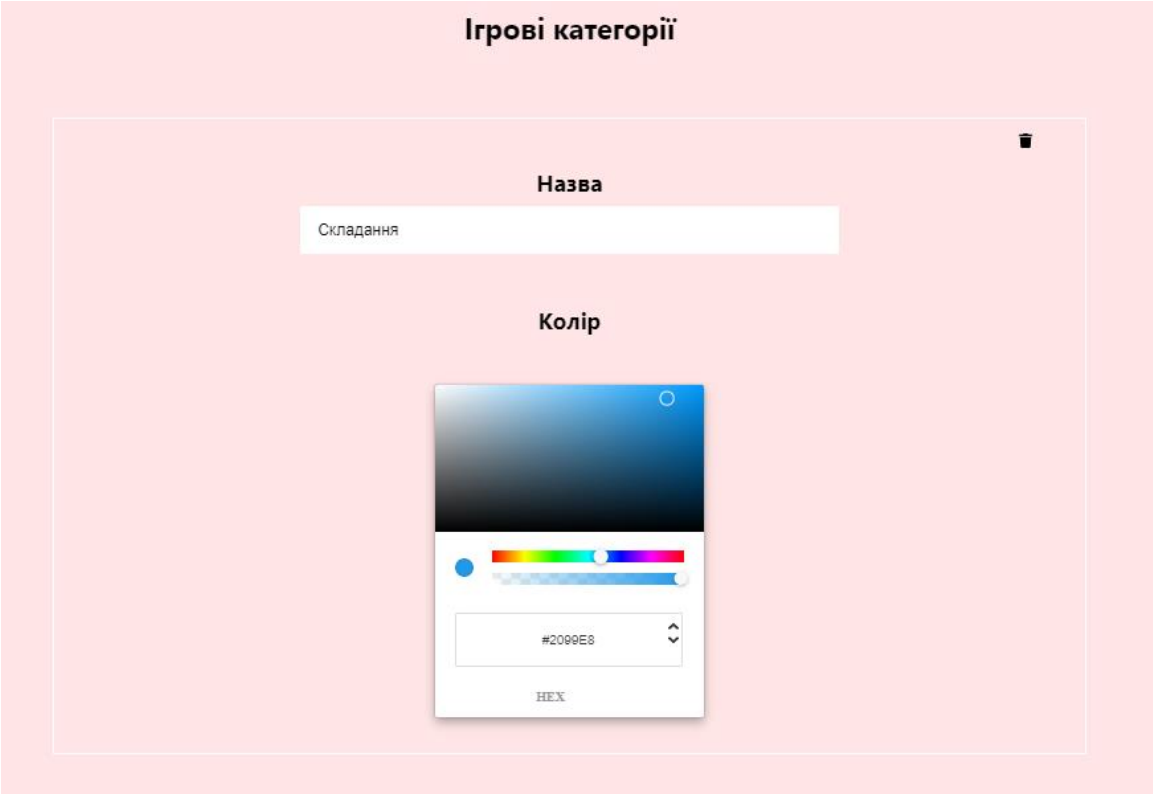


Рисунок 3.4 – Додавання категорії

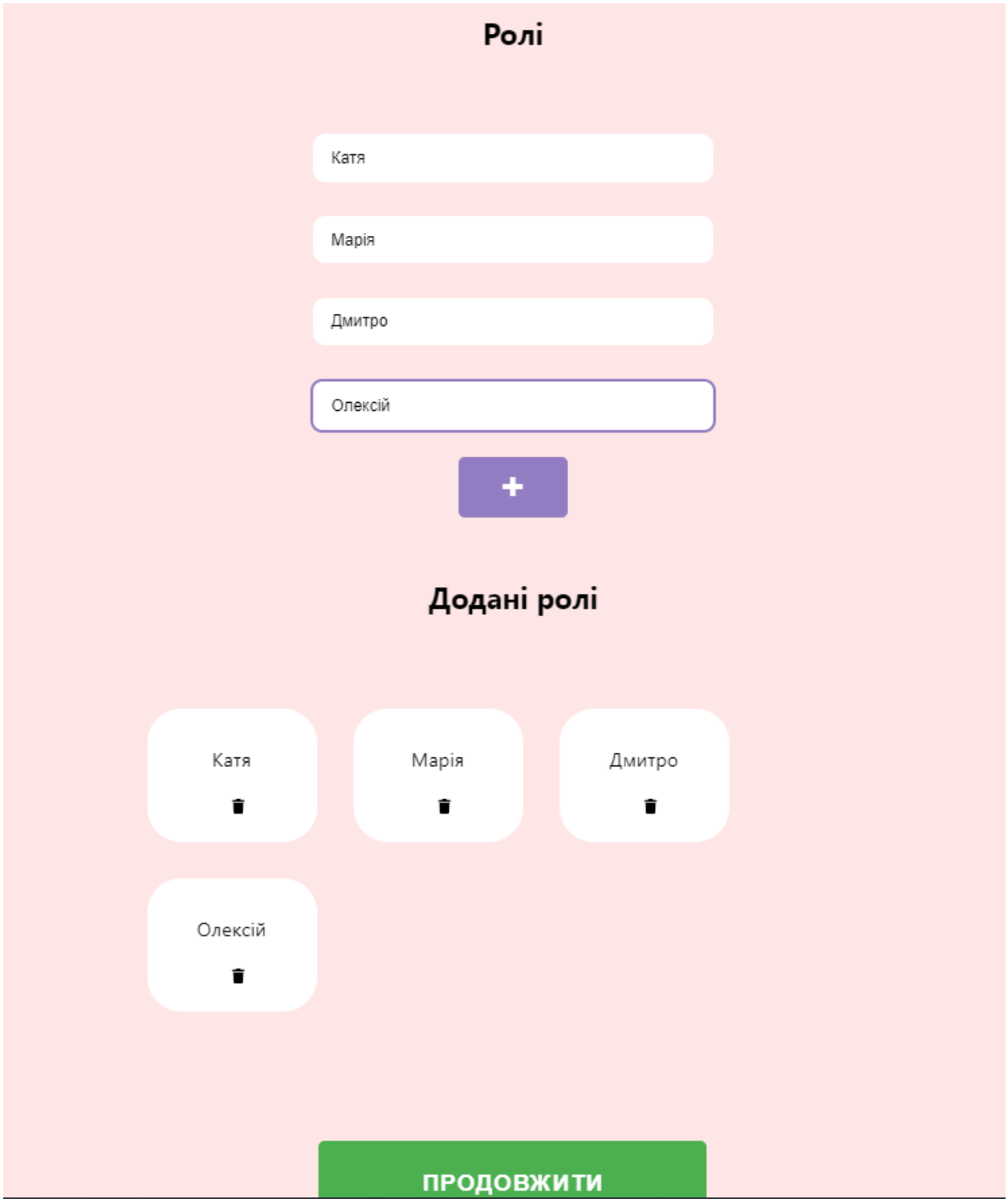


Рисунок 3.5 – Додавання ролей

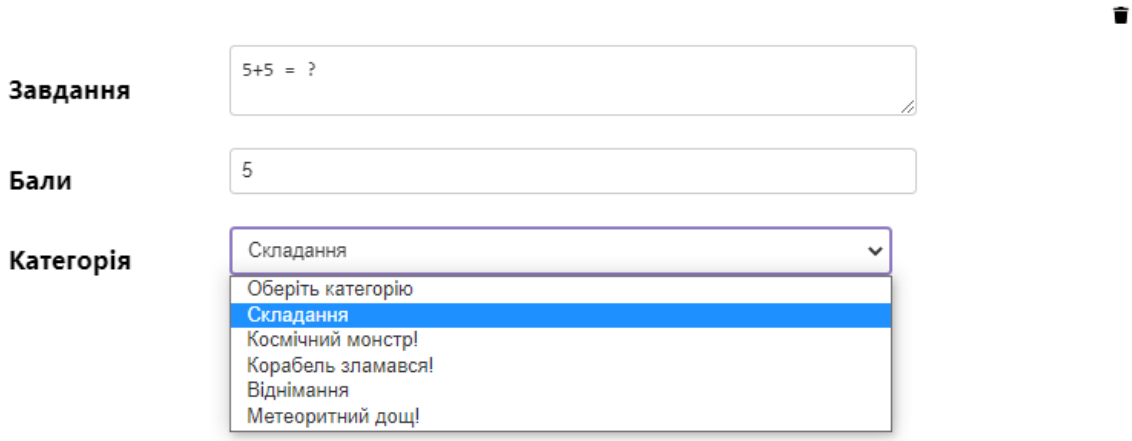


Рисунок 3.6 – Створення картки

Після створення гри користувач бачить повідомлення про успішне виконання.

Після створення гри вона з’являється у списку ігор. Перейдемо до гри математика у змагальному режимі. Початковий стан, який має мати щойно запущена гра, зображено на рисунках 3.7 та 3.8. Як видно, всі символи ракет мають бути на старті ігрового поля, рівні учнів – 1 зірка, планета Земля.

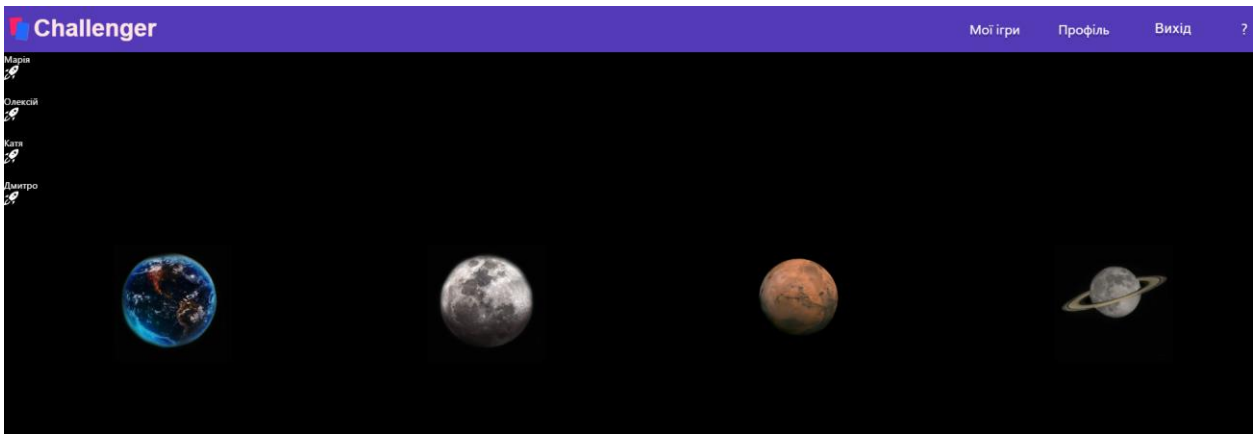


Рисунок 3.7 – Гра у змагальному режимі ч.1

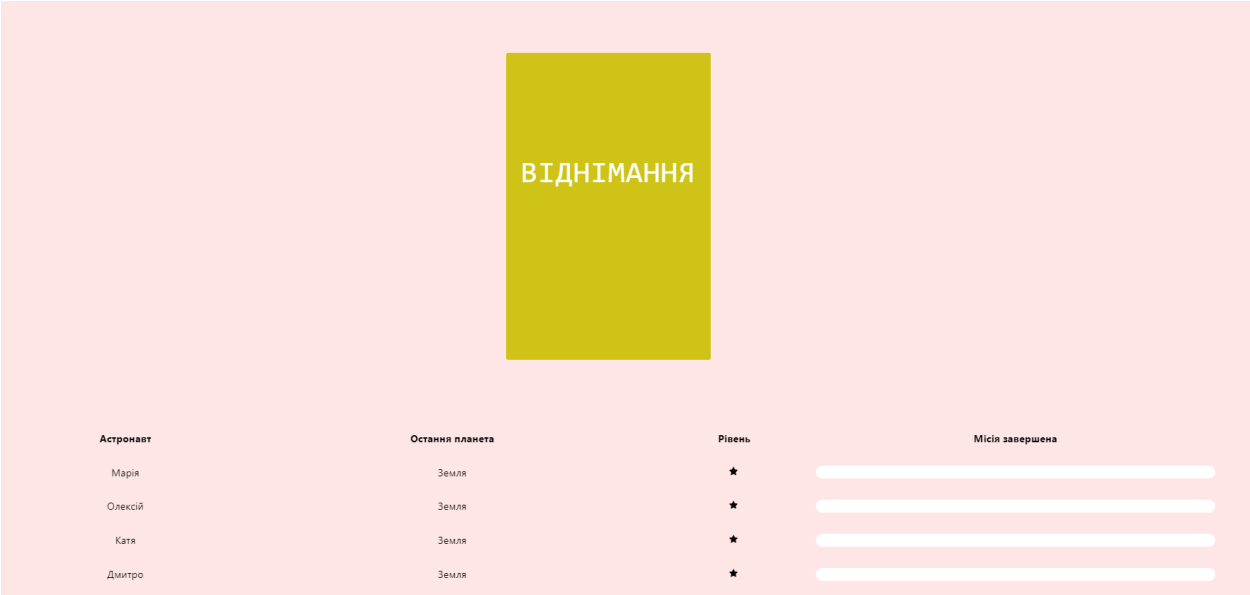


Рисунок 3.8 – Гра у змагальному режимі ч.2

При наведенні на картку користувач має побачити завдання та кількість балів за картку. Це зображено на рисунку 3.9. На картці видно три кнопки – зелена («Правильна відповідь»), блакитна («Оновити») та червона («штраф»).

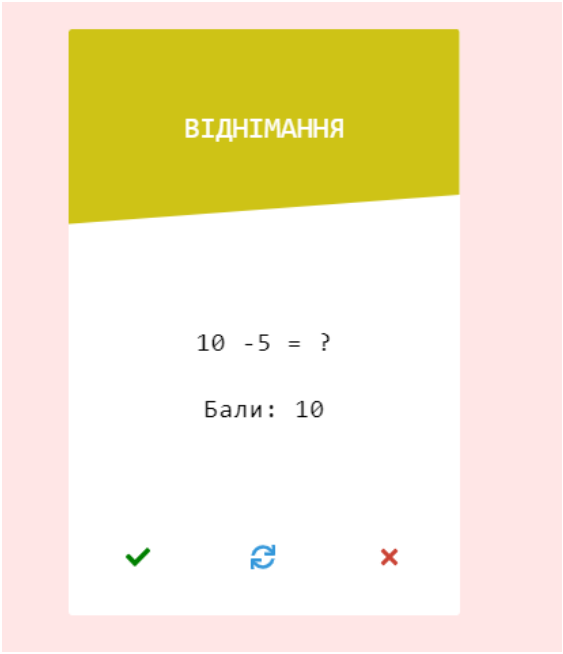


Рисунок 3.9 – Картка

Користувач натискає кнопки у такому порядку (ролі вказані у скобках) : зелена (1), зелена(2), зелена(3), зелена(4), зелена(1), червона(2), блакитна і

зелена(3) зелена(4), зелена(1), блакитна і червона(2), зелена(3), зелена(4), зелена(1). Таким чином, оскільки зелені кнопки рухають гравців вперед, сині нічого не змінюють, а червоні штовхають назад користувач бачить наступну картину, яка відповідає прогресу, обрахованому за допомогою балів за картку та мети гри. Змінений стан сторінки відображений на рисунках 3.10 та 3.11.

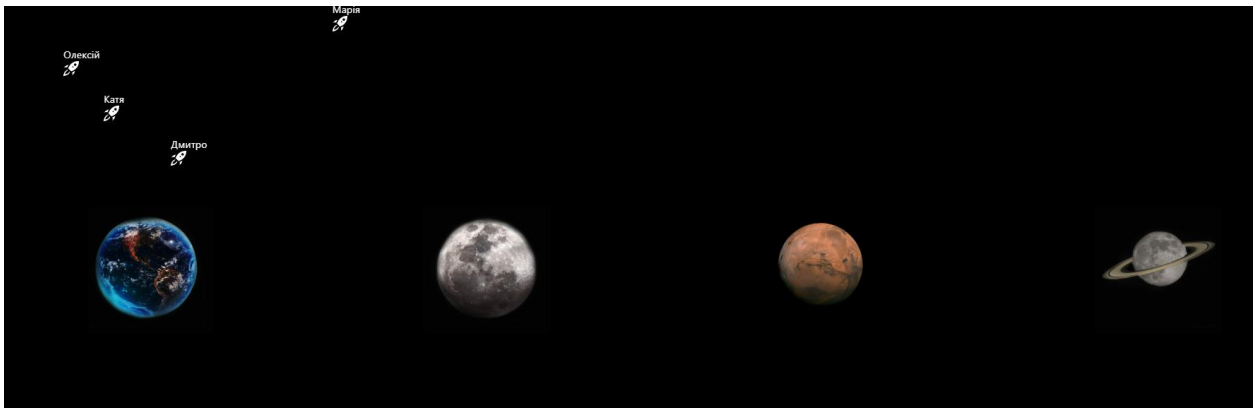


Рисунок 3.10 – Гра у змагальному режимі ч.1

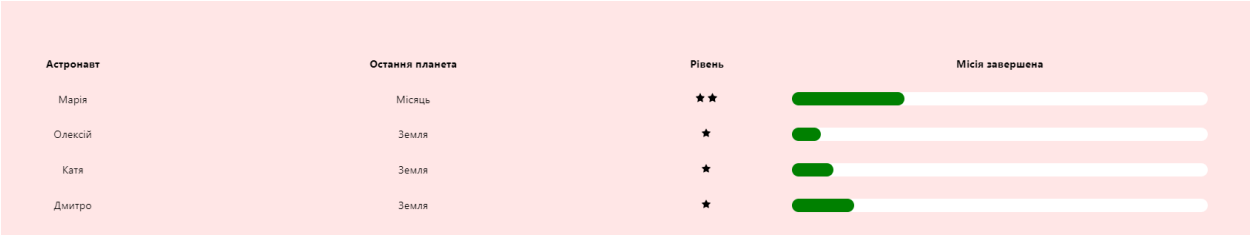


Рисунок 3.11 – Гра у змагальному режимі ч.2

Користувач завершує гру доведенням гравця Олексій до фініша і бачить наступне вікно.

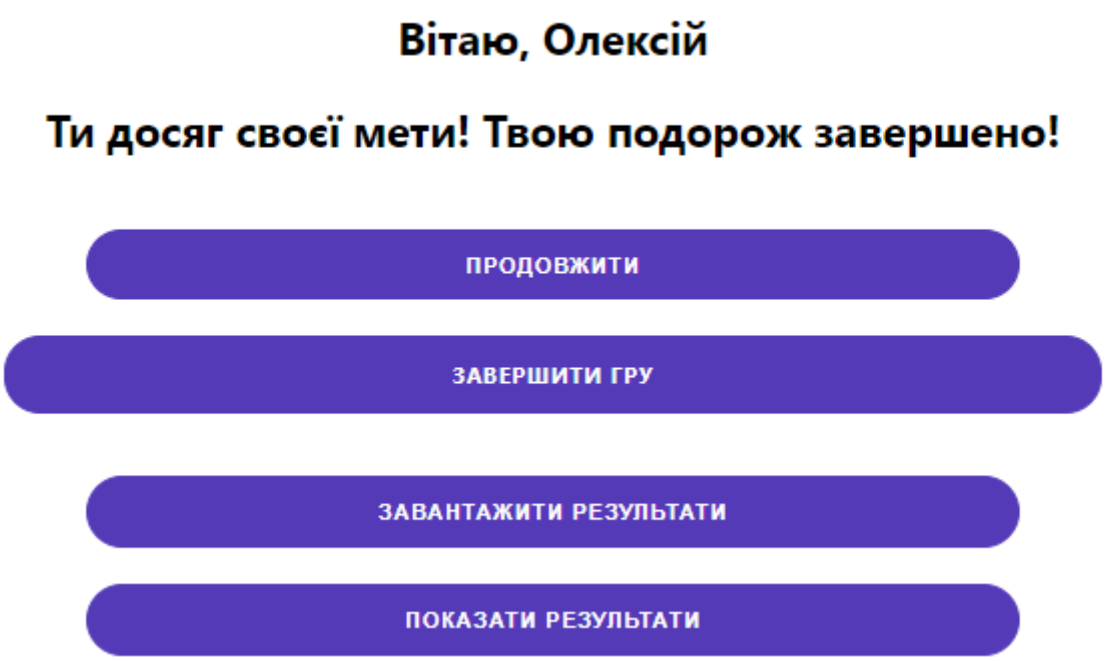


Рисунок 3.12 – Екран привітання із перемогою

Користувач натискає на «Показати результати» і вікно розширюється інформацією на рисунку 3.13.

