МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБСАЙТУ ДЛЯ КОЛЕКЦІОНУВАННЯ ГРАЛЬНИХ КАРТОК ЗАСОБАМИ MERN, REACT ТА POKEAPI»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
І.С. Тюшняков
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD Чопорова О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри комп'ютерних наук, професор, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Тюшнякову Іллі Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебсайту для колекціонування гральних карток засобами Mern, React та PokeApi

керівник роботи Чопорова Оксана Володимирівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та розробка вебдодатку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	11.03.2024	
2.	Збір вихідних даних.	15.04.2024	
3.	Обробка методичних та теоретичних	31.05.2024	
	джерел.		
4.	Розробка першого та другого розділу.	26.08.2024	
5.	Розробка третього розділу.	01.11.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	13.12.2024	

Студент _____ І.С. Тюшняков
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Чопорова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебсайту для колекціонування гральних карток засобами Mern, React та PokeApi»: 93 с., 34 рис., 10 джерел, 2 додатки.

ВЕБДОДАТОК, ВЕБСАЙТ, CARD GAME, EXPRESS, JAVASCRIPT, MERN, MONGODB, POKEAPI, REACT, VITE.

Об’єкт дослідження – розробка вебсайту для колекціонування гральних карток та простої гри з їх використанням.

Мета роботи: розробка вебсайту засобами MERN.

Метод дослідження – методи програмної інженерії: об’єктно-орієнтоване програмування, UML-методологія, тестування.

У роботі проведено дослідження процесу розробки вебсайту з використанням MERN та PokeApi. Наведено аналіз та аргументацію вибору інструментів. Визначено вимоги до застосунку, побудовано необхідні діаграми та реалізовано вебсайт.

В роботі детально описано процес розробки вебсайту з використанням технологій MERN та реалізацію всіх функцій зазначених у вимогах, таких як реєстрація та авторизація користувачів, робота з базою даних, використання PokeApi для створення карток, реалізація магазину карток та взаємодії користувачів, а також простої вебгри з використанням карток.

Результати роботи можуть бути використані для дослідження розробки вебсайтів з використанням MERN, а також для створення карткових ігор для використання в інших вебзастосунках.

SUMMARY

Master’s qualifying paper “Development of a Website for Collecting Playing Cards Using Mern, React and PokeApi”: 93 pages, 34 figures, 10 references, 2 supplements.

WEB APPLICATION, WEBSITE, CARD GAME, EXPRESS, JAVASCRIPT, MERN, MONGODB, POKEAPI, REACT, VITE.

The object of the study is development web application for collecting trading cards and simple game using it.

The aim of the study is to develop a web application using MERN.

The methods of research are methods of software engineering: object-oriented programming, UML-methodology, testing.

Work contains analysis of web application development process using MERN and PokeApi. Given argumentation of instrument choice. Defined requirements for application, built corresponding diagrams and developed web application.

Work contains description of web application development process using MERN and realization of all functions mentioned in requirements, such as registration and authorization of users, work with database, using PokeApi for creation of cards, implementation of cards shop and user interaction, as well as simple web game with usage of cards.

Results of work can be used for research of development of web applications using MERN, as well as for development of card games for other web applications.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз вимог до застосунку 8
1.1 Актуальність дослідження 8
1.2 Аналіз засобів розробки вебдодатків 12
1.3 Аналіз функціональних вимог 14
1.4 Висновки до розділу 1 16
2 Проєктування застосунку 17
2.1 Проєктування структури застосунку 17
2.2 Висновки до розділу 2 19
3 Програмна реалізація 20
3.1 Програмна реалізація додатку 20
3.2 Висновки до розділу 3 40
Висновки 41
Перелік посилань..... 42
Додаток А Код серверної частини додатку 43
Додаток Б Код фронтенду додатку..... 58

ВСТУП

Колекціонування гральних карток завжди було поширеним заняттям серед багатьох людей, однак з розповсюдженням технологій та інтернету фізичні версії ігор втрачають свою популярність, на відміну від онлайн розваг.

Отже, актуальною задачею є розробка вебсайту для колекціонування гральних карток.

Метою кваліфікаційної роботи є розробка вебсайту для колекціонування гральних карток та простої гри з їх використанням.

Задачі, які необхідно розв’язати для досягнення поставленої мети:

– визначити методи дослідження та набір технологій для розробки;
– провести порівняльний аналіз існуючих аналогів;
– визначити основні функції розроблюваного додатку;
– визначити список вимог до розроблюваного додатку;
– розробити необхідні діаграми;
– протестувати розроблений додаток та виправити помилки.

Об’єкт дослідження – розробка вебсайту для колекціонування гральних карток та простої гри з їх використанням.