Переможці

Місце	Гравець
1	Олексій

Ходи

№	Завдання	Роль	Статус	Результат	Бали
1	10 -5 = ?	Марія	Виконано	10	10
2	10 -5 = ?	Олексій	Виконано	10	10
3	7+3 = ?	Дмитро	Виконано	5	5
4	10 -5 = ?	Катя	Виконано	10	10
5	10 -5 = ?	Марія	Замінено		
6	7+3 = ?	Марія	Не виконано		
7	Ти потрапив у метеоретний дощ. Втрачаєш бали	Марія	Виконано	-3	7

Рисунок 3.13 – Таблиця результатів

Користувач натискає на «Завантажити результати» та відкриває файл у папці Загрузки. Частковий вміст файлу:

Переможці

---

1 - Олексій

Ходи

Завдання: 10 -5 = ?
Роль: Марія
Статус: Виконано
Бали: 10
Бали всього: 10

Користувач повертається на сторінку з іграми, перемикає режим гри «Математика» на командний та стирає прогрес гри. Цей режим відрізняється спрямованістю на командну роботу, а не змагання, тому всі бали учасників складаються разом та на ігровому полі рухається лише одна ракета.

Висновки до розділу

В цьому розділі було проаналізовано порядок тестування та складено план перевірки якості програмного забезпечення. Згідно з цим планом було протестовано якість розробки веб застосунку. При цьому було описано цілі, сценарії та механізми тестування, що використовувалися для цього. Були описані контрольні приклади тестування та процедура проведення тестів, а також проаналізована коректність роботи застосунку при різних тест кейсах.

Тестування показало, що програмне забезпечення відповідає всім функціональним та нефункціональним вимогам, які були поставлені до нього у попередніх розділах.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

У даному розділі описано процес розгортання програмного забезпечення. Для цього було проаналізовано різні сервіси для хостингу застосунків (Hiroku, Amazon Web Services, GitHub Pages). Після ознайомлення з можливостями різних хостингів та можливостями, які дає безкоштовна версія даних ресурсів було обрано використати Amazon Web Services (AWS).[19]

Наступним кроком, було сплановано всі кроки необхідні для хостингу застосунку. На рисунку 4.1 зображено діаграму розгортання програмного забезпечення.

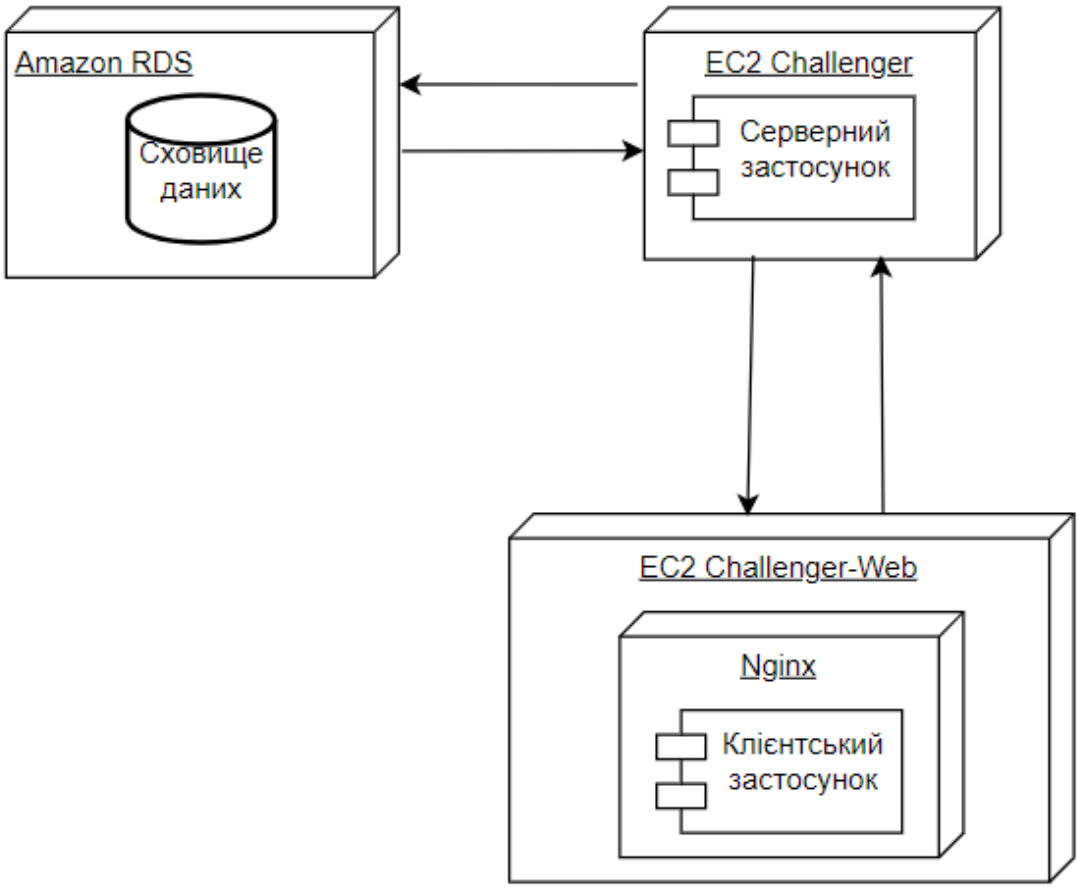


Рисунок 4.1 – Діаграма розгортання програмного забезпечення

4.1.1 Створення акаунту та налаштування хмарних ресурсів

Перш за все, для подальшої роботи було створено акаунт AWS та було створено і налаштовано приватну хмарну платформу (VPC) для забезпечення ізоляції обчислювальних ресурсів.

4.1.2 Хостинг бази даних

Для того щоб створити базу даних, я скористалась сервісом RDS. Створивши і налаштувавши базу даних, я потребувала переносу всіх даних та таблиць моєї існуючої бази даних до новоствореної. Я зробила копію локальної бази даних (backup), який використала, під ’єднавшись до бази даних на віртуальній машині.

Створену мною базу даних у системі моніторингу RDS можна побачити на рисунку 4.2.

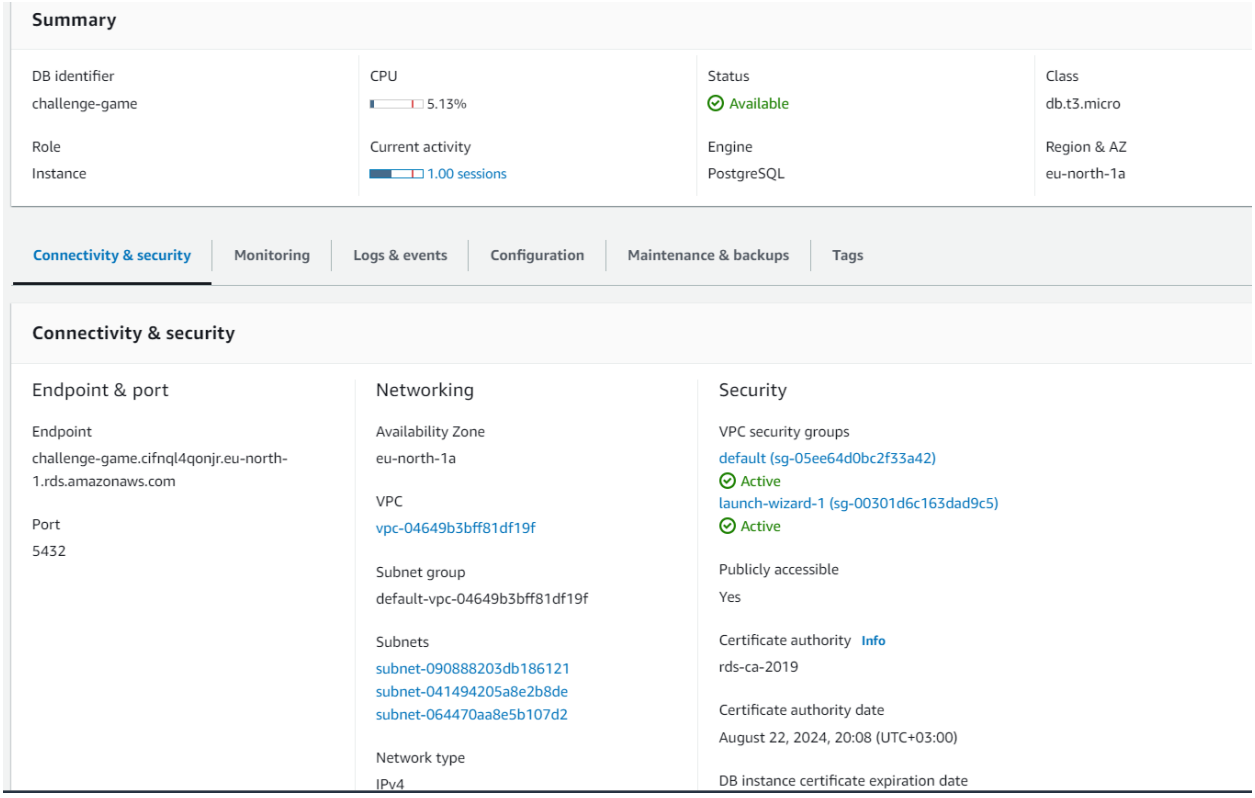


Рисунок 4.2 – База даних для застосунку Challenger

Після етапу створення бази даних я змінила рядок, який відповідає за доступ до сховища даних, у своєму бекенд застосунку.

4.1.3 Хостинг серверної частини застосунку.

Розгортання Java застосунку потребує декілька кроків. Перш за все, була створена віртуальна машина (EC2) на AWS. EC2 потребує певних налаштувань, для коректного зв'язку всіх сервісів між собою було налаштовано правила безпеки, які дозволяють підключення до бази даних та доступ до новоствореної машини. На рисунку 4.3 представлені правила безпеки, які були додані.

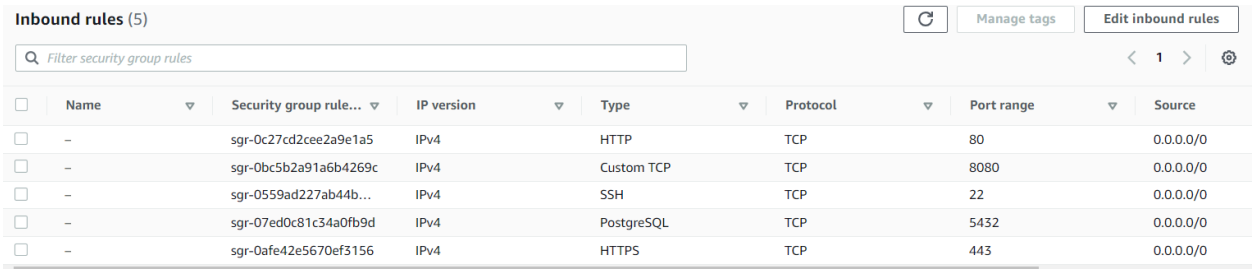


Рисунок 4.3 – Налаштування правил безпеки

Після завершення всіх налаштувань, було встановлене з'єднання із машиною. На цій машині було встановлено JVM та налаштовано змінні середовища, необхідні для роботи програми.
Для запуску програмного забезпечення було проведено декілька кроків.

- Клонування вихідного коду програми із github репозиторію.
- Зборка застосунку до jar файлу.
- Запуск jar файлу.
- Тестування роботи програмного забезпечння.

Після цього у клієнтському застосунку було змінено всі HTTP запити, щоб вони звертались до IP адреси розгорнутого застосунку, а не до локального порту.

4.1.4 Хостинг клієнтської частини застосунку.

Була створена нова віртуальна машина (EC2) на AWS у тому самому ізольованому хмарному просторі, як і попередня. Були проведені всі налаштування, у тому числі налаштування безпеки. Для роботи програми сервер потребує певного програмного забезпечення, а саме NodeJS та Nginx,

який запускатиме застосунок. Для запуску програмного забезпечення було проведено декілька кроків.

- Клонування вихідного коду програми із GitHub репозиторію.
- Зборка застосунку.
- Зміна налаштувань Nginx, встановлення шляху до головної сторінки «challenger/build/index.html»
- Тестування роботи програмного забезпечння.

Після проведення тестування за допомогою контрольного прикладу, описаного у розділі три, помилок не було виявлено. Застосунок працює разом з усіма HTTP запитами коректно і є доступний за публічною IP адресою.[19]

4.2 Підтримка програмного забезпечення

Після успішного розгортання програмного забезпечення, важливо забезпечити його належне управління та підтримку.

У разі виявлення проблем із програмним забезпечнням потрібно якнайшвидше виправити баг.

У разі необхідності оновлення програмного забезпечення алгоритм наступний:

- внесення змін до коду;
- тестування застосунку локально;
- розміщення оновленого коду у GitHub репозиторії;
- зборка застосунку на сервері;
- запуск jar файлу (для серверного застосунку) або перезапуск Nginx (для клієнтського застосунку).

Висновки до розділу

У розділі було описано розгортання системи з використанням Amazon Web Services (AWS) як хостингової платформи. Було встановлено, що AWS найкраще відповідає потребам проекту з урахуванням його можливостей та безкоштовних ресурсів. Також були розписані необхідні вимоги.

Процес розгортання включав такі етапи:

− Створення акаунту AWS і налаштування приватної хмарної платформи (VPC) для забезпечення ізоляції ресурсів.

− Створення бази даних за допомогою сервісу RDS і перенесення даних з існуючої бази даних.

− Розгортання серверної частини застосунку на віртуальній машині (EC2) з встановленням JVM та налаштуванням змінних середовища.

− Розгортання клієнтської частини застосунку на окремій EC2-машині з встановленням необхідного програмного забезпечення, такого як Node.js та Nginx.

В процесі розгортання було виконано кілька кроків, таких як клонування вихідного коду, збірка програмного забезпечення та перевірка його працездатності.

Важливо підкреслити, що після успішного розгортання необхідно забезпечити належну підтримку та управління програмним забезпеченням, включаючи виправлення помилок та оновлення системи за необхідності.

Результатом розгортання є доступний та працездатний застосунок, який може бути використаний користувачами через публічну IP-адресу.

ВИСНОВКИ

В даному дипломному проєкті було розроблено веб застосунок для впровадження інтерактивних ігрових елементів навчання у школах. Цей проєкт може стати важливим інструментом для підвищення зацікавленості дітей у навчанні.

Перед розробкою програмного забезпечення було проведено велику роботу з аналізу існуючих рішень для інтерактивного навчання, їх переваги та недоліки. В результаті порівняння різних аналогів було сформовано варіанти використання, функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення.

Враховуючи всі вимоги та сценарії використання, було спроектовано архітектуру застосунку, змодельовано бізнес-процеси, написано головний алгоритм застосунку – ігрову механіку. Було розроблено оптимальну схему сховища даних. У якості БД використано СУБД Postgres.

При розробці системи було підготовлено різноманітні діаграми за моделлю C4 для демонстрації взаємодії компонентів між собою.

Після реалізації застосунок був протестований різними методами, було ретельно проаналізовано коректність його роботи, відображення графічного інтерфейсу на різних пристроях. Всі заплановані тести повернули позитивні результати. Це означає, що застосунок працює так, як було задумано і відповідає всім нормативам якості.

Застосунок було розгорнуто за допомогою інструментів Amazon Web Services.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Салберг П. Finnish Lessons: What Can the World Learn from Educational Change in Finland. Teachers College Press, 2014. 41с.

2) Закон України «Про освіту» від 18 грудня 2018 року №2657-VIII. URL: https://zakon.rada.gov.ua/laws/show/2145-19#Text (дата звернення: 10.02.2023).

3) Фуллан М.: Stratosphere: Integrating Technology, Pedagogy, and Change Knowledge. Торонто, Pearson, 2013. 70с.

4) Браун П., Родігер Г., Макденіел М. Make It Stick: The Science of Successful Learning. Гарвард: Harvard University Press, 2014. 41 с.

5) Робінсон К., Ароніка Л. Creative Schools: The Grassroots Revolution That's Transforming Education. Старий Сейбрук: Tantor Media Inc city, 2015. 170 с.

6) Харріс К., Хоевер М. Technology Integration for Meaningful Classroom Use: A Standards-Based Approach. Cengage Learning, 2013. 51 с.

7) Вінінжер С.Р., Редіреф Д., Дерріберрі П. The Effects of Mobile Phones on Student Achievement and Student-Teacher Interaction in the Classroom // Computers in Human Behavior. – 2017.

8) Мартін Р. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Sebastopol: O'Reilly Media, 2017. 451 с.

9) Блох Д. Effective Java. Бостон: Addison-Wesley, 2001. 62с.

10) Сімпсон К. You Don't Know JS. Себастопол: O'Reilly Media, 2020. 20с.

11) Стефанов С. React Up and Running: Building Web Applications Себастопол: O'Reilly Media, 2016. 121 с.

12) Воллс К. Spring in Action. Менінг: Видавництво Manning, 2019. 130 с.

13) Поллак М., Гірке О., Рісберг Т., Брісбін Д., Гангер М. Spring Data: Modern Data Access for Enterprise Java. Себастопол: O'Reilly Media, 2012. 50 с.

14) Сілвермен Р. Е. Git Pocket Guide. Себастопол: O'Reilly Media, 2013. 7 с.

15) Документація Auth0. URL: https://auth0.com/docs (дата звернення: 5.03.2023).

16) Загальний регламент з охорони персональних даних. URL: https://gdpr-info.eu/ (дата звернення: 02.03.2023).

17) Обі Р., Хсу Л. PostgreSQL: Up and Running. Себастопол: O'Reilly Media, 2012. 44 с.

18) Документація Sowtfare Engeneering – Product Quality – Part 1: Quality Model. URL: https://www.iso.org/standard/22749.html (дата звернення: 22.05.2023).

19) Документація AWS. URL: https://docs.aws.amazon.com/ (дата звернення: 31.05.2023).

ДОДАТОК А - ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015376378

Дата перевірки:
01.06.2023 19:53:52 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2023 19:55:05 EEST

ID користувача:
76913

Назва документа: **IT_91_Гайова_ПЗ**

Кількість сторінок: **76** Кількість слів: **11140** Кількість символів: **84030** Розмір файлу: **2.77 MB** ID файлу: **1015042084**

11.3%
Схожість

Найбільша схожість: **3.31%** з джерелом з Бібліотеки (ID файлу: **1014961450**)

5.13% Джерела з Інтернету	99	Сторінка 78
11.1% Джерела з Бібліотеки	181	Сторінка 80

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0.04%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

0% Вилучення з Інтернету	28	Сторінка 81
0.04% Вилученого тексту з Бібліотеки	38	Сторінка 81

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	79

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНИХ ІГРОВИХ ЕЛЕМЕНТІВ НАВЧАННЯ В УКРАЇНСЬКІЙ ШКОЛІ

Текст програми

КПІ.ІТ-9106.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов ІВАНОВА

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Альона ГАЙОВА

Київ – 2023

Файл CardController.java

```
@RestController
@RequestMapping(Endpoint.CARD)
@RequiredArgsConstructor
public class CardController {
  private final CardService cardService;
   @PostMapping
   public ResponseEntity<IdDto> create(
       @PathVariable("gameId") long gameId,
       @Valid @RequestBody CardDto cardDto
   ) {
     return ResponseEntity
         .status(HttpStatus.CREATED)
         .body(cardService.create(cardDto));
   }

 @PutMapping("/{id}")
 public void update(
     @PathVariable("gameId") long gameId,
     @PathVariable("id") long id,
     @Valid @RequestBody CardDto cardDto
 ) {
   cardService.update(id, cardDto);
  }

 @DeleteMapping("/{id}")
 public void delete(
     @PathVariable("gameId") long gameId,
     @PathVariable("id") long id
 ) {
   cardService.delete(id);
 }

 @GetMapping
 public List<FullCardDataDto> getAllCards(
     @PathVariable("gameId") long gameId
 ) {
   return cardService.getAllCards(gameId);
 }

 @GetMapping("/next")
 public RandomCardDto getNextPlayableCard(
     @PathVariable("gameId") long gameId
 ) {
   return cardService.getRandomPlayableCard(gameId);
 }

 @GetMapping("/refresh")
 public RandomCardDto getRefreshedPlayableCard(
     @PathVariable("gameId") long gameId
 ) {
   return cardService.getRandomRefreshedPlayableCard(gameId);
 }

 @GetMapping("/penalty")
 public RandomCardDto getPenaltyCard(
     @PathVariable("gameId") long gameId
 ) {
   return cardService.getRandomPenaltyCard(gameId);
 }
}
```

Файл GameController.java

```
@RestController
@RequestMapping(Endpoint.GAME)
@RequiredArgsConstructor
public class GameController {
  private final GameService gameService;
  private final RoleService roleService;
  private final MoveService moveService;

  @PostMapping
  public ResponseEntity<IdDto> create(
      @Valid @RequestBody CreateGameDto createGameDto
  ) {
    return ResponseEntity
        .status(HttpStatus.CREATED)
        .body(gameService.create(createGameDto));
  }

  @GetMapping("/{userId}")
  public List<GameEntity> getAllGamesOfUser(
      @PathVariable("userId") long userId
  ) {
    return gameService.getGames(userId);
  }

  @GetMapping("/game/{id}")
  public GameEntity getGame(
      @PathVariable("id") long id
  ) {
    return gameService.getGame(id);
  }

  @GetMapping("/game/{id}/mode")
  public void changeMode(
      @PathVariable("id") long id
  ) {
    gameService.changeMode(id);
  }

  @GetMapping("/game/{id}/reset")
  public void reset(
      @PathVariable("id") long id
      ) {
    List<RoleDto> roles = roleService.getRoles(id);
    roles.forEach(role -> roleService.reset(new GameRoleDto(role.getId(), 0)));
  }

  @DeleteMapping("/{id}")
  public void delete(
      @PathVariable("id") long gameId
  ) {
    moveService.deleteMovesForGame(gameId);
    gameService.delete(gameId);
  }
}
```