Предмет дослідження – процес розробки вебсайту з використанням MERN.

Метод дослідження – методи програмної інженерії: об’єктно-орієнтоване програмування, UML-методологія, тестування.

Структурно дипломна робота складається з таких компонентів: вступ, 3 розділи, висновки, 2 додатки, перелік посилань з 10 найменувань.

В першому розділі наведено аналіз аналогів розробки, порівняння та обґрунтування вибору стеку технологій, визначено функціональні вимоги та можливості користувача. В другому розділі розроблено необхідні діаграми та спроєктовано додаток, наведено пояснення змісту діаграм. В третьому розділі описано програмну реалізацію застосунку.

1 АНАЛІЗ ВИМОГ ДО ЗАСТОСУНКУ

1.1 Актуальність дослідження

В першу чергу варто зазначити важливість та актуальність дослідження для користувачів – колекціонування грального карток на різноманітні тематики протягом багатьох років було дуже популярним, а також прибутковим для їх виробників. Однак в останні роки збирання фізичних варіантів карток стало неактуальним, оскільки їх важко знайти, ціна часто є дуже високою, а знайти поруч опонентів для гри, що збирають такі самі картки та готові грати виявляється ще важчим завданням. Саме тому інтернет-версія такого популярного заняття є дуже актуальною та навіть необхідною в даний час – люди зможуть збирати власні колекції, отримувати задоволення від отримання нових карток, обмінюватися та грати один з одним.

В результаті аналізу наукової літератури за темою розробки вебдодатків, як наприклад джерела [1-8] (переклад з використанням словника [9]), можна побачити, що переважна більшість продемонстрованих та описаних в них програмних застосунків є соціальними додатками, інакше кажучи, соціальними мережами. Розроблюваний додаток для колекціонування гральних карток також частково є соціальною мережею із загальними притаманними цьому типу додатків функціями, такими як реєстрація, авторизація, сторінка користувача та ін. Однак, крім соціальної складової, додаток також буде мати ігрову складову, а саме просту карткову вебгру, аналогів або схожих технологій для якої не було знайдено в процесі аналізу відомої літератури.

Серед програмних аналогів можна виділити кілька продуктів найбільш схожих за темою та функціоналом. Найближчим за ідеєю та найпопулярнішим представником є «Pokemon Trading Card Game», показана на рисунку 1.1. Це класична гра, що дозволяє колекціонувати та грати картками покемонів з

іншими гравцями або комп’ютером, однак є певні складнощі з їх колекціонуванням. Для отримання карток необхідно або купувати їх за реальну валюту, або витрачати велику кількість часу на те, щоб заробити ігрову валюту та придбати ігрові картки, що є великим недоліком. Окрім цього, цифрова версія гри не надає гравцям можливості обмінюватися картками, оскільки розробники вважають, що це ускладнює ігрову економіку. Не зважаючи на це, гравці як і раніше можуть обмінюватися фізичними версіями карток, однак зараз більшість гравців не обмінює, а просто продає та купує картки. Також, ця гра не має браузерної версії, лише десктоп та мобільну, що в купі з іншими недоліками робить гру менш зручною та доступною для користувачів. Фізична версія цієї гри була найпопулярнішою з ігор з колекційними картками до поширення інтернету, тому розробники намагалися зробити комп’ютерну версію своєї гри, однак вона не стала настільки популярною як оригінальна гра. До того ж, «Pokemon Trading Card Game» має дуже складні правила які потребують певного часу на вивчення, а також величезну кількість різних унікальних карток з такими особливостями, що також вимагають багато часу на опанування. Це відштовхує нових гравців через велику складність, а програмна реалізація подібного ігрового процесу вимагає величезних ресурсів.



Рисунок 1.1 – «Pokemon Trading Card Game»

Іншим аналоговим продуктом є відома гра «Hearthstone» (рис. 1.2), яка також дозволяє колекціонувати картки та грати з іншими користувачами та з комп’ютером. На відміну від «Pokemon Trading Card Game», ця гра не має фізичних версій карток та є повністю цифровою. Картки можна купувати за ігрову та реальну валюту, через що виникає класична суперечка між користувачами, предметом якої є можливість грати краще та частіше перемагати, якщо ти витратив більше грошей та отримав кращі картки, що також відштовхує багатьох гравців. Також виникає складність колекціонування карток та створення хорошої колоди через неможливість придбати конкретні картки – гравець може купувати лише закриті набори та не знає, що саме отримає в результаті. Окрім цього, як і в попередньому додатку, в цій грі неможливо обмінюватися картками з іншими гравцями. Недоліками «Hearthstone» є також відсутність версії для браузерів, та в результаті менша доступність гри для великої аудиторії гравців.