Файл RoleController.java

```
@RestController
@RequestMapping(Endpoint.ROLE)
@RequiredArgsConstructor
public class RoleController {
  private final RoleService roleService;

  @GetMapping
  public List<RoleDto> getRoles(
      @PathVariable("gameId") long gameId
  ) {
    return roleService.getRoles(gameId);
  }

  @PostMapping
  public ResponseEntity<IdDto> create(
      @Valid @RequestBody RoleDto roleDto
  ) {
    return ResponseEntity
        .status(HttpStatus.CREATED)
        .body(roleService.save(roleDto));
  }

  @PostMapping("/points")
  public void update(
      @Valid @RequestBody GameRoleDto roleDto
  ) {
    roleService.update(roleDto);
  }
}
```

Файл AuthController.java

```
@RestController
@RequestMapping(Endpoint.AUTH)
@RequiredArgsConstructor
public class AuthController {
  private final AuthService authService;
  private final UserService userService;

  @PostMapping("/register")
  public IdDto register(
      @Valid @RequestBody RegisterDto registerDto
  ) {
    return authService.register(registerDto);
  }

  @PostMapping("/login")
  public TokenDto login(
      @Valid @RequestBody LoginDto loginDto
  ) {
    return authService.login(loginDto);
  }

  @GetMapping("/info/{email}")
  public UserDto getUserInfo(
      @PathVariable("email") String email
  ) {
    Optional<UserDto> user = userService.findUserByEmail(email);
    return user.orElseThrow(() -> new IllegalArgumentException("No user with email " + email));
  }
```

```
  @PostMapping("/renew")
  public TokenDto renewAuth(
      @RequestBody TokenDto expiredTokenDto
  ) {
    return authService.renewAuth(expiredTokenDto);
  }

  @PostMapping("/reset")
  public MessageDto resetAuth(
      @Valid @RequestBody EmailDto emailDto
  ) {
    return authService.resetPassword(emailDto);
  }
}
```

Файл AuthService.java

```
public interface AuthService {
  IdDto register(RegisterDto registerDto);

  TokenDto login(LoginDto loginDto);

  TokenDto renewAuth(TokenDto tokenDto);

  MessageDto resetPassword(EmailDto emailDto);
}
```

Файл DefaultAuthService.java

```
@Service
@RequiredArgsConstructor
public class DefaultAuthService implements AuthService {
  private final AuthAPI auth;
  private final UserRepository userRepository;
  private final UserService userService;
  private final UserMapper userMapper;
  private final Auth0Properties auth0Properties;
  private final TokenMapper tokenMapper;

  @Transactional
  @Override
  public IdDto register(RegisterDto registerDto) {
    userService.findUserByEmailOrNickname(registerDto.getEmail(), registerDto.getNickname())
        .ifPresent(userDto -> {
          throw new AlreadyExistsException("User with this email or nickname already exists");
        });
    SignUpRequest request = auth.signUp(registerDto.getEmail(), registerDto.getPassword().toCharArray(),
auth0Properties.getConnection());

    CreatedUser createdUser = executeRequest(request);
    String userId = createdUser.getUserId();
    NewUserDto newUser = new NewUserDto(registerDto.getNickname(), registerDto.getFirstName(),
                                registerDto.getLastName(), registerDto.getEmail(), userId);
    UserEntity user = userMapper.toEntity(newUser);
    userRepository.save(user);

    return new IdDto(user.getId());
  }
```

```
  @Override
  public TokenDto login(LoginDto loginDto) {
    AuthRequest request = auth.login(loginDto.getEmail(), loginDto.getPassword().toCharArray(),
auth0Properties.getConnection())
        .setAudience(auth0Properties.getAudience())
        .setScope(auth0Properties.getScope());
    TokenHolder holder = executeRequest(request);
    return tokenMapper.toDto(holder);
  }

  @Override
  public TokenDto renewAuth(TokenDto expiredTokenDto) {
    AuthRequest request = auth.renewAuth(expiredTokenDto.getRefreshToken())
        .setScope(auth0Properties.getScope());
    TokenHolder holder = executeRequest(request);
    return tokenMapper.toDto(holder);
  }

  @Override
  public MessageDto resetPassword(EmailDto emailDto) {
    Request<Void> request = auth.resetPassword(emailDto.getEmail(), auth0Properties.getConnection());
    executeRequest(request);
    return new MessageDto("Email was sent");
  }

  private <T> T executeRequest(Request<T> request) {
    try {
      return request.execute();
    } catch (Auth0Exception exception) {
      throw new AuthProviderException("An exception happened in AuthProvider trying to login");
    }
  }
}
```

Файл DefaultCardService.java

```
public interface CardService {
  IdDto create(CardDto cardDto);
  void update(Long id, CardDto cardDto);

  void delete(Long id);

  RandomCardDto getRandomPlayableCard(Long gameId);

  RandomCardDto getRandomRefreshedPlayableCard(Long gameId);

  RandomCardDto getRandomPenaltyCard(Long gameId);

  List<FullCardDataDto> getAllCards(Long gameId);
}
```

Файл DefaultCardService.java

```
@Service
@RequiredArgsConstructor
public class DefaultCardService implements CardService {
  private static final Random RANDOM = new Random();

  private final CardMapper cardMapper;
  private final CardRepository cardRepository;
  private final CardTypeRepository cardTypeRepository;
```

```
private final CardRoleRepository cardRoleRepository;
private final MoveService moveService;
private final CardValidator cardValidator;

@Transactional
@Override
public IdDto create(CardDto cardDto) {
  cardValidator.ensureIfValid(cardDto);
  CardEntity entity = cardMapper.toEntity(cardDto);
  cardRepository.save(entity);
  cardRoleRepository.saveCardRoles(entity.getId(), cardDto.getRoleIds());
  return new IdDto(entity.getId());
}

@Transactional
@Override
public void update(Long id, CardDto cardDto) {
  Long gameId = cardTypeRepository.getGameIdByCardId(id);
  cardValidator.ensureIfValid(gameId, cardDto);
  Set<Long> roleIds = cardRoleRepository.getRolesIdsByCardId(id);
  Set<Long> newRoleIds = new HashSet<>(cardDto.getRoleIds());
  List<Long> roleIdsToInsert = getChangedIdsList(roleIds, newRoleIds);
  List<Long> roleIdsToDelete = getChangedIdsList(newRoleIds, roleIds);
  CardEntity entity = cardMapper.toEntity(cardDto);
  cardRepository.update(id, entity);
  if (roleIdsToInsert.size() > 0) {
    cardRoleRepository.saveCardRoles(id, roleIdsToInsert);
  }
  if (roleIdsToDelete.size() > 0) {
    cardRoleRepository.deleteCardRoles(id, roleIdsToDelete);
  }
}

@Override
public void delete(Long id) {
  cardRepository.deleteById(id);
  cardRoleRepository.deleteByCardId(id);
}

@Override
public List<FullCardDataDto> getAllCards(Long gameId) {
  return cardRepository.getAllCardsByGameId(gameId).stream()
      .map(cardMapper::toCardDto)
      .collect(Collectors.toList());
}

@Override
@Transactional
public RandomCardDto getRandomPlayableCard(Long gameId) {
  Long nextRoleId = moveService.getNextRoleId(gameId);
  CardProjection cardProjection = cardRepository.getRandomCard(gameId, true, nextRoleId);
  moveService.updateLastRoleId(gameId, nextRoleId);
  return mapCardProjectionToRandomCardDto(cardProjection);
}

@Override
@Transactional
public RandomCardDto getRandomRefreshedPlayableCard(Long gameId) {
  return getRandomCardCardForLastRole(gameId, true);
}

@Override
@Transactional
```

```
  public RandomCardDto getRandomPenaltyCard(Long gameId) {
    return getRandomCardCardForLastRole(gameId, false);
  }

  private RandomCardDto getRandomCardCardForLastRole(Long gameId, boolean isPlayable) {
    Long lastRoleId = moveService.getLastRoleId(gameId);
    CardProjection cardProjection = cardRepository.getRandomCard(gameId, isPlayable, lastRoleId);
    return mapCardProjectionToRandomCardDto(cardProjection);
  }

  private RandomCardDto mapCardProjectionToRandomCardDto(CardProjection cardProjection) {
    if (cardProjection == null) {
      return null;
    }
    RandomCardDto card = cardMapper.toRandomCardDto(cardProjection);
    return card;
  }

  private List<Long> getChangedIdsList(Set<Long> ids, Set<Long> newIds) {
    return new ArrayList<>(CollectionUtils.getDifference(newIds, ids));
  }
}
```

Файл GameService.java

```
public interface GameService {
  IdDto create(CreateGameDto createGameDto);

  List<GameEntity> getGames(Long userId);

  GameEntity getGame(Long id);

  void changeMode(Long gameId);

  void delete(Long gameId);
}
```

Файл DefaultGameService.java

```
@Service
@RequiredArgsConstructor
public class DefaultGameService implements GameService {
  private final GameRepository gameRepository;
  private final GameMapper gameMapper;

  @Override
  public IdDto create(CreateGameDto createGameDto) {
    GameEntity gameEntity = gameMapper.toEntity(createGameDto);
    gameRepository.save(gameEntity);
    return new IdDto(gameEntity.getId());
  }

  @Override
  @Transactional
  public List<GameEntity> getGames(Long userId) {
    return gameRepository.getAllGamesByUserId(userId);
  }

  @Override
```

```
  @Transactional
  public GameEntity getGame(Long id) {
    return gameRepository.getGameById(id);
  }

  @Override
  public void changeMode(Long id) {
    gameRepository.changeMode(id);
  }

  @Override
  public void delete(Long gameId) {
    gameRepository.delete(gameId);
  }
}
```

Файл MoveService.java

```
public interface MoveService {
  Long getLastRoleId (Long gameId);

  Long getNextRoleId (Long gameId);

  Boolean isFirstMove(Long gameId);

  void updateLastRoleId (Long gameId, Long nextRoleId);

  void deleteMovesForGame(Long gameId);
}
```

Файл DefaultMoveService.java

```
@Service
@RequiredArgsConstructor
public class DefaultMoveService implements MoveService {
  private final MoveRepository moveRepository;
  private final RoleRepository roleRepository;

  @Override
  public Long getLastRoleId(Long gameId) {
    return moveRepository.findLastRoleId(gameId);
  }

  @Override
  public Boolean isFirstMove(Long gameId) {
    return moveRepository.findLastRoleId(gameId) == null;
  }

  @Override
  public Long getNextRoleId(Long gameId) {
    return moveRepository.findNextRoleId(gameId, getLastRoleId(gameId));
  }

  @Override
  public void updateLastRoleId(Long gameId, Long nextRoleId) {
    moveRepository.updateLastRoleId(gameId, nextRoleId);
  }

  @Override
  public void deleteMovesForGame(Long gameId) {
```

```
    moveRepository.deleteMovesForGame(gameId);
  }

}
```

Файл RoleService.java

```
public interface RoleService {
  List<RoleDto> getRoles(Long gameId);

  IdDto save(RoleDto entity);

  void update(GameRoleDto entity);

  void reset(GameRoleDto entity);

  void delete(Long id);
}
```

Файл DefaultRoleService.java

```
@Service
@RequiredArgsConstructor
public class DefaultRoleService implements RoleService {
  private final RoleMapper roleMapper;
  private final RoleRepository roleRepository;

  private final CardRoleRepository cardRoleRepository;

  @Override
  public List<RoleDto> getRoles(Long gameId) {
    return roleRepository.getRolesByGameId(gameId).stream()
        .map(roleMapper::toDto)
        .collect(Collectors.toList());
  }

  @Transactional
  @Override
  public IdDto save(RoleDto role) {
    RoleEntity entity = roleMapper.toEntity(role);
    roleRepository.save(entity);
    return new IdDto(entity.getId());
  }

  @Transactional
  @Override
  public void update(GameRoleDto role) {
    Integer old_points = roleRepository.getRolePoints(role.getId());
    role.setPoints(role.getPoints() + old_points);
    roleRepository.update(role);
  }

  @Transactional
  @Override
  public void reset(GameRoleDto role) {
    role.setPoints(0);
    roleRepository.update(role);
  }

  @Transactional
```

```
  @Override
  public void delete(Long id) {
    roleRepository.deleteById(id);
    cardRoleRepository.deleteByRoleId(id);
  }
}
```

Файл UserService.java

```
public interface UserService {
  Optional<UserDto> findUserByEmail(String email);

  Optional<UserDto> findUserByEmailOrNickname(String email, String nickname);
}
```

Файл DefaultUserService.java

```
@Service
@RequiredArgsConstructor
public class DefaultUserService implements UserService {
  private final UserRepository userRepository;
  private final UserMapper userMapper;

  @Override
  public Optional<UserDto> findUserByEmail(String email) {
    UserEntity userEntity = userRepository.findUserByEmail(email).orElse(null);
    return Optional.ofNullable(userMapper.toDto(userEntity));
  }

  @Override
  public Optional<UserDto> findUserByEmailOrNickname(String email, String nickname) {
    UserEntity userEntity = userRepository.findUserByEmailOrNickname(email, nickname).orElse(null);
    return Optional.ofNullable(userMapper.toDto(userEntity));
  }
}
```

Файл CardTypeService.java

```
public interface CardTypeService {
  IdDto create(CreateCardTypeDto createCardTypeDto);

  List<GameCardTypeDto> getCardTypes(Long gameId);
}
```

Файл DefaultCardTypeService.java

```
@Service
@RequiredArgsConstructor
public class DefaultCardTypeService implements CardTypeService {
  private final CardTypeMapper cardTypeMapper;
  private final CardTypeRepository cardTypeRepository;

  @Override
  public List<GameCardTypeDto> getCardTypes(Long gameId) {
    return cardTypeRepository.getCardTypesByGameId(gameId)
        .stream()
        .map(cardTypeMapper::toDto).collect(Collectors.toList());
  }

  @Override
  public IdDto create(CreateCardTypeDto createCardTypeDto) {
```

```
    CardTypeEntity entity = cardTypeMapper.toEntity(createCardTypeDto);
    cardTypeRepository.save(entity);
    return new IdDto(entity.getId());
  }
}
```

Файл MoveRepository.java

```
@Mapper
public interface MoveRepository {
  Long findLastRoleId(
      @Param("gameId") Long gameId);

  Long findNextRoleId(
      @Param("gameId") Long gameId,
      @Param("lastRoleId") Long lastRoleId);

  void updateLastRoleId(
      @Param("gameId") Long gameId,
      @Param("nextRoleId") Long nextRoleId);

  void deleteMovesForGame(
      @Param("gameId") Long gameId);
}
```

Файл MoveRepository.xml

```
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.gameapi.repository.MoveRepository">

    <delete id="deleteMovesForGame">
        delete from moves
        where game_id = #{gameId}
    </delete>

    <select id="findLastRoleId" resultType="java.lang.Long">
        select m.last_role_id
        from public.moves m
        where m.game_id = #{gameId}
    </select>

    <select id="findNextRoleId" resultType="java.lang.Long">
        select *
        from ((select r.id
               from public.roles r
               where r.game_id = #{gameId}
                 and r.id > #{lastRoleId}
               order by r.id
               limit 1)
              union
              (select min(r.id) as res_id
               from public.roles r
               where r.game_id = #{gameId})) as res
        order by res.id desc
        limit 1
    </select>

    <update id="updateLastRoleId">
        update public.moves
```

```
        set last_role_id = #{nextRoleId}
        where game_id = #{gameId};

        insert into public.moves (game_id, last_role_id)
        select #{gameId}, #{nextRoleId}
        where not exists (
              select 1 from public.moves where game_id = #{gameId}
            );
    </update>
</mapper>
```

Файл CardRepository.java

```
@Mapper
public interface CardRepository {
  void save(
      @Param("entity") CardEntity entity);

  void update(
      @Param("id") Long id,
      @Param("entity") CardEntity entity);

  void deleteById(
      @Param("id") Long id);

  CardProjection getRandomCard(
      @Param("gameId") Long gameId,
      @Param("isPlayable") boolean isPlayable,
      @Param("roleId") Long roleId);

  List<CardProjection> getAllCardsByGameId(
      @Param("gameId") Long gameId);
}
```

Файл CardRepository.xml

```
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.gameapi.repository.CardRepository">
    <insert id="save" useGeneratedKeys="true" keyColumn="id" keyProperty="entity.id">
        insert into public.cards(
            type_id,
            description,
            points)
        values (#{entity.typeId},
                #{entity.description},
                #{entity.points})
    </insert>

    <update id="update">
        update cards
        set type_id     = #{entity.typeId},
            description = #{entity.description},
            points = #{entity.points}
        where id = #{id}
    </update>

    <delete id="deleteById">
        delete
        from cards
        where id = #{id}
```

```
</delete>

<select id="getRandomCard" resultMap="card">
    select
    c.id as id,
    c.description as description,
    c.points as points,
    ct.title as card_type_title,
    ct.color as card_type_color,
    ct.is_playable as card_type_is_playable,
    ct.is_penalty as card_type_is_penalty,
    r.id as role_id,
    r.title as role_title,
    r.points as role_points
    from cards c
    left join card_types ct on ct.id = c.type_id
    left join cards_roles cr on c.id = cr.card_id
    left join roles r on r.id = cr.role_id
    <if test="isPlayable">
        where ct.is_playable and cr.role_id = #{roleId}
    </if>
    <if test="!isPlayable">
        where ct.is_penalty and cr.role_id = #{roleId}
    </if>
    order by random()
    limit 1
</select>

<select id="getAllCardsByGameId" resultMap="card">
    select c.id          as id,
           c.description as description,
           c.points as points,
           ct.title      as card_type_title,
           ct.color      as card_type_color,
           r.id          as role_id,
           r.title       as role_title
    from cards c
             left join card_types ct on ct.id = c.type_id
             left join cards_roles cr on c.id = cr.card_id
             left join roles r on r.id = cr.role_id
    where ct.game_id = #{gameId}
    order by c.id
</select>

<resultMap id="card" type="com.example.gameapi.entity.projection.CardProjection">
    <id column="id" property="id"/>
    <result column="description" property="description" />
    <result column="points" property="points" />

    <association property="type" javaType="com.example.gameapi.entity.CardTypeEntity">
        <result column="card_type_title" property="title" />
        <result column="card_type_color" property="color" />
        <result column="card_type_is_playable" property="isPlayable" />
        <result column="card_type_is_penalty" property="isPenalty" />
    </association>

    <association property="role" javaType="com.example.gameapi.entity.RoleEntity">
        <result column="role_id" property="id" />
        <result column="role_title" property="title" />
        <result column="role_points" property="points" />
    </association>

    <collection property="roles" javaType="List" ofType="com.example.gameapi.entity.RoleEntity">
```

```
        <result column="role_id" property="id" />
        <result column="role_title" property="title" />
        <result column="role_points" property="points" />
      </collection>
  </resultMap>
</mapper>
```

Файл AudienceValidator.java

```
class AudienceValidator implements OAuth2TokenValidator<Jwt> {
  private final String audience;

  public AudienceValidator(String audience) {
    this.audience = audience;
  }

  public OAuth2TokenValidatorResult validate(Jwt jwt) {
    if (jwt.getAudience().contains(audience)) {
      return OAuth2TokenValidatorResult.success();
    }
    OAuth2Error error = new OAuth2Error("invalid_token", "The required audience is missing", null);
    return OAuth2TokenValidatorResult.failure(error);
  }
}
```

Файл JwtConverter.java

```
@Component
@RequiredArgsConstructor
public class JwtConverter implements Converter<Jwt, AbstractAuthenticationToken> {
  private final UserService userService;

  @Override
  public AbstractAuthenticationToken convert(Jwt source) {
    UserDto user = userService.findUserByEmail(source.getClaim("https://example.com/email")).orElse(null);
    return new UserPrincipal(user, Collections.emptySet());
  }
}
```

Файл SecurityConfig.java

```
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig {
  private final JwtConverter jwtConverter;
  @Value("${auth0.audience}")
  private String audience;
  @Value("${spring.security.oauth2.resourceserver.jwt.issuer-uri}")
  private String issuer;

  @Bean
  JwtDecoder jwtDecoder() {
    NimbusJwtDecoder jwtDecoder = JwtDecoders.fromOidcIssuerLocation(issuer);
    OAuth2TokenValidator<Jwt> audienceValidator = new AudienceValidator(audience);
    OAuth2TokenValidator<Jwt> withIssuer = JwtValidators.createDefaultWithIssuer(issuer);
    OAuth2TokenValidator<Jwt> withAudience = new DelegatingOAuth2TokenValidator<>(withIssuer,
audienceValidator);
    jwtDecoder.setJwtValidator(withAudience);
    return jwtDecoder;
  }

  @Bean
```

```
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests(auth -> auth
            .mvcMatchers(
                Endpoint.GAME + "/**",
                Endpoint.CARD_TYPE + "/**",
                Endpoint.CARD + "/**"
            ).authorized()
            .mvcMatchers(Endpoint.AUTH + "/**").permitAll())
        .csrf().disable()
        .oauth2ResourceServer(oauth2 -> oauth2
            .jwt(jwt -> jwt
                .jwtAuthenticationConverter(jwtConverter)
            )
        );
    return http.build();
  }

}
```

Файл Game.js

```
import {useEffect, useState} from "react";
import './Game.css'
import axios from "axios";
import Card from "./Card";
import {useContext} from "react";
import {UserContext} from "../../context/UserContext";
import * as earth from "../../asset/image/Earth.jpg";
import * as moon from "../../asset/image/Moon.jpg";
import * as mars from "../../asset/image/Mars.jpg";
import * as saturn from "../../asset/image/Saturn.jpg";
import Rocket from "./Rocket"
import {PlayersProgress, TeamProgress} from "./PlayersProgress";
import Winner from "./Winner";

const Game = ({id}) => {

    const { user } = useContext(UserContext);

    const [card, setCard] = useState({});
    const [game, setGame] = useState({});
    const [moves, setMoves] = useState([]);
    const [players, setPlayers] = useState([]);
    const [winners, setWinners] = useState([]);
    const [teamRocketPosition, setTeamRocketPosition] = useState({});
    const [showWinnerPopup, setShowWinnerPopup] = useState(false);
    const [isTeam, setIsTeam] = useState(false);
    const [previousPlayer,  setPreviousPlayer] = useState({});

    const planets = [
        {
            title: 'Земля',
            pointsToGet: 0,
            src: earth.default
        },
        {
            title: 'Місяць',
            pointsToGet: game.goal/4,
            src: moon.default
        },
        {
            title: 'Марс',
            pointsToGet: 2*(game.goal/5),
            src: mars.default
        },
        {
            title: 'Сатурн',
            pointsToGet: game.goal,
            src: saturn.default
        },
    ]

    const levels = [
        {
            title: 'Астронавт Початківець',
            pointsToGet: 0,
            stars: 1
        },
        {
            title: 'Дослідник',
            pointsToGet: game.goal / 4,
```

```
        stars: 2
      },
      {
        title: 'Помічник Капітана',
        pointsToGet: game.goal / 2,
        stars: 3
      },
      {
        title: 'Капітан',
        pointsToGet: game.goal / 100 * 75,
        stars: 4
      },
      {
        title: 'Легенда',
        pointsToGet: game.goal / 100 * 90,
        stars: 5
      }
    ]

    useEffect(() => {
        axios.get(`http://localhost:8080/api/games/${id}/cards/next`, {})
          .then((response) => {
            const updatedData = {
              ...response.data,
              gameId: id
            };
            setCard(updatedData);
          })
          .catch((error) => {
            console.log("Error retrieving card:", error);
          });
      axios
        .get(`http://localhost:8080/api/games/game/${id}`, {
          headers: {
            Authorization: `Bearer ${user.token.accessToken}`,
          },
        })
        .then((response) => {
          setGame(response.data);
          setIsTeam(response.data.isTeam);
        })
        .catch((error) => {
          console.log("Error retrieving game:", error);
        });

        axios
          .get(`http://localhost:8080/api/games/${id}/roles`, {
            headers: {
              Authorization: `Bearer ${user.token.accessToken}`,
            },
          })
          .then((response) => {
            let totalPoints = 0
            const updatedPlayers = response.data.map((p) => {
              totalPoints += p.points;
              let lastPlanet = planets[0];
              let lastLevel = levels[0];

              for (const level of levels) {
                if (level.pointsToGet <= p.points) {
                  lastLevel = level;
                } else {
                  break;
```

```
                }
            }

            for (const planet of planets) {
                if (planet.pointsToGet <= p.points) {
                    lastPlanet = planet;
                } else {
                    break;
                }
            }
            return {
                ...p,
                level: lastLevel,
                planet: lastPlanet,
                progress: p.points / game.goal * 100
            };
        });

        setPlayers(updatedPlayers);
        let progress = totalPoints / game.goal * 100
        let teamPlanet = planets[0];

        for (const planet of planets) {
            if (planet.pointsToGet <= totalPoints) {
                teamPlanet = planet;
            } else {
                break;
            }
        }
        setTeamRocketPosition({progress: progress, points: totalPoints, planet: teamPlanet});
    })
    .catch((error) => {
        console.log("Error retrieving roles:", error);
    });
}, [id, user.token.accessToken]);

const setPoints = () => {
    axios.post(`http://localhost:8080/api/games/${id}/roles/points`, {
        id: card.role.id,
        points: card.points
    }).catch((error) => {
        console.log("Error updating points:", error);
    });
}

const recordMove = (action) => {
    let newMove  = {
        description: card.description,
        role: card.role.title
    }

    if (action === 1) {
        newMove.status = "Виконано"
        newMove.points = card.points
        newMove.totalPoints = card.role.points + card.points
    } else if (action === 2) {
        newMove.status = "Замінено"
    } else if (action === 3) {
        newMove.status = "Не виконано"
    }
    setMoves([...moves, newMove])
}
```

```
const finishMove = async (status) => {
    let player = players.find(p => p.id === card.role.id);

    if (status === 1) {
        player.points = player.points + card.points;
        player.progress = player.points / game.goal * 100;
    }

    let teamPoints = 0;
    let newPos = {
        points: teamPoints,
        progress: teamPoints / game.goal * 100,
        planet: planets[0]
    }

    // Calculating progress
    if (isTeam) {
        teamPoints = players.reduce((sum, player) => sum + player.points, 0);
        newPos.points = teamPoints;
        newPos.progress = teamPoints / game.goal * 100;
    }

    // Levels
    for (let i = levels.length - 1; i >= 1; i--) {
        if (levels[i].pointsToGet <= player.points) {
            player.level = levels[i]
            break;
        }
    }

    // Planets
    if (isTeam) {
        for (let i = planets.length - 1; i >= 1; i--) {
            if (planets[i].pointsToGet <= teamPoints) {
                newPos.planet = planets[i];
                break;
            }
        }
    } else {
        for (let i = planets.length - 1; i >= 1; i--) {
            if (planets[i].pointsToGet <= player.points) {
                player.planet = planets[i]
                break;
            }
        }
    }

    // Set winners
    if (isTeam) {
        if (teamPoints >= game.goal) {
            await setShowWinnerPopup(true);
            setWinners((prevWinners) => [...prevWinners, { player: "Team", place: 1 }]);
        }
    } else {
        if (player.points >= game.goal) {
            const isPlayerAlreadyInWinners = winners.some(
                (winner) => winner.player === player.title
            );
            if (!isPlayerAlreadyInWinners) {
                await setShowWinnerPopup(true);
                setWinners((prevWinners) => [...prevWinners, {
                    player: player.title,
                    place: prevWinners.length + 1
```

```
                }]);
            }
        }
    }

    setTeamRocketPosition(newPos);

    // Update players
    setPlayers(prevPlayers => {
        return prevPlayers.map(p => {
            if (p.id === player.id) {
                return {
                    ...p,
                    level: player.level,
                    planet: player.planet,
                    points: player.points,
                    progress: player.progress,
                    isWinner: player.isWinner,
                };
            }
            return p;
        });
    });
}

const next = async () => {
    await setPoints();
    await setPreviousPlayer(players.find(p => p.id === card.role.id));
    await finishMove(1);

    await recordMove(1);
    axios.get(`http://localhost:8080/api/games/${id}/cards/next`, {
    })
        .then((response) => {
            const updatedData = {
                ...response.data,
                gameId: id
            };
            setCard(updatedData);

        })
        .catch((error) => {
            console.log("Error retrieving card:", error);
        });
}

const refresh = () => {
    recordMove(2)
    axios.get(`http://localhost:8080/api/games/${id}/cards/refresh`, {
    })
        .then((response) => {
            const updatedData = {
                ...response.data,
                gameId: id
            };
            setCard(updatedData);
        })
        .catch((error) => {
            console.log("Error retrieving card:", error);
        });
}

const penalty = async () => {
```

```
    await setPreviousPlayer(players.find(p => p.id === card.role.id));
     await finishMove(2);

     await recordMove(3);
    axios.get(`http://localhost:8080/api/games/${id}/cards/penalty`, {
    })
       .then((response) => {
          const updatedData = {
             ...response.data,
             gameId: id
          };
          setCard(updatedData);
       })
       .catch((error) => {
          console.log("Error retrieving card:", error);
       });
}

if (Object.keys(teamRocketPosition).length === 0) {
    teamRocketPosition.progress = 0;
    teamRocketPosition.planet = planets[0]
}

    return (
       <div>
          {showWinnerPopup && (
             <Winner
                player={previousPlayer.title}
                moves={moves}
                teamMode={isTeam}
                winners = {winners}
                onClose={() => setShowWinnerPopup(false)}
             />
          )}

          <div className="game-container">
             <div className={"space"}>
                <div className="rocket-container">
                   {isTeam ? (
                      <div className={`rocket-item team`}>
                         <Rocket progress={teamRocketPosition.progress} name={"Екіпаж"} />
                      </div>
                   ) : (
                      players.map((player) => (
                         <div className="rocket-item" key={player.id}>
                             <Rocket progress={player.progress} name={player.title}/>
                         </div>
                      ))
                   )}
                </div>

                <div className="planet-container">
                   {planets.map((planet, index) => (
                      <div className="planet-item" key={index}>
                         <img src={planet.src} alt={planet.title} />
                      </div>
                   ))}
                </div>
             </div>
             {Object.keys(card).length > 0 ?
                <div>
                   <h2>Твій хід {card.role.title}</h2>
```

```
                <Card card={card} next={next} refresh={refresh} penalty={penalty}/>
              </div>
            : null}

            {isTeam ?

            <TeamProgress team={teamRocketPosition} />
              :
            <PlayersProgress players={players} />
            }
          </div>
        </div>
      );

}
export default Game;
```

Файл LayoutRules.js

```
import { Route, Routes } from "react-router-dom";
import { pages } from "../../../meta/page";
import Navbar from "../../Navbar/Navbar";
import HomePage from "../../page/home/HomePage";
import LoginPage from "../../page/login/LoginPage";
import ProfilePage from "../../page/profile/ProfilePage";
import MyAccountPage from "../../page/myAccount/MyAccount";
import NewGame from "../../page/newGame/NewGame";
import HelpPage from "../../page/help/Help";
import GamePage from "../../page/game/GamePage";
import ContactUs from "../../ContactUs/ContactUs";

export const LayoutRoutes = () => {
  return (
    <>
      <Navbar />
      <Routes>
        <Route path={pages.home} element={<HomePage />} />
        <Route path={pages.login} element={<LoginPage />} />
        <Route path={pages.profile} element={<ProfilePage />} />
        <Route path={pages.myAccount} element={<MyAccountPage />} />
        <Route path={pages.newGame} element={<NewGame />} />
        <Route path={"/my-account/game/:id"} element={<GamePage />} />
        <Route path={pages.help} element={<HelpPage />} />
      </Routes>
      <ContactUs />
      <footer className={"fixed-footer"} />
    </>
  );
}

export default LayoutRoutes;
```

Файл LoginPage.js

```
import React, {useState} from "react";
import {LoginForm} from "../../Auth/LoginForm";
import "./LoginPage.css";
import {RegistrationForm} from "../../Auth/RegistrationForm";

const LoginPage = () => {
  const [isRightSideActive, setIsRightSideActive] = useState(false);

  function SignInChange() {
```

```
    setIsRightSideActive(false);
  }

  function SignUpChange() {
    setIsRightSideActive(true);
  }

  return (
    <div>
    <div className={isRightSideActive ? 'right-panel-active container centered' : 'container centered'}
id="container">
      <div className="form-container sign-up-container">
        <RegistrationForm />
      </div>
      <div className="form-container sign-in-container">
        <LoginForm />
      </div>
      <div className="overlay-container">
        <div className="overlay">
          <div className="overlay-panel overlay-left">
            <h1>Вітаю знову!</h1>
            <p>Вже маєш акаунт?</p>
            <button className="ghost" id="signIn" onClick={SignInChange}>Увійти</button>
          </div>
          <div className="overlay-panel overlay-right">
            <h1>Привіт, друже!</h1>
            <p>Ще не маєш акаунту?</p>
            <button className="ghost" id="signUp" onClick={SignUpChange}>Зареєструватись</button>
          </div>
        </div>
      </div>
    </div>
  );
}

export default LoginPage;
```

Файл MyAccount.js

```
import GameList from "../../GamesList/GameList";
import {useContext} from "react";
import {UserContext} from "../../../context/UserContext";
import LoginPage from "../login/LoginPage";

const MyAccountPage = () => {

  const { user } = useContext(UserContext);

  if (!user) {
    return(
      <LoginPage />
    )
  } else {
    return (
      <div>
        <div className={"my-games-container"}>
          <h2>Мої ігри</h2>
          <GameList />
        </div>
      </div>
    );
```

```
    }
}

export default MyAccountPage;
```

Файл GameList.js

```
import React, { useState, useEffect } from 'react';
import {Link, useNavigate} from 'react-router-dom';
import axios from 'axios';
import {useContext} from "react";
import {UserContext} from "../../context/UserContext";
import CreateButton from "./CreateButton";
import {FontAwesomeIcon} from "@fortawesome/react-fontawesome";
import {faTrash, faEraser, faToggleOn} from "@fortawesome/free-solid-svg-icons";

import "./../Game/Game.css";

const GameList = () => {

    const [games, setGames] = useState([]);
    const [showConfirmation, setShowConfirmation] = useState(false);
    const [confirmationAction, setConfirmationAction] = useState("");
    const [selectedGameId, setSelectedGameId] = useState("");
    const { user } = useContext(UserContext);
    const navigate = useNavigate();

    useEffect(() => {
        axios.get(`http://13.51.85.16:8080/api/games/${user.id}`, {
            headers: {
                Authorization: `Bearer ${user.token.accessToken}`
            }
        })
            .then(response => setGames(response.data))
            .catch(error => console.log(error));
    }, [user.id, user.token.accessToken]);

    const navigateToCreateGame = () => {
        navigate("/my-account/new-game")
    }

    const handleRemoveGame = (gameId) => {
        setSelectedGameId(gameId);
        setConfirmationAction("видалити");
        setShowConfirmation(true);
    }

    const handleReset = (gameId) => {
        setSelectedGameId(gameId);
        setConfirmationAction("стерти");
        setShowConfirmation(true);
    };

    const handleChangeMode = (gameId) => {
        setSelectedGameId(gameId);
        setConfirmationAction("перемкнути на інший режим");
        setShowConfirmation(true);
    };

    const confirmAction = () => {
        switch (confirmationAction) {
            case "видалити":
```

```
            removeGame(selectedGameId);
            break;
        case "стерти":
            resetGame(selectedGameId);
            break;
        case "перемкнути на інший режим":
            changeGameMode(selectedGameId);
            break;
        default:
            break;
    }
    setShowConfirmation(false);
}

const removeGame = (gameId) => {
    axios.delete(`http://13.51.85.16:8080/api/games/${gameId}`)
        .then(response => {
            setGames(prevGames => prevGames.filter(game => game.id !== gameId));
        })
        .catch(error => console.log(error));
}

const resetGame = (gameId) => {
    axios.get(`http://13.51.85.16:8080/api/games/game/${gameId}/reset`)
        .catch(error => console.log(error));
};

const changeGameMode = (gameId) => {
    axios.get(`http://13.51.85.16:8080/api/games/game/${gameId}/mode`)
        .catch(error => console.log(error));
};

return (
    <div>
        <CreateButton onSubmit={navigateToCreateGame} />

        <div className={"games-added-items centered"}>
            {games.map(game => (
                <div className={"game-card"}>
                    <Link to={`/my-account/game/${game.id}`} >
                        <div className={game.isTeam ? "game-card-bg" : "game-card-bg green"}></div>
                        <h3 className={"card-title"}>{game.title}</h3>
                        <p className={"game-card-description"}>{game.description}</p>
                    </Link>
                    <div className={"game-buttons"}>
                    <button onClick={() => handleRemoveGame(game.id)} className={"remove-item-on-card"}>
                        <FontAwesomeIcon icon={faTrash} color={"black"}/>
                    </button>
                    <button onClick={() => handleReset(game.id)} className={"settings-on-card"}>
                        <FontAwesomeIcon icon={faEraser} color={"black"}/>
                    </button>
                    <button onClick={() => handleChangeMode(game.id)} className={"settings-on-card"}>
                        <FontAwesomeIcon icon={faToggleOn} color={"black"}/>
                    </button>
                    </div>
                </div>
            ))}
        </div>

        {showConfirmation && (
            <div className="popup">
                <div className="popup-content">
                    <h2>Увага!</h2>
```

```
                <p>Ви впевнені що хочете {confirmationAction} цю гру?</p>
                <div className="popup-buttons">
                  <button onClick={confirmAction} className="popup-confirm">Так</button>
                  <button onClick={() => setShowConfirmation(false)} className="popup-cancel">Ні</button>
                </div>
              </div>
            </div>
          )}
        </div>
    );
}

export default GameList;
```

Файл NewGame.js

```
import React, { useState, useContext } from 'react';
import GameBaseInfoForm from './../../GamesList/CreateGame/GameBaseInfoForm';
import GameCardCategoriesForm from './../../GamesList/CreateGame/GameCardCategoriesForm';
import GameRolesForm from './../../GamesList/CreateGame/GameRolesForm';
import {UserContext} from "../../../context/UserContext";
import "./../../GamesList/CreateGame/CreateGame.css"
import CardsForm from "../../GamesList/CreateGame/CardsForm";
import LoginPage from "../login/LoginPage";

function NewGame() {
    const [gameData, setGameData] = useState({});
    const [gameId, setGameId] = useState(null);
    const [step, setStep] = useState(1);
    const [errorMessage, setErrorMessage] = useState(null);
    const {user} = useContext(UserContext);

    const handleBaseInfoSubmit = async (baseInfo) => {
        try {
            const response = await fetch('http://13.51.85.16:8080/api/games', {
                method: 'POST',
                body: JSON.stringify({
                    userId: user.id,
                    title: baseInfo.title,
                    description: baseInfo.description,
                    goal: baseInfo.goal,
                    isTeam: baseInfo.teamMode
                }),
                headers: {
                    'Content-Type': 'application/json',
                     'Authorization': `Bearer ${user.token.accessToken}`
                },
            });
            if (!response.ok) {
                throw new Error('Не можу створити гру');
            }
            const data = await response.json();
            setGameId(data.id);
            setGameData({...gameData, ...baseInfo});
            setStep(2);
        } catch (error) {
            setErrorMessage(error.message);
            console.log(errorMessage);
        }
    };
```

```
const handleCardCategoriesSubmit = async (categories) => {
    try {
        const cardTypePromises = categories.map((category) =>
            fetch('http://13.51.85.16:8080/api/card-types', {
                method: 'POST',
                body: JSON.stringify({
                    gameId: gameId,
                    title: category.title,
                    color: category.color,
                    isPlayable: !category.isPenalty,
                    isPenalty: category.isPenalty
                }),
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${user.token.accessToken}`
                },
            }).then((response) => {
                if (!response.ok) {
                    throw new Error(`Не можу додати категорію ${category.title}`);
                }
            })
        );
        await Promise.all(cardTypePromises);
        setGameData({...gameData, ...categories});
        setStep(3);
    } catch (error) {
        setErrorMessage(error.message);
        console.log(errorMessage);
    }
};

const handleRolesSubmit = async (roles) => {
    try {
        const rolePromises = roles.map((role) =>
            fetch(`http://13.51.85.16:8080/api/games/${gameId}/roles`, {
                method: 'POST',
                body: JSON.stringify({
                    gameId: gameId,
                    title: role.title,
                    points: 0
                }),
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${user.token.accessToken}`
                },
            }).then((response) => {
                if (!response.ok) {
                    throw new Error(`Не можу додати роль ${role.title}`);
                }
            })
        );
        await Promise.all(rolePromises);
        setGameData({...gameData, roles});
        setStep(4);
    } catch (error) {
        setErrorMessage(error.message);
        console.log(errorMessage);
    }
};

const handleCardsSubmit = async (cards) => {
    try {
        const cardPromises = cards.map((card) =>
```

```
            fetch(`http://13.51.85.16:8080/api/games/${gameId}/cards`, {
                method: 'POST',
                body: JSON.stringify({
                    description: card.description,
                    points: card.points,
                    typeId: card.typeId,
                    roleIds: card.roleIds
                }),
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${user.token.accessToken}`
                },

            }).then((response) => {
                if (!response.ok) {
                    throw new Error(`Не можу створити картку ${card.title}`);
                }
            })
        );
        await Promise.all(cardPromises);
        setGameData({...gameData, cards});
        setStep(5);
    } catch (error) {
        setErrorMessage(error.message);
        console.log(errorMessage);
    }
};

const renderForm = () => {
    switch (step) {
        case 1:
            return <GameBaseInfoForm onSubmit={handleBaseInfoSubmit}/>;
        case 2:
            return <div>
                    <h2>Ігрові категорії</h2>
                    <GameCardCategoriesForm key="cardCategories" onSubmit={handleCardCategoriesSubmit}/>
                </div>;
        case 3:
            return <div>
                    <h2>Ролі</h2>
                    <GameRolesForm onSubmit={handleRolesSubmit}/>
                </div>;
        case 4:
            return <div>
                    <h2>Завдання</h2>
                    <CardsForm onSubmit={handleCardsSubmit} gameId={gameId}/>
                </div>;
        default:
            return <div className={"game-created"}>Гра успішно створена!</div>;
    }
};

if (!user) {
    return(
        <LoginPage />
    )
}  else {
    return (
        <div>
            <div className={"new-game-container centered"}>
                {renderForm()}
            </div>
```

```
            </div>
        );
    }
}
export default NewGame;
```