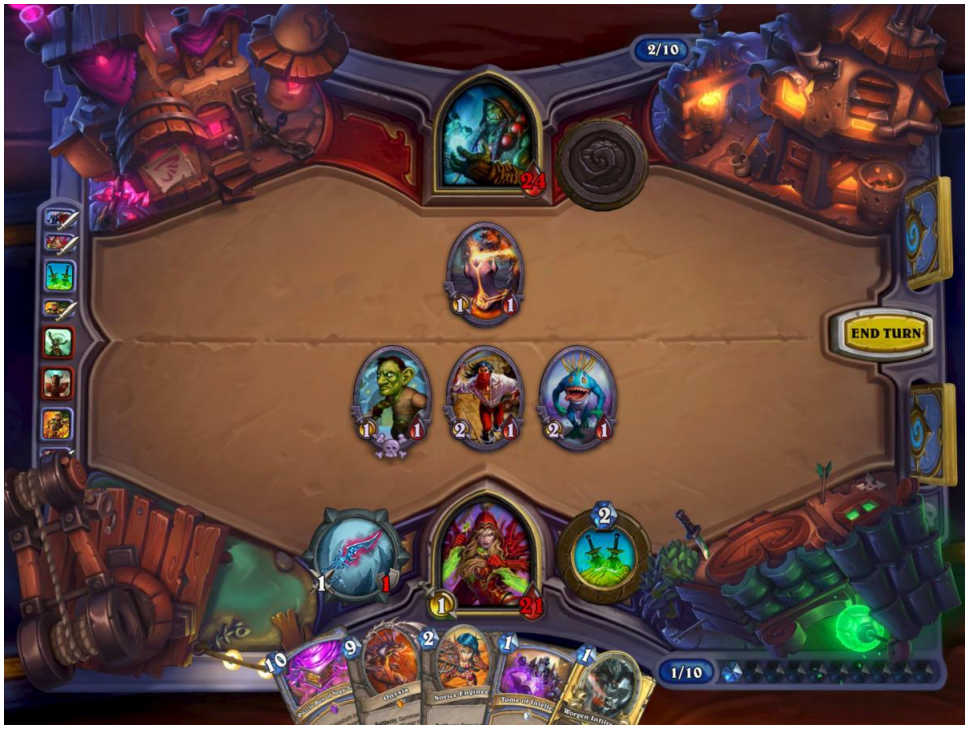


Рисунок 1.2 – «Hearthstone»

Серед інших ігрових додатків є дуже багато представників карткових ігор, однак вони мають лише механіки карткової гри, а не надають повноцінну

можливість колекціонування ігрових карток. Серед подібних програмних продуктів можна виділити «Slay the Spire», один з найвідоміших представників карткових ігор, та величезна кількість інших подібних ігор є дуже схожими за всіма параметрами саме на неї. Ця гра є представником жанру Roguelike, тому ігровий процес розділяється ігровими сесіями, що зазвичай тривають до години реального часу. Гравець збирає власну колоду карток під час сесії, однак ця колода не виходить за її межі, а отже не зберігається до наступної ігрової сесії. Картки використовуються для боротьби з ворогами та просування вперед. Важливим аспектом є те, що гра є повністю одиночною, тому зіграти з іншими гравцями неможливо. Також значною відмінністю від інших аналогів є неможливість ані придбати, ані обмінятися картками. Враховуючи всі вище зазначені відмінності, не є доцільним вважати «Slay the Spire» (рис. 1.3) та інші схожі ігри аналоговими продуктами.



Рисунок 1.3 – «Slay the Spire»

Також єдиним вебдодатком за схожою тематикою є вебсайт «CardMarket», зображений на рисунку 1.4, який на відміну від інших аналогів є лише інтернет-платформою для продажу фізичних версій ігрових карток, та

навіть не має ігрового аспекту, тому не може вважатися аналогом розроблюваного додатку, що більше орієнтується на ігровий процес та колекціонування карток.