Файл GameBaseInfoForm.js

```
import React, { useState } from 'react';
import  "./CreateGame.css"

function GameBaseInfoForm({ onSubmit }) {
    const [title, setTitle] = useState('');
    const [description, setDescription] = useState('');
    const [goal, setGoal] = useState(0);
    const [teamMode, setTeamMode] = useState(false);

    const handleCheckboxChange = () => {
        setTeamMode(!teamMode);
    };

    const handleSubmit = (event) => {
        event.preventDefault();
        if (goal === 0) {
            setGoal(100);
        }
        onSubmit({ title, description, goal, teamMode});
    };

    return (
        <form onSubmit={handleSubmit} className={"base-info-form"}>
            <label htmlFor="title">Назва</label>
            <input
                id="title"
                type="text"
                value={title}
                onChange={(event) => setTitle(event.target.value)}
                required
                className="title-input centered"
            />

            <label htmlFor="description">Опис</label>
            <textarea
                id="description"
                value={description}
                onChange={(event) => setDescription(event.target.value)}
                required
                className="description-input centered"
            />

            <label htmlFor="title">Фініш гри</label>
            <input
                id="goal"
                type="number"
                value={goal}
                onChange={(event) => setGoal(event.target.value)}
                defaultValue={100}
                className="title-input centered"
            />

            <div>
                <input
```

```
            type="checkbox"
            checked={teamMode}
            onChange={handleCheckboxChange}
            id="mode"
          />
          <label htmlFor="mode">Ввімкнути командний режим</label>
        </div>

        <button type="submit" className="form-continue-button">Продовжити</button>
      </form>
    );
}

export default GameBaseInfoForm;
```

Файл GameCardCategoriesForm.js

```
import React, { useState } from 'react';
import { ChromePicker } from 'react-color';
import CreateGameListItem from "../CreateButton";
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faTrash } from '@fortawesome/free-solid-svg-icons';

const MIN_TITLE_LENGTH = 3;
const MAX_TITLE_LENGTH = 50;

const penaltyCategories = [
    {
      title: "Космічний монстр!",
      color: "#45ad27",
      isPenalty: true
    },
    {
      title: "Корабель зламався!",
      color: "#f02211",
      isPenalty: true
    },
    {
      title: "Метеоритний дощ!",
      color: "#3498db",
      isPenalty: true
    }
]

const initialCategories = [
    { title: '', color: '#f9b23', isPenalty: false },
    ...penaltyCategories,
];

function GameCardCategoriesForm({ onSubmit }) {
    const [categories, setCategories] = useState(initialCategories);
    const handleSubmit = async (event) => {
      event.preventDefault();

      const isFormValid = categories.every((category) => {
        return category.title.length >= MIN_TITLE_LENGTH && category.title.length <=
MAX_TITLE_LENGTH;
      });

      if (isFormValid) {
        onSubmit(categories);
        setCategories([]);
      } else {
```

```
            alert(`Будь ласка, переконайтесь що всі назви категорій знаходяться в межах від
${MIN_TITLE_LENGTH} до ${MAX_TITLE_LENGTH} символів.`);
        }
    };

    const handleCategoryChange = (index, field, value) => {
        const updatedCategories = [...categories];
        updatedCategories[index][field] = value;
        setCategories(updatedCategories);
    };

    const handleAddCategory = () => {
        setCategories([...categories, { title: '', color: '', isPenalty: false }]);
    };

    const handleRemoveCategory = (index) => {
        const updatedCategories = [...categories];
        updatedCategories.splice(index, 1);
        setCategories(updatedCategories);
    };

    return (
        <div>
        <form onSubmit={handleSubmit} className={"w-50 centered category-form"}>
            {categories.map((category, index) => (
                <div key={index} className={"add-category-card add-item"}>
                    <div><label htmlFor={`categoryTitle${index}`}>Назва</label></div>
                    <input
                        id={`categoryTitle${index}`}
                        type="text"
                        value={category.title}
                        onChange={(event) => handleCategoryChange(index, 'title', event.target.value)}
                        required
                        className={"w-50 centered category-title"}
                    />
                    <div className={"colorpicker-container"}>
                        <label htmlFor={`categoryColor${index}`}>Колір</label>
                        <ChromePicker
                            color={category.color}
                            onChangeComplete={(color) => handleCategoryChange(index, 'color', color.hex)}
                            required
                            className={"w-50 centered"}
                        />

                    </div>
                    <button type="button" onClick={() => handleRemoveCategory(index)} className={"remove-item"}>
                        <FontAwesomeIcon icon={faTrash} color={"black"}/>
                    </button>
                </div>
            ))}

            <CreateGameListItem onSubmit={handleAddCategory}/>

            {categories.length > 0 ? (
                <>
                    <h2>Додані категорії</h2>
                    <ul className="categories-added-items">
                        {categories.map((category, index) => (
                            category.title.length > 0 ? (
                                <li key={index} className="category-card">
                                    <div className="game-card-bg" style={{ backgroundColor: category.color }}></div>
                                    <div>
                                        <div className="card-title category-card-title">{category.title}</div>
```

```
                        </div>
                        <button onClick={() => handleRemoveCategory(index)} className={"remove-item-on-
card"}>
                            <FontAwesomeIcon icon={faTrash} color={"black"}/>
                        </button>
                    </li>
                ) : null
            ))}
        </ul>
    </>
) : null}

    <button type="submit">Продовжити</button>

</form>

</div>
);
}

export default GameCardCategoriesForm;
```

Файл GameRolesForm.js

```
import React, { useState } from 'react';
import {FontAwesomeIcon} from "@fortawesome/react-fontawesome";
import {faTrash} from "@fortawesome/free-solid-svg-icons";
import CreateGameListItem from "../CreateButton";

const MIN_TITLE_LENGTH = 3;
const MAX_TITLE_LENGTH = 20;

function GameRolesForm({ onSubmit }) {
  const [newRoles, setNewRoles] = useState([{ title: '' }]);

  const handleAddRole = () => {
    setNewRoles([...newRoles, { title: '' }]);
  };

  const handleRemoveRole = (index) => {
    const updatedRoles = [...newRoles];
    updatedRoles.splice(index, 1);
    setNewRoles(updatedRoles);
  };

  const handleRoleChange = (index, value) => {
    const updatedRoles = [...newRoles];
    updatedRoles[index].title = value;
    setNewRoles(updatedRoles);
  };

  const handleSubmit = (event) => {
    event.preventDefault();

    const isFormValid = newRoles.every((role) => {
      return role.title.length >= MIN_TITLE_LENGTH && role.title.length <= MAX_TITLE_LENGTH;
    });

    if (isFormValid) {

      onSubmit(newRoles);
    } else {
```

```
        alert(`Please make sure all category titles are between ${MIN_TITLE_LENGTH} and
${MAX_TITLE_LENGTH} characters long.`);
      }
    };

    return (
      <form onSubmit={handleSubmit} className={"w-50 centered roles-form"}>
        {newRoles.map((role, index) => (
          <div key={index}>
            <input
              type="text"
              value={role.title}
              onChange={(event) => handleRoleChange(index, event.target.value)}
              required
              className={"centered role-title"}
            />

          </div>
        ))}

        <CreateGameListItem onSubmit={handleAddRole}/>

        {newRoles.length > 1 ? (
          <>
            <h2 className={"added-items-heading"}>Додані ролі</h2>
            <ul className="roles-added-items">
              {newRoles.map((role, index) => (
                role.title.length > 0 ? (
                  <li key={index} className={"role-card"}>
                    {role.title}
                    <button type="button" onClick={() => handleRemoveRole(index)} className={"remove-
item-on-card"}>
                      <FontAwesomeIcon icon={faTrash} color={"black"}/>
                    </button>
                  </li>
                ) : null
              ))}
            </ul>
          </>
        ) : null}

        <button type="submit">Продовжити</button>
      </form>
    );
}

export default GameRolesForm;
```

Файл CardsForm.js

```
import React, {useContext, useEffect, useState} from "react";
import CreateGameListItem from "../CreateButton";
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faTrash } from '@fortawesome/free-solid-svg-icons';
import {UserContext} from "../../../context/UserContext";
import axios from "axios";

function CardsForm({ onSubmit, gameId }) {

    const [cards, setCards] = useState([{ description: "", points: 0, typeId: 0, roleIds: []}]);
    const [errors, setErrors] = useState([]);
    const [categories, setCategories] = useState([]);
```

```
const [roles, setRoles] = useState([]);

const { user } = useContext(UserContext);

useEffect(() => {
    axios
        .get(`http://13.51.85.16:8080/api/card-types/${gameId}`, {
            headers: {
                Authorization: `Bearer ${user.token.accessToken}`,
            },
        })
        .then((response) => {
            setCategories(response.data);
        })
        .catch((error) => {
            console.log("Error retrieving categories:", error);
        });

    axios
        .get(`http://13.51.85.16:8080/api/games/${gameId}/roles`, {
            headers: {
                Authorization: `Bearer ${user.token.accessToken}`,
            },
        })
        .then((response) => {
            setRoles(response.data.map((r) => r.id));
        })
        .catch((error) => {
            console.log("Error retrieving roles:", error);
        });
}, [gameId, user.token.accessToken]);

const handleSubmit = (event) => {
    event.preventDefault();

    const isFormValid = cards.every((card) => {
        const cardErrors = [];

        if (card.description.trim() === "") {
            cardErrors.push(...cardErrors,"Завдання є обов'язковим");
        }

        if (card.typeId === 0) {
            cardErrors.push(...cardErrors,"Оберіть категорію");
        }

        setErrors(cardErrors);
        return cardErrors.length === 0;
    });

    if (isFormValid) {
        cards.forEach((card) => card.roleIds = roles);
        onSubmit(cards);
        setCards([]);
    }
};

const handleChange = (index, field, value) => {
    const updatedCards = [...cards];
    updatedCards[index][field] = value;
    setCards(updatedCards);
};
```

```
const handleAddCard = () => {
   setCards([...cards, { description: '', points: 0, typeId: '', roleIds: [] }]);
};

const handleRemoveCard = (index) => {
   const updatedCards = [...cards];
   updatedCards.splice(index, 1);
   setCards(updatedCards);
};

return (
   <div className={"add-cards-page"}>
      <form onSubmit={handleSubmit} className={"w-50 centered card-form"}>
         {cards.map((card, index) => (
            <div key={index} className={"add-card add-item"}>
               <div className="form-field">
                  <label htmlFor={`description-${index}`} className="label">Завдання</label>
                  <textarea
                     id={`description-${index}`}
                     value={card.description}
                     onChange={(e) => handleChange(index, 'description', e.target.value)}
                     className={"card-input"}
                  />
                  {errors[index] && errors[index].description && <div>{errors[index].description}</div>}
               </div>
               <div className="form-field">
                  <label htmlFor={`points-${index}`} className="label">Бали</label>
                  <input
                     type="number"
                     id={`points-${index}`}
                     value={card.points}
                     onChange={(e) => handleChange(index, 'points', e.target.value)}
                     className={"card-input"}
                  />
                  {errors[index] && errors[index].points && <div>{errors[index].points}</div>}
               </div>
               <div className="select-wrapper form-field">
                  <label htmlFor={`category-${index}`} className="label">Категорія</label>
                  <select
                     id={`category-${index}`}
                     value={card.typeId}
                     onChange={(e) => handleChange(index, 'typeId', e.target.value)}
                     className="select-category card-input"
                  >
                     <option key="zero" value="">Оберіть категорію</option>
                     {categories.map((category) => (
                        <option key={category.id} value={category.id}>
                           {category.title}
                        </option>
                     ))}
                  </select>
                  {errors[index] && errors[index].typeId && <div>{errors[index].typeId}</div>}
               </div>

               <button type="button" onClick={() => handleRemoveCard(index)} className={"remove-item"}>
                  <FontAwesomeIcon icon={faTrash} color={"black"}/>
               </button>
            </div>
         ))}

         <CreateGameListItem onSubmit={handleAddCard}/>

         {cards.length > 0 ? (
```

```
                <>
                  <h2>Додані картки</h2>
                  <ul className="cards-added-items">
                    {cards.map((card, index) => (
                      card.description.length > 0 ? (
                        <li key={index} className="card-added" >

                          <div>
                            <div>{card.description}</div>
                          </div>
                          <div>{card.points} балів</div>
                          <button onClick={() => handleRemoveCard(index)} className={"remove-item-on-
card"}>
                            <FontAwesomeIcon icon={faTrash} color={"black"}/>
                          </button>
                        </li>
                      ) : null
                    ))}
                  </ul>
                </>
              ) : null}

              <button type="submit">Продовжити</button>
            </form>
          </div>
        );
}

export default CardsForm;
```

Файл Rocket.js

```
import React from 'react';

const Rocket = ({ progress, name }) => {

  if (progress >= 100) {
    progress = 95;
  }

  const spaceStyle = {
    left: `${progress}%`,
  };

  return (
    <div>
      <div className="rocket" style={spaceStyle}>
        <div>{name}</div>
        <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24" style={{ fill:
'white' }}>
          <path d="M8.011 6.215c-1.711-.009-3.86.918-5.499 2.557-.625.625-1.176 1.355-1.601 2.174 1.479-
1.119 3.057-1.47 4.903-.434.544-1.437 1.27-2.9 2.197-4.297zm9.785 9.773c-1.516.991-3.007 1.706-4.297 2.21
1.036 1.848.686 3.424-.434 4.902.819-.424 1.549-.975 2.175-1.602 1.644-1.642 2.572-3.796 2.556-5.51zm6.152-
15.946c-.412-.028-.816-.042-1.213-.042-8.602 0-13.498 6.558-15.28 11.833l4.728 4.729c5.428-1.946 11.817-6.661
11.817-15.172v-.057c-.002-.42-.019-.851-.052-1.291zm-9.888 9.91c-.391-.391-.391-1.023 0-1.414s1.023-.391
1.414 0 .391 1.023 0 1.414-1.024.39-1.414 0zm2.828-2.828c-.781-.78-.781-2.047 0-2.828s2.048-.781 2.828
0c.781.781.781 2.047 0 2.828s-2.047.781-2.828 0zm-6.534 10.438c-1.492 3.81-5.803 6.208-10.354 6.438.219-
4.289 2.657-8.676 6.64-10.153l.805.806c-4.331 2.755-4.653 5.346-4.665 6.575 1.268-.015 4.054-.344 6.778-
4.464l.796.798z" />
        </svg>
      </div>
    </div>
```

```
    );
};

export default Rocket;
```

Файл Winner.js

```
import './Game.css'
import './Popup.css'
import { Link } from 'react-router-dom';
import React, { useState } from 'react';

const Statistic = ({ moves, winners}) => {
    if (!winners) {
        return <div>Загрузка...</div>;
    }
    return (
        <div>
        <h2>Переможці</h2>
        <table className="centered moves">
            <thead>
            <tr>
                <th>Місце</th>
                <th>Гравець</th>
            </tr>
            </thead>
            <tbody>
            {winners.map((winner, index) => (
                <tr key={index}>
                    <td>{winner.place}</td>
                    <td>{winner.player}</td>
                </tr>
            ))}
            </tbody>
        </table>
    <h2>Ходи</h2>
        <table className="centered moves">
            <thead>
            <tr>
                <th>№</th>
                <th>Завдання</th>
                <th>Роль</th>
                <th>Статус</th>
                <th>Результат</th>
                <th>Бали</th>
            </tr>
            </thead>
            <tbody>
            {moves.map((move, index) => (
                <tr key={index}>
                    <td>{index + 1}</td>
                    <td>{move.description}</td>
                    <td>{move.role}</td>
                    <td>{move.status}</td>
                    <td>{move.points}</td>
                    <td>{move.totalPoints}</td>
                </tr>
            ))}
            </tbody>
        </table>
        </div>
    );
```

```
};

const Winner = ({ player, moves, winners, teamMode, onClose }) => {
  const [showStatistic, setShowStatistic] = useState(false);

  if (!winners) {
    return <div>Загрузка...</div>;
  }

  const generateFileName = () => {
    const date = new Date();
    const year = date.getFullYear();
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const day = String(date.getDate()).padStart(2, '0');
    const timestamp = date.getTime();
    return `game_${year}-${month}-${day}_${timestamp}.csv`;
  };

  const handleShowStatistic = () => {
    setShowStatistic(true);
  };

  const handleDownloadStatistic = () => {
    let winnersSection = '';
    if (!teamMode) {
      const winnersList = winners.map((winner, index) => `${index + 1} - ${winner.player}`).join('\n');
      winnersSection = `Переможці\n${'_'.repeat(30)}\n${winnersList}\n\n`;
    }

    const movesSection = moves.map((move, index) => {
      return `\n${'_'.repeat(32)}\nЗавдання: ${move.description}\nРоль: ${move.role}\nСтатус:
${move.status}\nБали: ${move.points}\nБали всього: ${move.totalPoints}`;
    }).join('\n\n');

    const textReport = `${winnersSection}Ходи\n${'_'.repeat(32)}\n${movesSection}`;

    const downloadLink = document.createElement('a');
    downloadLink.href = window.URL.createObjectURL(new Blob([textReport], { type: 'text/plain;charset=utf-8;'
}));
    downloadLink.download = generateFileName().replace('.csv', '.txt');
    downloadLink.click();
  };

  return (
    <div className="popup">
      <div className="popup-content">
        {teamMode ? (
            <div>
              <h2>Вітаю, командо</h2>
              <h2>Ви досягли своєї мети! Вашу подорож завершено!</h2>
            </div>

        ) : (
          <div>
            <h2>Вітаю, {player}</h2>
            <h2>Ти досяг своєї мети! Твою подорож завершено!</h2>
          </div>
        )}
        <div className={"button-container"}>
        <button className="popup-button" onClick={onClose}>
          Продовжити
        </button>
```

```
            <Link to="/my-account" className="popup-button">
              Завершити гру
            </Link>
            <button className="popup-button" onClick={handleDownloadStatistic}>
              Завантажити результати
            </button>
            <button className="popup-button" onClick={handleShowStatistic}>
              Показати результати
            </button>
            </div>

            {showStatistic && <Statistic moves={moves} winners={winners} />}
          </div>
        </div>
    );
};

export default Winner;
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНИХ ІГРОВИХ ЕЛЕМЕНТІВ НАВЧАННЯ В УКРАЇНСЬКІЙ ШКОЛІ

Програма та методика тестування

КПІ.ІТ-9106.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов Іванова

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Альона Гайова

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб застосунок, призначений для створення ігрових навчальних завдань для використання у навчальних закладах.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– Перевірка дотримання веб застосунком функціональних вимог;

– перевірка безпеки збереження даних;

– перевірка коректної роботи веб-додатку з декількома основними версіями сучасних веб браузерів, таких як Chrome, Opera, Firefox, Edge;

– інтеграція веб застосунку із сторонніми сервісами, такими як Auth0.

– перевірка правильності відображення графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення було використано наступні методи перевірки якості програмного забезпечення:

– статичне тестування – код та документація програми перевіряються на дотримання стандартів якості розробки програмного забезпечення;

– динамічне тестування – проведення перевірки функціоналу застосунку у процесі його використання, відтворення всіх сценаріїв використання, описаних у документації;

– функціональне тестування – перевірка наявності та коректної роботи всіх функцій застосунку, які були винесені як функціональні вимоги до програмного забезпечення;

– інтеграційне тестування – перевірка коректної роботи програми зі сторонніми програмами, у даному випадку – Auth0, який дозволяє аутентифікацію та авторизацію користувачів у застосунку;

– системне тестування – перевірка коректної роботи програми, відсутність критичних помилок, перевірка дотримання всіх нефункціональних вимог до веб застосунку;

– тестування «чорної скриньки» – виконання всіх сценаріїв використання і порівняння очікуваних вихідних результатів із реальними вихідними даними;

– тестування «сірої скриньки» – об'єктом тестування є алгоритм гри, проводиться додаткове і довше тестування різних сценаріїв проходження гри.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Коректна робота веб застосунку та його відповідність очікуванням перевіряється за допомогою декількох видів тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування у різних веб браузерах (Chrome, Opera, Firefox, Edge) ;
- статичне тестування;
- тестування користувацького інтерфейсу для різних розмірів екрану(середній розмір мобільного телефону, планшету, персонального комп’ютеру);
- тестування зручності використання;
- тестування валідації введених даних;
- тестування на переривання операції у разі вводу неправильних даних;
- перевірка відповідності вихідних даних тому, які дані очікуються;
- перевірка коректної роботи алгоритмів бізнес логіки;
- тестування механізму авторизації користувача, який досягається за допомогою стороннього програмного забезпечення;
- інтеграційне тестування.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНИХ ІГРОВИХ ЕЛЕМЕНТІВ НАВЧАННЯ В УКРАЇНСЬКІЙ ШКОЛІ

Керівництво користувача

КПІ.ІТ-9106.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов ІВАНОВА

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Альона ГАЙОВА

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Challenger» - це веб застосунок для додання інтерактивних ігрових елементів навчання у школах. Застосунок представляє собою інструмент для створення ігор, повних карток із завданнями. Кожна гра представляє собою ігрове поле з космічними перегонами, де перед учасниками стоїть мета якнайшвидше дістатись фінішу. Пересування учасників відбувається за рахунок відповідей на завдання карток. Вони це роблять по черзі. Таким чином, застосунок чудово відходить для тестування, викликів до дошки у цікавому форматі. Гра зберігає свій прогрес, тому одну гру можна використовувати, наприклад, протягом декількох уроків або цілого семестру. Тільки користувач, який створює і проводить гру вирішує наскільки швидко вона має бути пройдена, у якому режимі, яка буде нагорода за досягнення фінішу першим, тощо.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність доступу до Інтернету;

– сучасний веб браузер, наприклад Chrome, Firefox, Opera, Edge. У застарілих браузерах (наприклад Internet Explorer) неможливо гарантувати коректне відображення інтерфейсу веб застосунку.

2.2 Завантаження застосунку

Веб застосунок доступний за публічною IP адресою 16.171.16.92.

2.3 Перевірка коректної роботи

При переході за IP адресою у будь-якому сучасному веб браузері (Chrome, Firefox, Opera, Edge) має відкритись веб сторінка, зображення якої представлені нижче. Сторінка має відповідати на всі запити, всі кнопки мають виконувати описані нижче функції.

3 ВИКОНАННЯ ПРОГРАМИ

Для початку роботи із застосунком користувач має перейти за IP адресою 16.171.16.92.

При запуску веб застосунку користувач переходить на домашню сторінку «Challenger», де він може ознайомитись з ідеєю проєкту, прикладами завдань та перевагами системи. Сторінка зображена на рисунку 3.1.

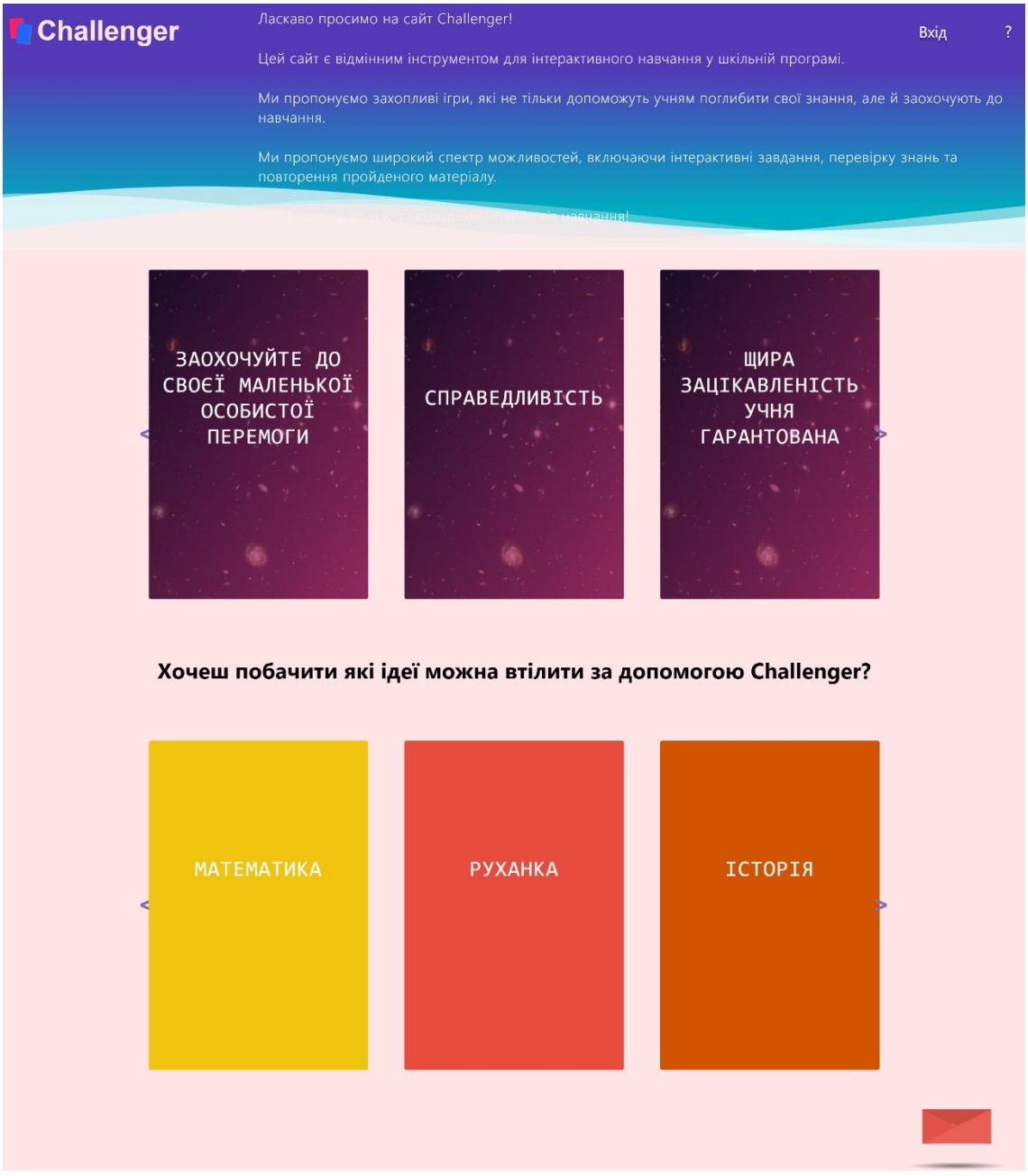


Рисунок 3.1 – Домашня сторінка додатку

З цієї сторінки, як і з будь-якої іншої користувач може керувати застосунком через меню навігації у горі.

Меню складається з наступних посилань:

– Логотип і назва застосунку (домашня сторінка).

– «?» (перехід на сторінку з частими запитаннями та відповідями на них, контактами розробника).

У разі якщо користувач не авторизований:

– «Вхід» (сторінка входу та реєстрації).

У разі якщо користувач авторизований:

– «Мої ігри» (сторінка з усіма створеними цим користувачем іграми).

– «Профіль» (сторінка, де відображаються особисті дані користувача).

– «Вихід» (користувач виходить із свого облікового запису).

Різниця між навігаційною панеллю для неавторизованого та авторизованого користувачів наявна на рисунках 3.3 та 3.4 відповідно.

Рисунок 3.3 – Навігація незареєстрованого користувача

Рисунок 3.4 – Навігація зареєстрованого користувача

Також на кожній сторінці користувач може помітити конверт внизу екрану. Це посилання на сторінку «Зв’яжіться з нами», таке саме як «?».

Для авторизації користувачу потрібно увійти у свій обліковий запис або створити новий. Він може зробити це на сторінці авторизації, для переходу на яку потрібно натиснути на кнопку «Вхід» на меню. Для входу в існуючий акаунт потрібно ввести адресу електронної пошти та пароль і натиснути кнопку «Увійти». Заповнення полів логіну коректними і існуючими у системі даними і натискання на кнопку «Увійти» авторизує

користувача і переведе його на сторінку «Мої ігри». У разі введення некоректних даних цього не відбудеться. Сторінка зображена на рисунку 3.4.

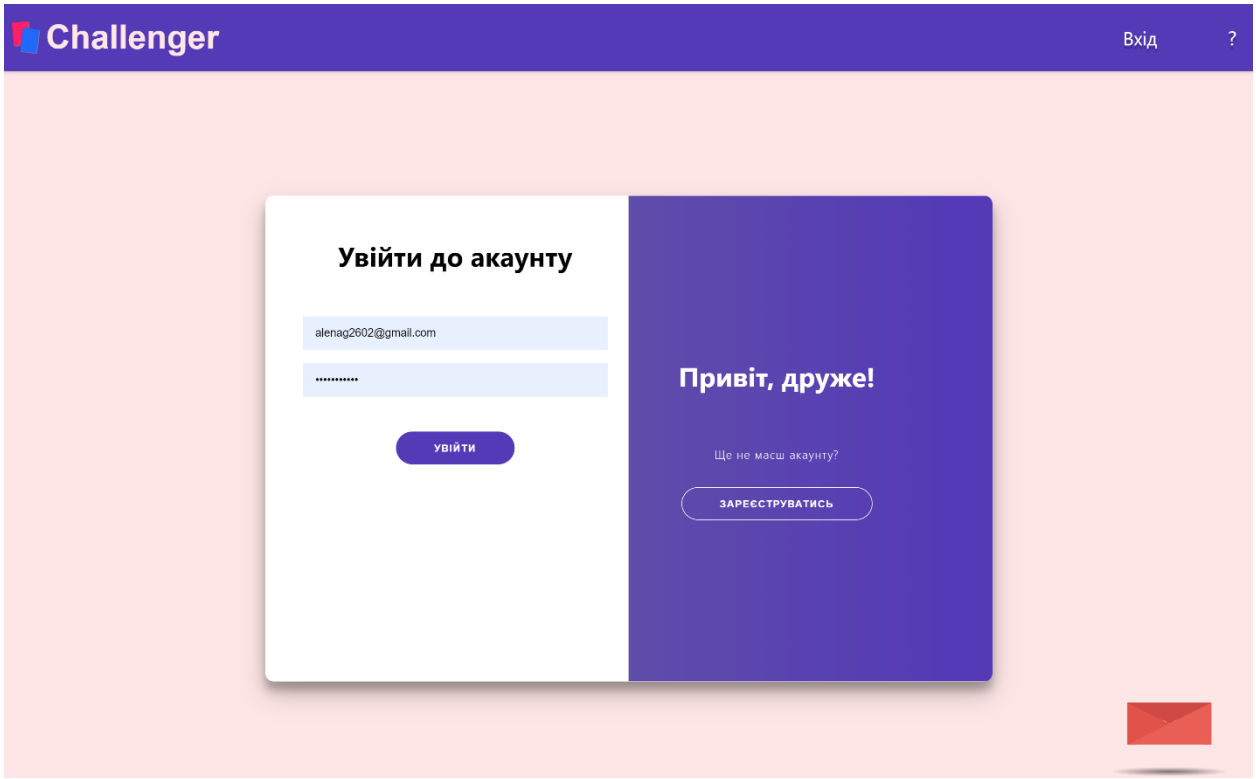


Рисунок 3.5 – Форма логіну

Якщо користувачу потрібно зареєструвати новий обліковий запис, то йому потрібно натиснути на кнопку «Зареєструватись» на фіолетовій панелі. Завдяки кнопкам на фіолетовій панелі відбувається перемикання між формами для входу та реєстрації. На цьому екрані з метою створення акаунту користувач має ввести коректні значення для полів ім’я (мінімум 3 символи), прізвище (мішімум три символи), юзернейм (мінімум 3 символи), електронна пошта (валідна адреса електронної пошти за шаблоном ^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$), пароль (мінімум 1 велика літера, 1 символ, 1 цифра, не менше 8 символів). Якщо всі дані валідні і ця адреса пошти ще не зайнята, натискання на кнопку «Створити» авторизує користувача і переведе його на сторінку «Мої ігри». У разі введення некоректних даних цього не відбудеться. Сторінка зображена на рисунку 3.5.

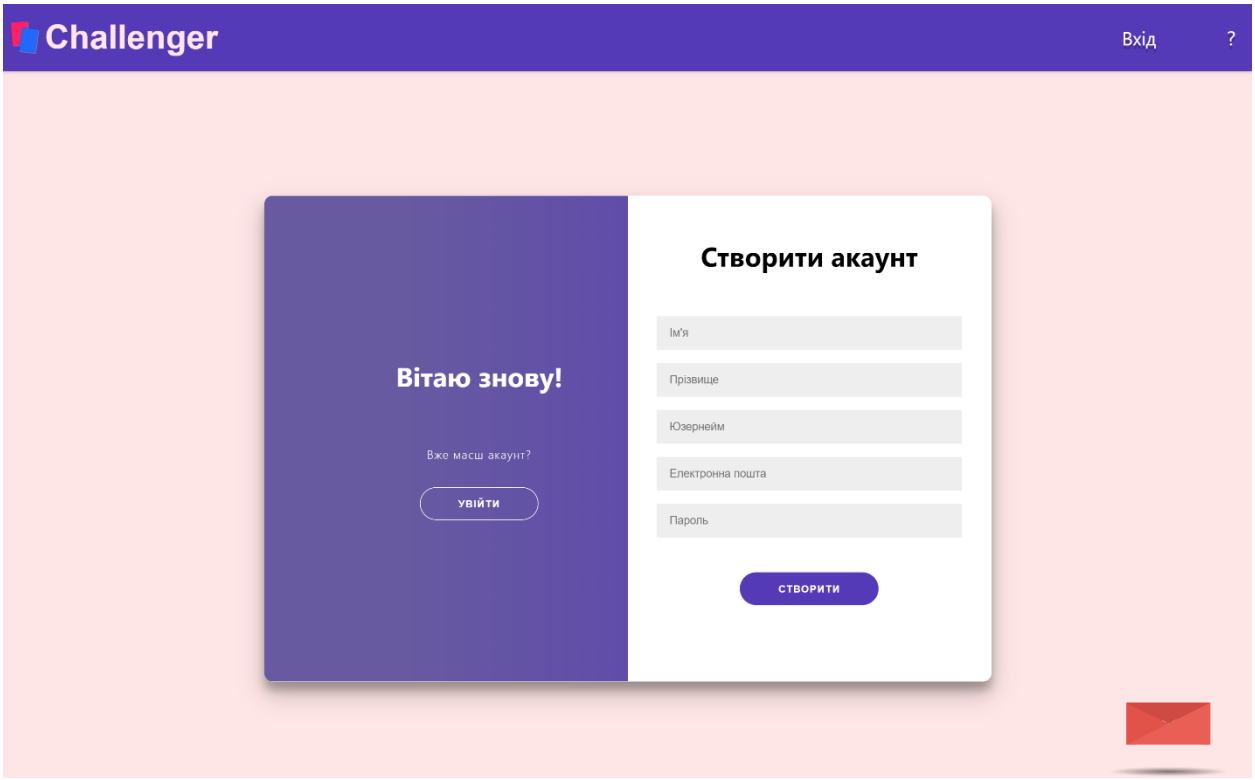


Рисунок 3.6 – Форма реєстрації

Після авторизації користувач опиняється на сторінці «Мої ігри». Це основна сторінка для подальшого використання застосунку. Тут користувач бачить список всіх ігор, які були додані ним. Зеленим кольором відмічені ігри у змагальному режимі та жовтим – у командному.

Різниця між змагальним та командним режимом полягає у розрахунку прогресу гри. Кожна гра має мету (фініш) і кожна картка має кількість балів за її виконання. У змагальному режимі кожен учасник набирає свої власні бали за виконані ним завдання і тим самим просувається до мети. Змагальним він називається не просто так, адже фактично це перегони, де є переможці. У командному ж режимі кожен учасник, який дає відповідь, впливає на спільний рахунок і рухаються до мети усі разом. Це також відображається у графічному інтерфейсі. Детальніше про це буде розказано

при описі процесу гри. Сторінку «Мої ігри» зображено на рисунку 3.7.

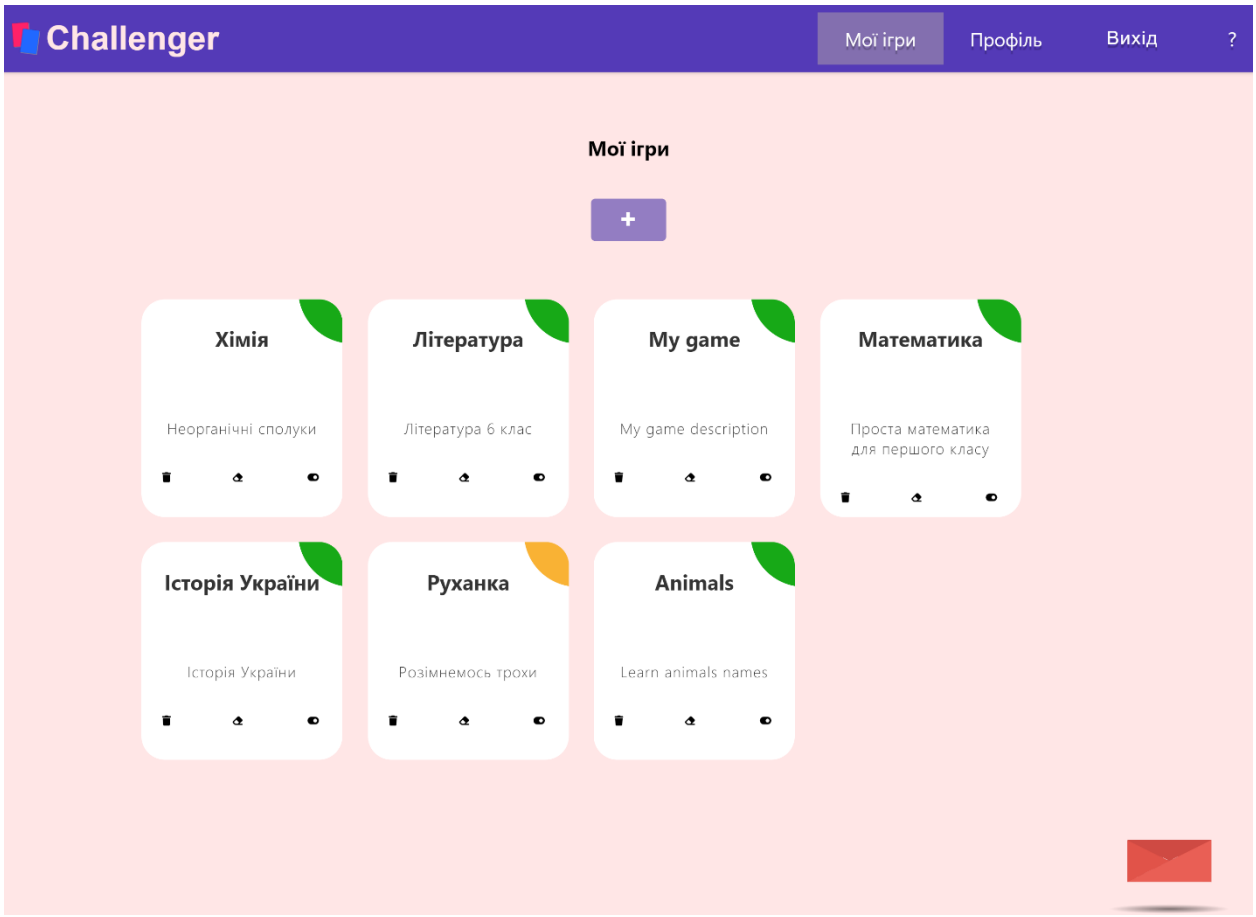


Рисунок 3.7 – Сторінка «Мої ігри»

В першу чергу розглянемо додавання нової гри. Для цього наявна кнопка «+». Після її натискання запускається п’яти етапний процес, який треба пройти повністю, щоб створити працюючу гру. Ці етапи – додавання базової інформації про гру, категорій, ролей, карток, підтвердження створення гри.

На першому екрані відбувається додавання базової інформації про гру: назва (3-20 символів), опис (5-1000 символів), мета гри (число більше 0), чи є гра командною (у вигляді чекбоксу). Для початку спробуйте створити змагальну гру (не відмічений чекбокс). Якщо всі дані є правильними, то натискання на кнопку «Продовжити» переведе користувача на наступний екран. Сторінку додавання гри зображено на рисунку 3.8.

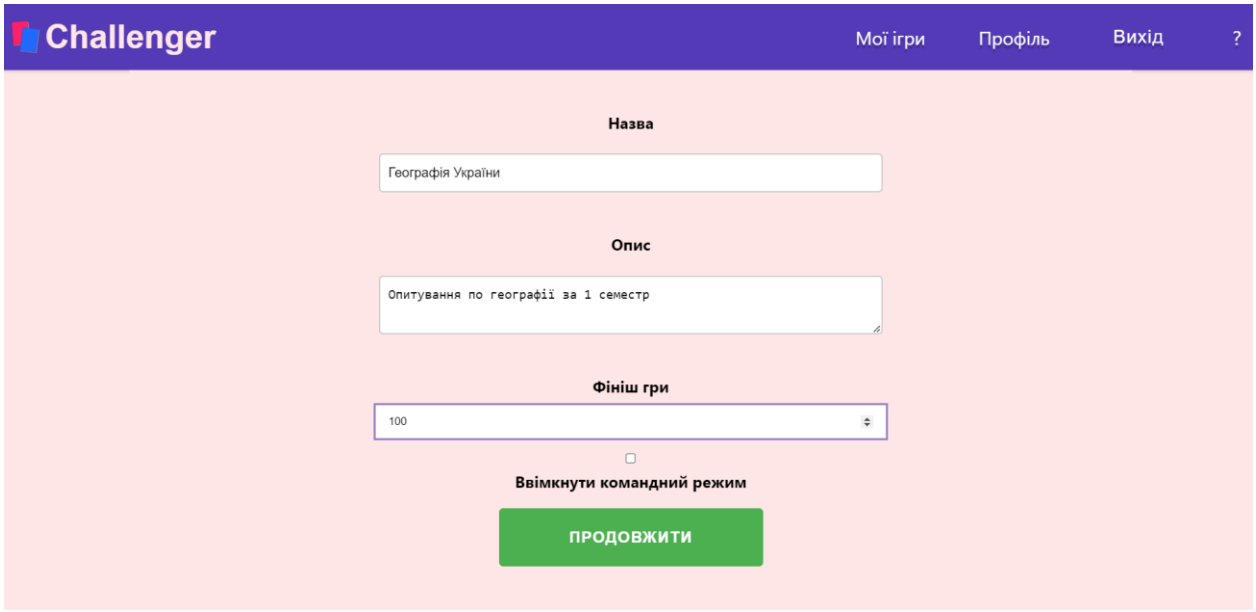


Рисунок 3.8 – Сторінка додавання базової інформації про гру

Наступним іде компонент для створення категорій гри. Категорії потрібні для групування карток за тематикою, їх візуального позначення окремим кольором у інтерфейсі та для розділення карток на ігрові та штрафні. Кожна категорія має складатись з назви (3-50 символів) та кольору. Можливо додати декілька категорій (кнопка +) та переглянути список створених категорій. На кожній створеній категорії є кнопка видалення. Три категорії є доданими автоматично – це штрафні категорії, які будуть використовуватись у якості покарань. Вони мають тематичні космічні назви, але їх можна редагувати, Так само можна редагувати кольори. За замовчуванням це категорії «Корабель зламався!», «Космічний монстр!» і «Метеоритний дощ!». Якщо всі дані є правильними, то натискання на кнопку «Продовжити» переведе користувача на наступний екран. Сторінка додавання категорій зображена на рисунку 3.9.

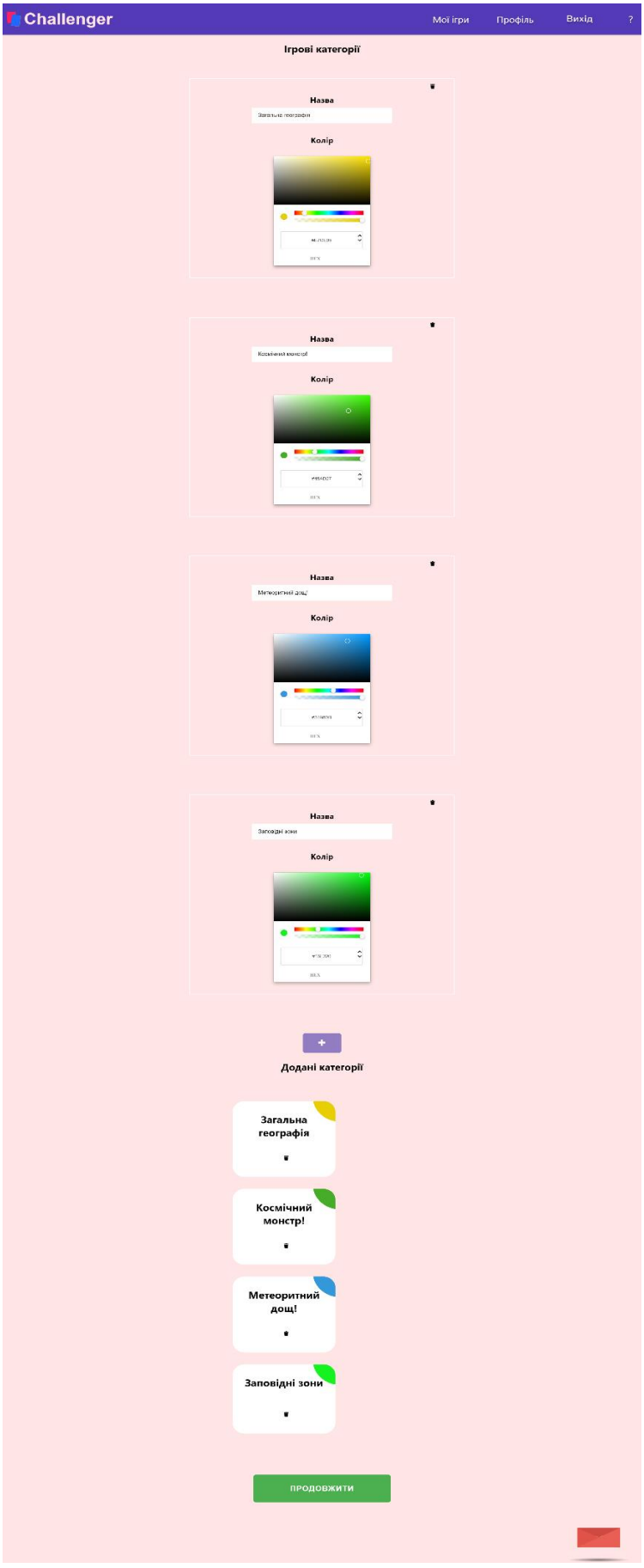


Рисунок 3.9 – Сторінка додавання категорій гри

Наступний іде компонент для додавання ролей гри. Ролі – це учасники гри, які будуть по колу отримувати завдання. Наприклад, заповніть їх іменами учнів класу. У ролі є лише одне поле для створення – назва (3-20 символів). Також є можливість для додання декілької ролей, перегляду та видалення. Якщо всі дані є правильними, то натискання на кнопку «Продовжити» переведе користувача на наступний екран. Сторінка додавання категорій зображена на рисунку 3.10.

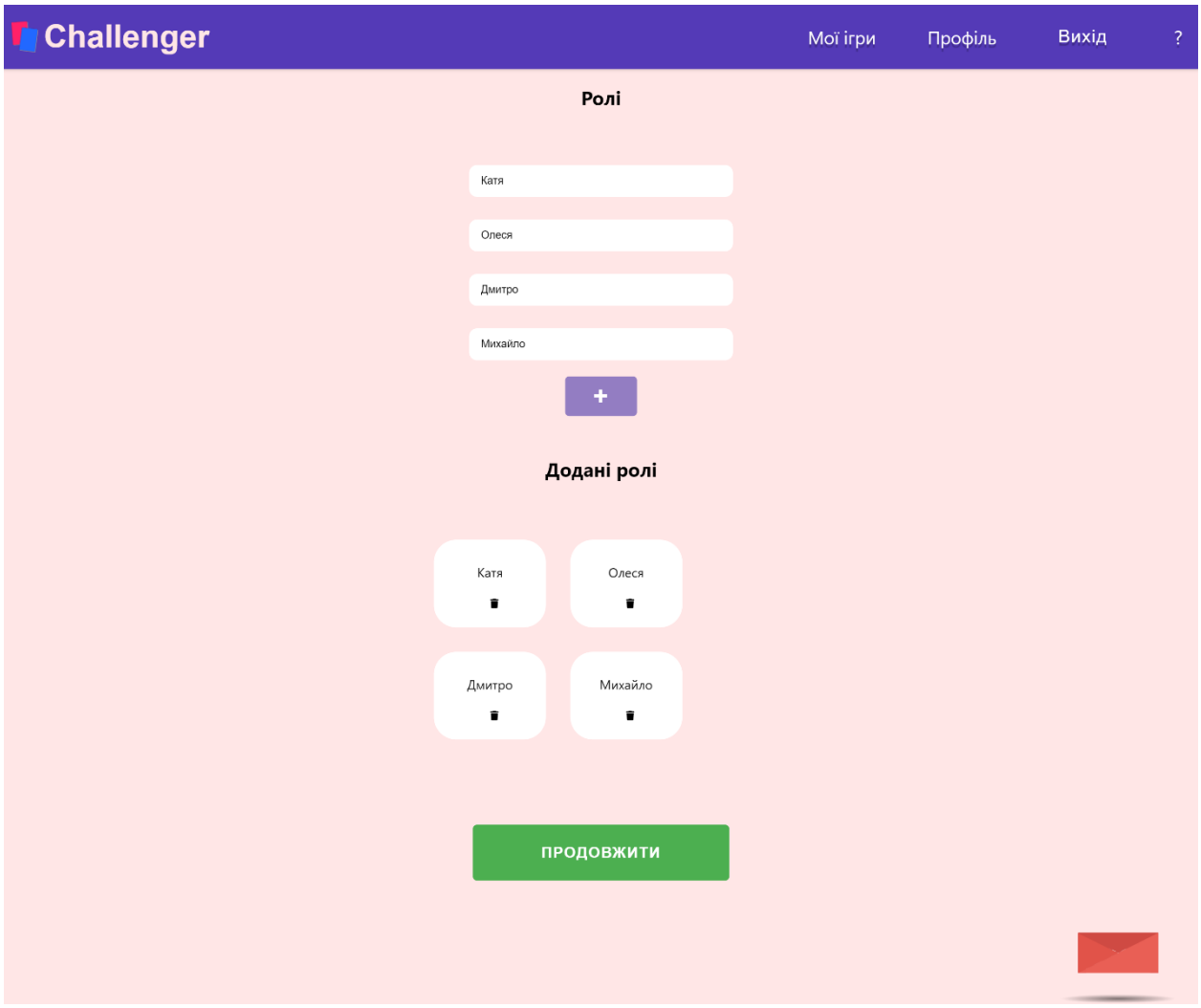


Рисунок 3.10 – Сторінка додавання ролей гри

Останнім етапом створення гри є створення безпосередньо карток із завданнями – це і є завдання, які виконуватимуть учні.

По перше, введіть завдання (5-2000 символів). Прикладами завдань можуть бути як питання із чіткою відповіддю, так і більш абстрактні – їх

перевірка залежить від вчителя. Наприклад, «2+2=?», «Назвіть основну ідею твору «Катерина» Т.Г. Шевченка», «tell about your favourite movie in 5 sentences», тощо. У випадку, якщо ви хочете використати завдання для проходження нового матеріалу, а не тестування, можна написати на картках нову інформацію і просто йти вперед по грі, доки не дійдете до мети.

По друге, оберіть категорію із випадаючого списку. Вибір категорії визначить чи буде картка відображатись у якості звичайного завдання чи виступатиме покаранням.

По третє, введіть кількість балів за картку. Це має бути число, але не обов'язково позитивне. Нуль чи від'ємне число чудово підійдуть для штрафних карток.

Сторінка також дає можливість переглянути створені картки та видалити зайві, як і попередні. Якщо всі дані є правильними, то натискання на кнопку «Продовжити» виведе на екран надпис «Гра була створена успішно». Сторінка створення карток і сторінка підтвердження зображені на рисунках 3.11 та 3.12 відповідно.

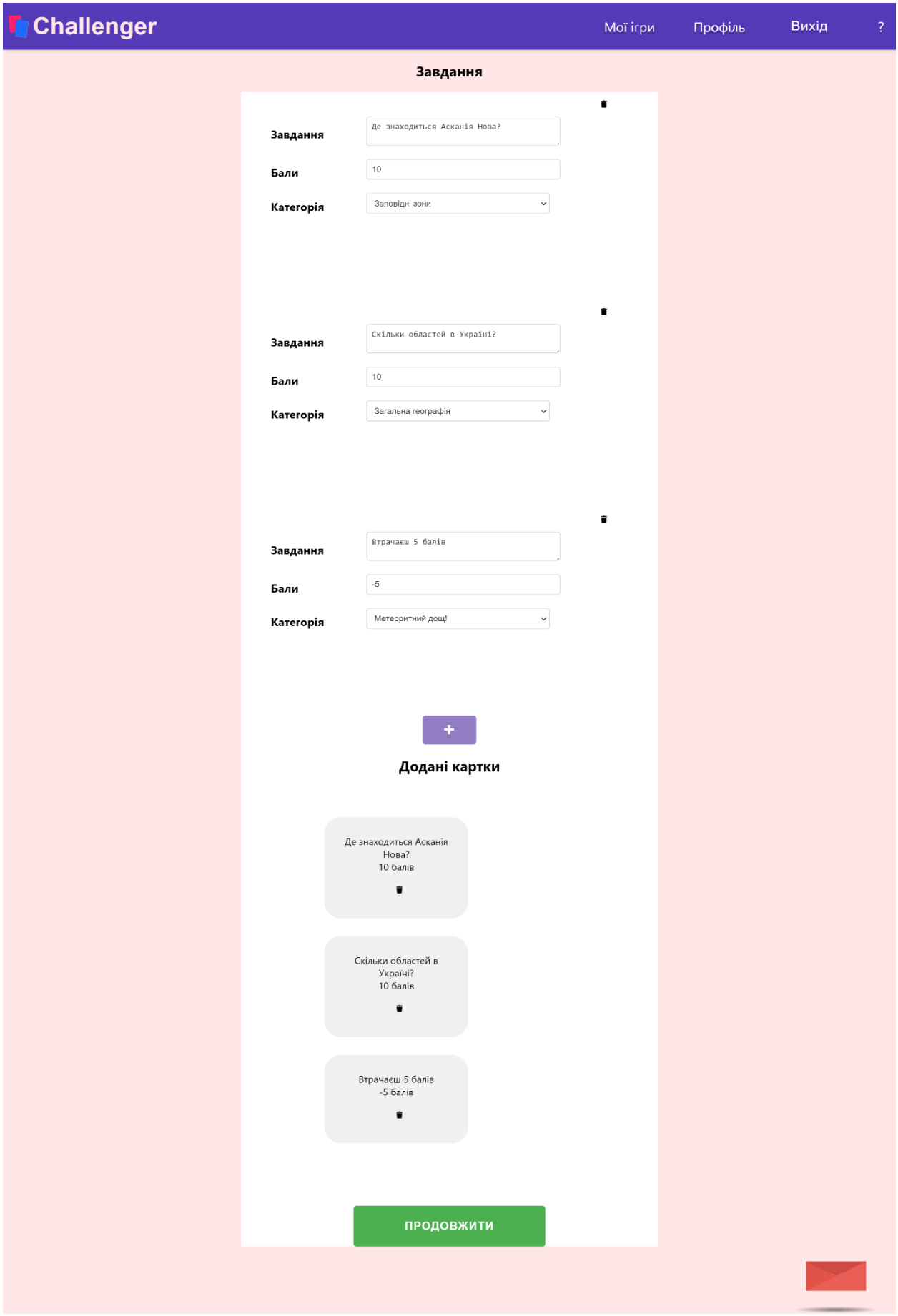


Рисунок 3.11 – Сторінка додавання карток гри

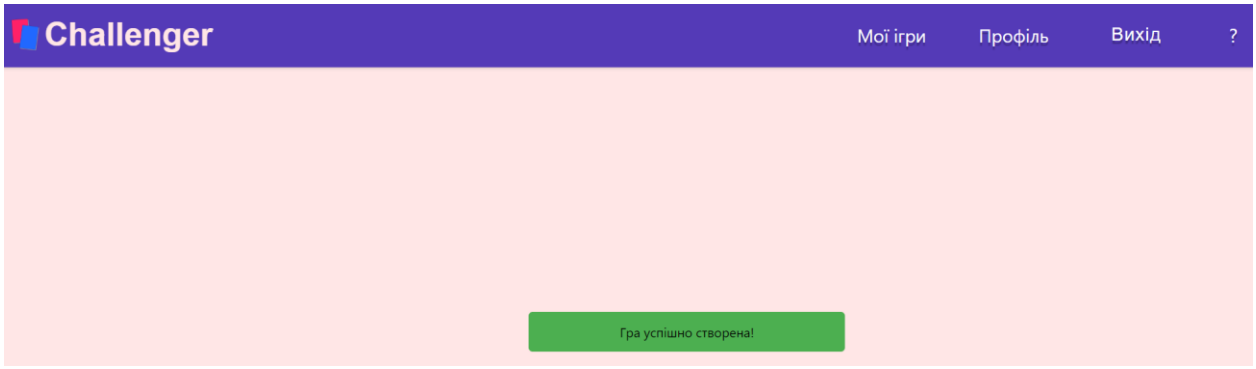


Рисунок 3.12 – Сторінка підтвердження успішного створення гри

Після успішного створення в гру можна зіграти. Натисніть на кнопку «Мої ігри», щоб повернутись до списку, де ви знайдете новостворену гру. Для цього потрібно натиснути на неї. Сторінку гри зображено на рисунку 3.13.

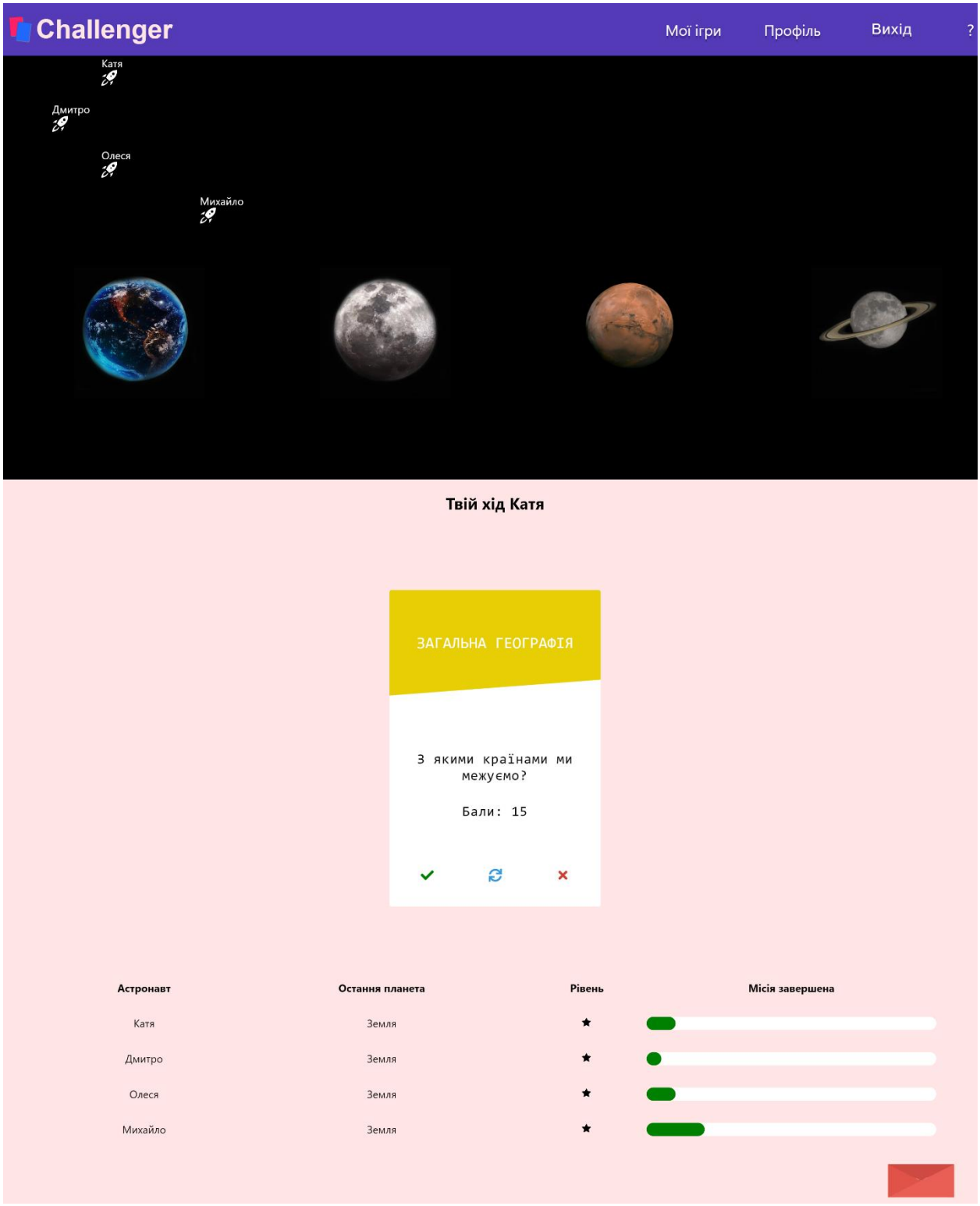


Рисунок 3.13 – Сторінка гри у змагальному режимі

Гра складається з трьох основних елементів – наверху ви побачите ігрове поле – космос із рядом космічних тіл (Земля, Місяць, Марс і Сатурн), де Земля є точкою відправлення, а Сатурн – метою гри. У змагальному режимі ви побачите по одній ракеті для кожної ролі, які будуть просуватись

поміж планет по мірі прогресу учасника. Як не складно здогадатись, хто першим дістанеться Сатурну, той і перемагає.

Другим елементом гри є завдання. Позначена поточна роль і картка, на якій написана назва категорії. При наведені курсора вона перевернеться і покаже завдання, бали та елементи управління. Якщо учень надає правильну відповідь, вчитель має натиснути зелену кнопку, яка нарахує учню бали і переведе хід на наступну роль. Блакитна кнопка призначена для ситуацій, коли хочеться пропустити якесь завдання (наприклад для нього немає часу чи певного обладнання). Це ніяк не вплине на бали і просто видасть нвоу картку тому самому учаснику. А ось у разі неправильної відповіді натискається червона кнопка. Важливо відмітити, що це не вирахує і не додасть бали до рахунку учасника. Вона теж поверне завдання для того самого поточного учасника, але вже із штрафної категорії. Саме бали за штрафну картку нарахуються учаснику, тому якщо механізм покарання спирається на віднімання балів, то важливо прописати це при створенні карток. Після відповіді на штрафну категорію, хід перейде до наступного учасника.

Третім елементом є таблиця відслідковування прогресу. Для кожного учасника прописано рівень, якого він досягнув, планета, до якої він долетів, і показано прогрес його шляху до Марсу на зручній шкалі.

Гру можна призупинити в будь який момент – просто перейшовши на іншу сторінку. З прогресом гри нічого не станеться, якщо його не стерти вручну. Після доходження кожного з учасників до фінішу буде з’являтись вікно з варіантами подальших дій:

– Продовжити: просто закриє вікно і дасть змогу грати далі.

– Завершити гру: вийде на сторінку «Мої ігри», але не стирає прогрес.

– Показати результати: виведе таблицю з переможцями і всіма ходами у тому самому вікні.

– Завантажити результати: виведе таблицю з переможцями і всіма ходами у текстовий файл, який завантажить на девайс. Файл буде позначено датою.

Теоретично, гру можна продовжувати після того як всі учасники дійшли до фінішу, але це не має сенсу. За потреби, гру можна просто почати спочатку. Зображення вікна переможця із виведеними на ньому результатами зображено на рисунку 3.14.

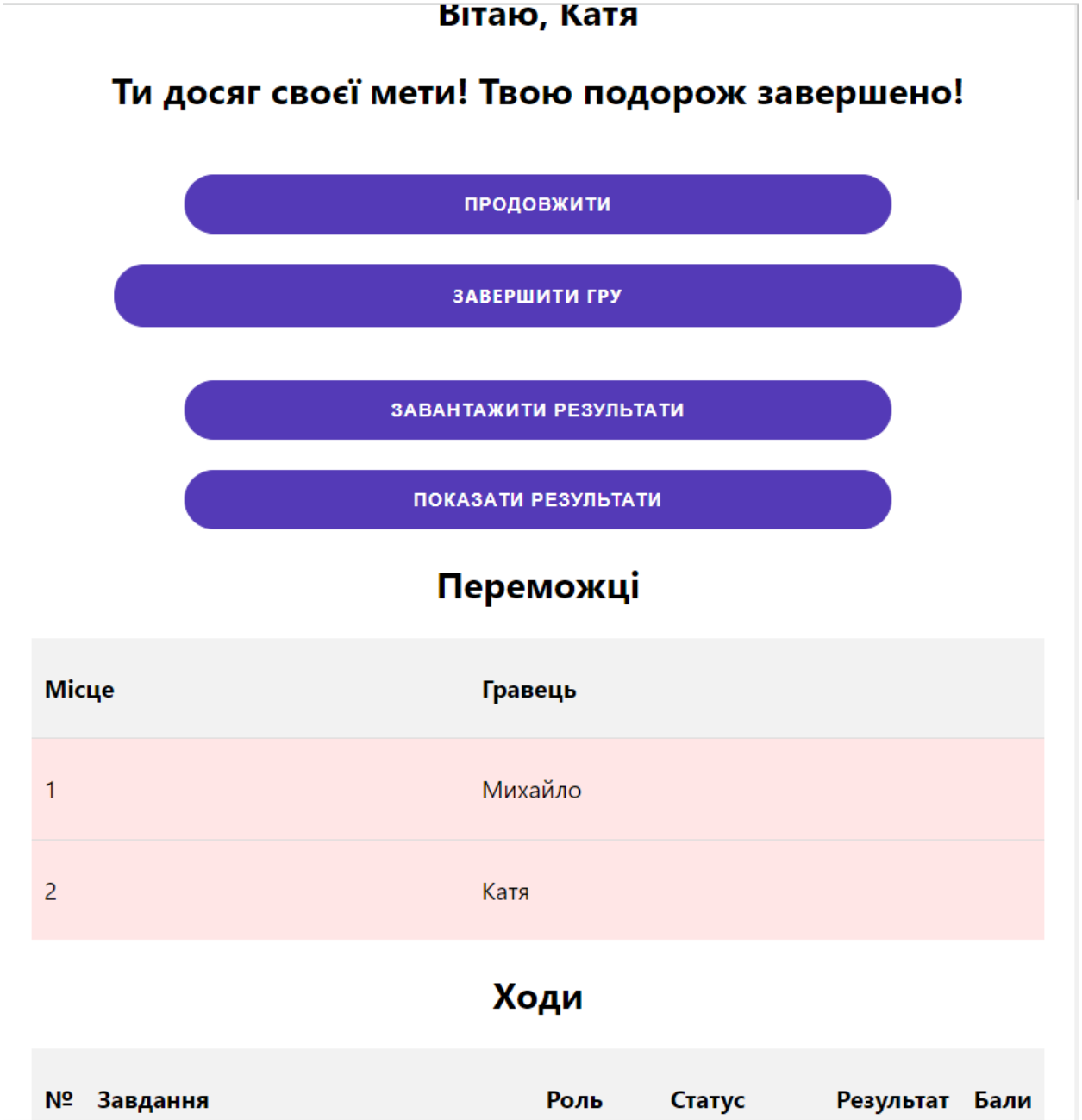


Рисунок 3.14 – Вікно завершення гри

Для того, щоб дозволити учням попрацювати як одній команді не треба створювати нову гру. Режим гри можна переключити. Важливо пам’ятати, що перемикання режиму все ще не стирає прогресу, так що якщо хочеться початку гру як нову – натисніть кнопку у вигляді ластику на грі. Після цього у спливаючому вікні, зображеному на рисунку 3.15, підтвердіть свою дію.

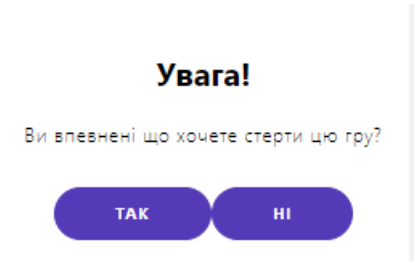


Рисунок 3.15 – Вікно для підтвердження видалення прогресу гри

Після оновлення гри натисніть на кнопку перемикача, підтвердіть свою дію. Після цього, якщо ви зайдете в гру, ви побачите, що вона змінилась на командний режим. Сторінка гри у командному режимі трохи відрізняється. Вона зображена на рисунку 3.16.

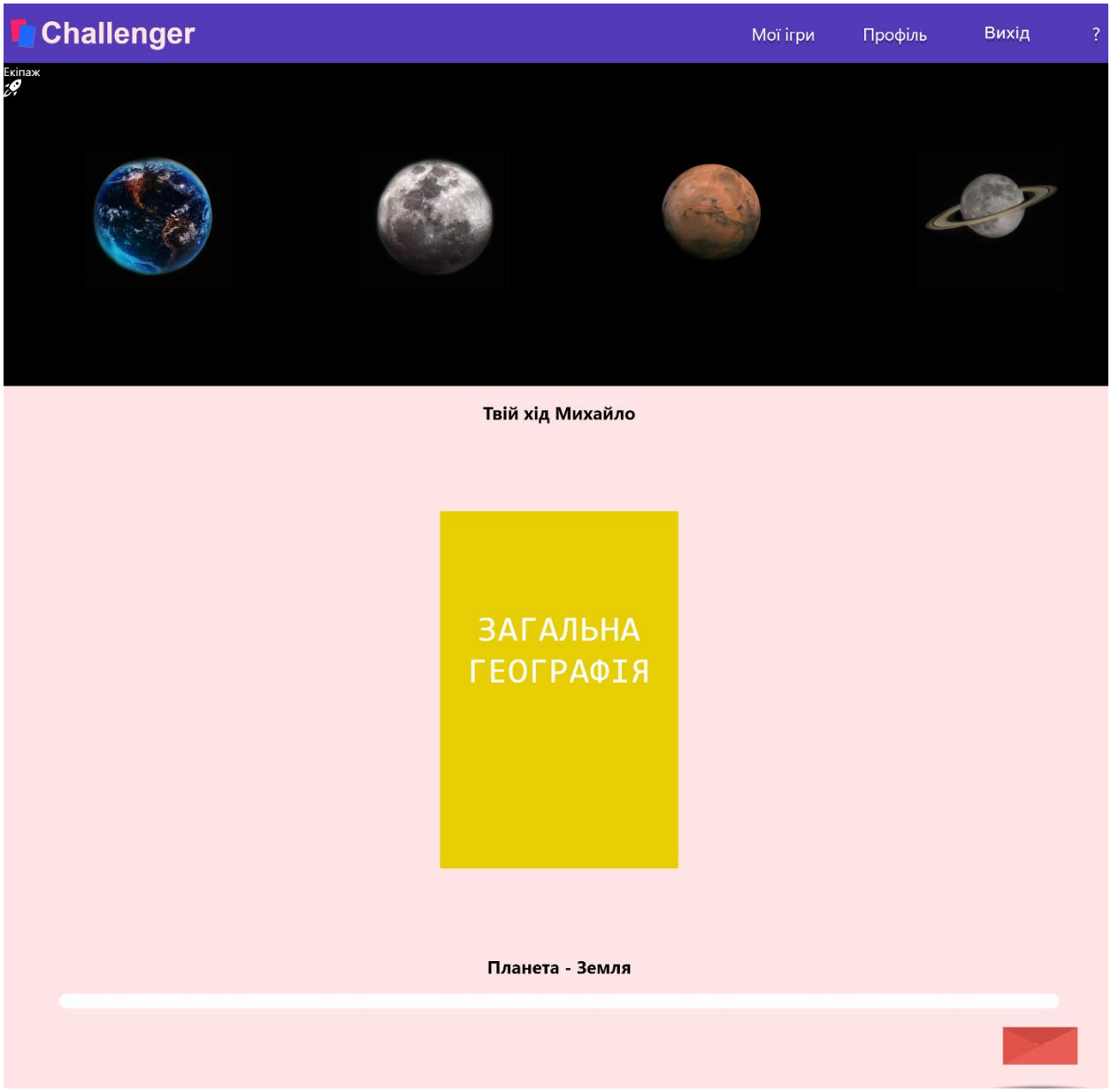


Рисунок 3.16 – Сторінка гри у командному режимі

Відмінності у командному режимі :

– один екіпаж замість багатьох ракет;

– всі бали складаються разом для досягнення мети;

– замість таблиці є лише одна шкала прогреса із планетою;

– у разі виведення чи завантаження результатів не вказуються переможці.

Також, гра може бути видалена натисканням кнопки із зображенням сміттєвого баку. Ця дія також потребує підтвердження.

Це вся функціональність яка стосується ігор у застосунку. Застосунок також має сторінки «Профіль» і «Зв’яжіться з нами», які доступні за кнопками з меню чи конвертом внизу сторінок.

На сторінці профілю можна знайти інформацію про акаунт (нікнейм, ім’я, прізвище та адресу електронної пошти). Її зображено на рисунку 3.14.

Сторінка для зворотного зв’язку містить контакти і поширені запитання та відповіді на них. Вона зображена на рисунку 3.15.

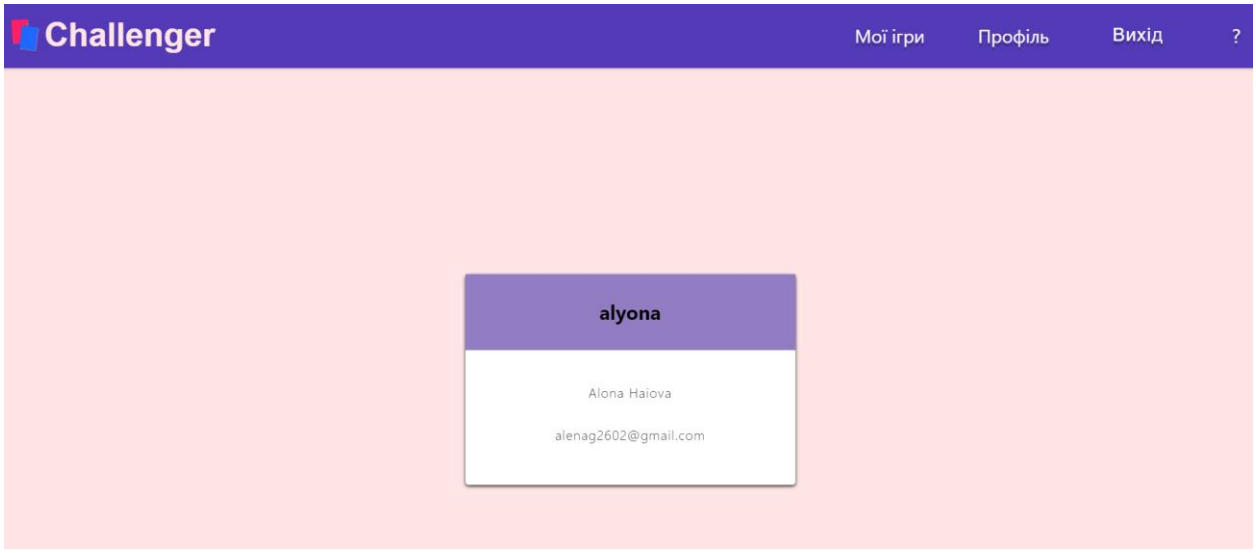


Рисунок 3.14 – Сторінка профілю

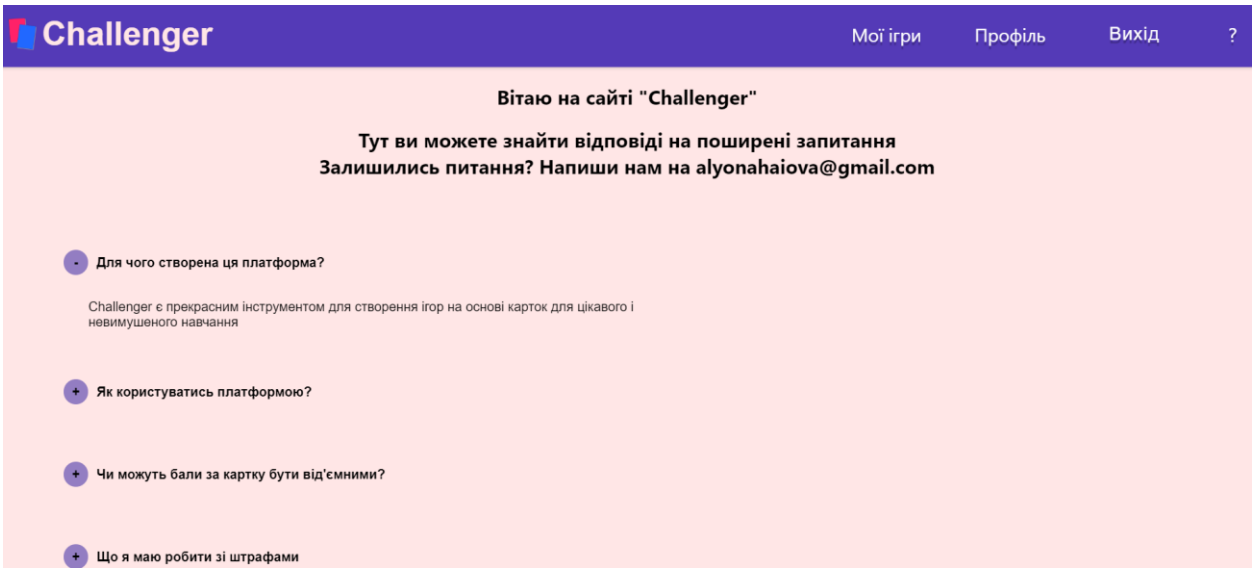


Рисунок 3.15 – Сторінка «Зв’яжіться з нами»

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНИХ ІГРОВИХ ЕЛЕМЕНТІВ НАВЧАННЯ В УКРАЇНСЬКІЙ ШКОЛІ

Графічний матеріал

КПІ.ІТ-9106.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов ІВАНОВА

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

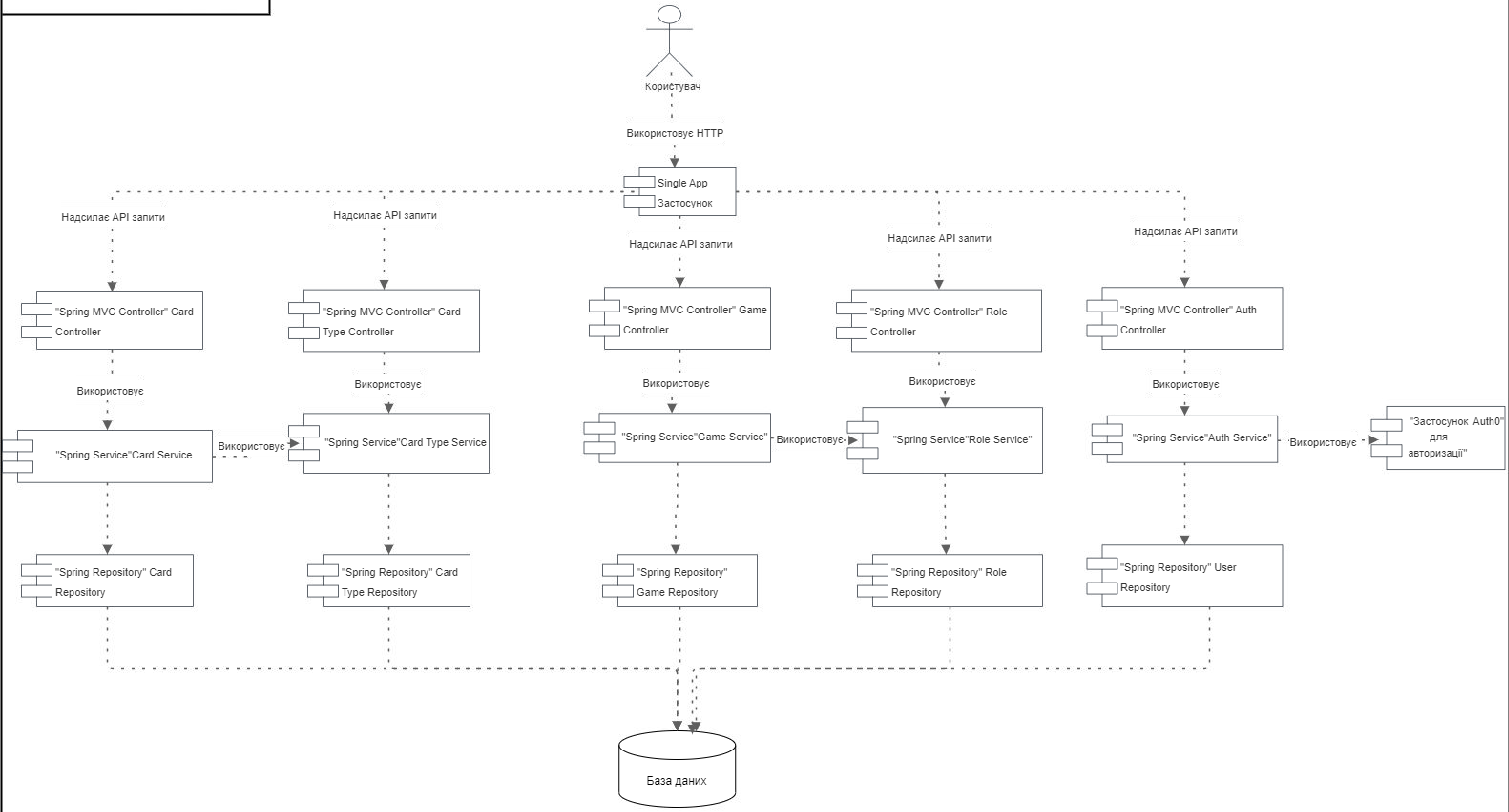
Виконавець:
_____ Альона ГАЙОВА

Київ – 2023

					КПІ.ІТ-91.045440.06.99.СС	
						Літера / Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Гайова А.Д.			Діаграма варіантів використань	
Перевірив		Іванова Л.М.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Головченко М.М.			Програмне забезпечення для створення інтерактивних ігрових елементів навчання в українській школі	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91
Затвердив		Іванова Л.М.				

КПІ.ІТ-91.045440.06.99.СБД

					КПІ.ІТ-91.045440.06.99.СБД	
Зм.	Арк.	№ документа	Підпис	Дата	Схема бази даних	Літера / Маса / Масштаб
Розробив		Гайова А.Д.				
Перевірив		Іванова Л.М.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Головченко М.М.			Програмне забезпечення для створення інтерактивних ігрових елементів навчання в українській школі	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-91
Затвердив		Іванова Л.М.				



					КПІ.ІТ-91.045440.06.99.СС
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Гайова А.Д.			*Діаграма компонентів*
Перевірив		Іванова Л.М.			
Т. кон.					
Н. кон.		Головченко М.М.			*Програмне забезпечення для створення інтерактивних ігрових елементів навчання в українській школі*
Затвердив		Іванова Л.М.			

Літера	Маса	Масштаб
Аркуш	Аркушів	

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІТ-91