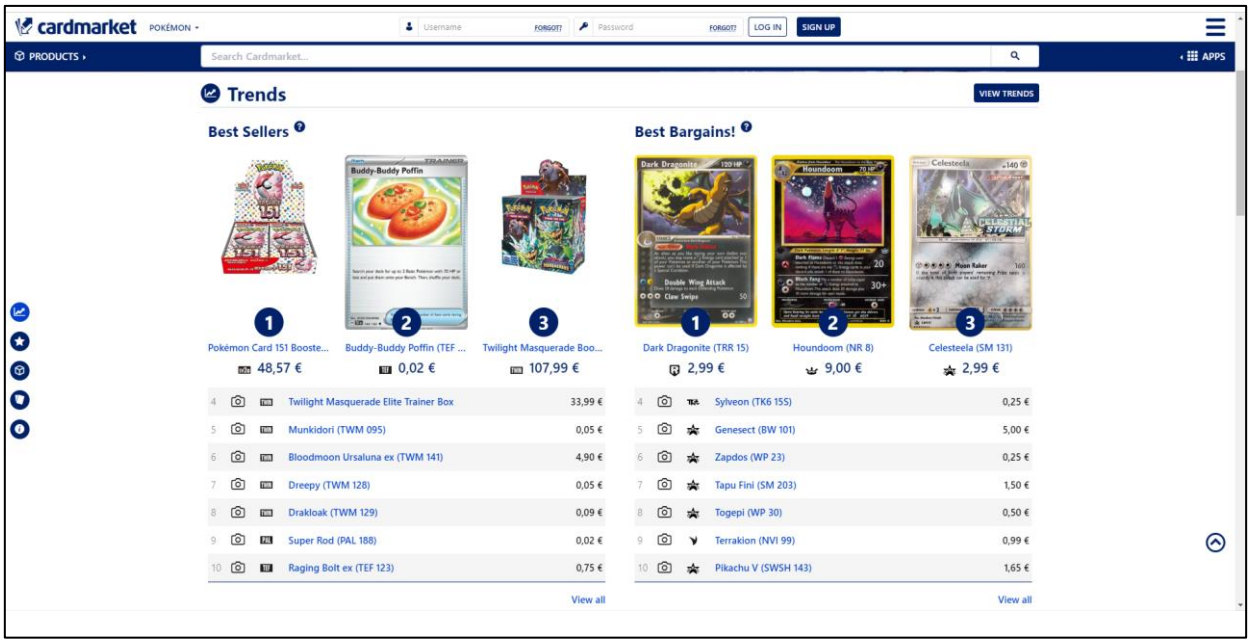


Рисунок 1.4 – Вебсайт «CardMarket»

В результаті порівняльного аналізу чітко видно, що повноцінних аналогів дослідження на тему «Розробка вебсайту для колекціонування грального карток засобами MERN та PokeApi» немає, а тому проведення дослідження та розробка відповідного додатку для колекціонування грального карток та гри ними є актуальним та вартим уваги.

1.2 Аналіз засобів розробки вебдодатків

Для розробки вебдодатків існує багато різноманітних технологій, та найпопулярнішим з них є стек розробки MERN (MongoDB, Express, React, Node), який дозволяє створювати повноцінні вебсайти з сучасним інтерфейсом, інтерактивними елементами, базою даних для зберігання інформації та серверною частиною.

MongoDB це нереляційна СУБД, що зберігає дані в вигляді документів, а не таблиць. Вона є зручною для зберігання як великих, так і малих обсягів даних, є швидкою, має багато функцій для роботи з документами, а також проста у використанні і не вимагає чіткої структури даних, до того ж чудово і зручно поєднується з іншими компонентами стеку MERN.

Node.js це середовище, що дозволяє запускати серверну частину додатку та виконувати відповідні команди. Це найпоширеніша й найзручніша платформа для роботи з серверною частиною JavaScript.

Express.js являє собою фреймворк для розробки серверної частини додатків на JavaScript, який працює з використанням Node.js. Він дозволяє створювати роути, обробляти запити з клієнтської частини на сервер, надсилати відповіді та багато інших функцій.

Стек MERN відзначається саме використанням фреймворку React, оскільки існують його менш популярні аналоги MEAN та MEVN, що використовують Angular та Vue відповідно.

Vue є простим у вивченні й використанні фреймворком, що найкраще підходить для невеликих проєктів, однак є менш функціональним в порівнянні з іншими й надає менше можливостей розробнику.

React більше підходить для великих проєктів, що зосереджені на великій кількості повторного використання компонентів, а також має велику кількість сторонніх матеріалів та бібліотек для розробки, разом з інтеграцією з більшість популярних сучасних технологій.

В свою чергу Angular є найскладнішим фреймворком з трьох, має багато вбудованих функцій для розробки великих та складних проєктів, в тому числі бібліотеку Angular Material з великою кількістю готових компонентів для створення сучасного інтерфейсу.

Для розробки даного вебзастосунку найкраще підходить саме React, оскільки він не є занадто складним, має велику кількість різноманітних функцій та чудово інтегрується з іншими засобами розробки. Таким чином видно, чому саме стек MERN був обраний для розробки вебсайту для колекціонування гральних карток.

Окрім основного стеку технологій варто зазначити, що для створення ігрових карток використовується PokeApi, який є безкоштовним та зручним сховищем даних про покемонів, яке містить актуальну інформацію та має зрозумілу документацію. Оскільки метою дослідження є розробка вебсайту, а не розробка оригінальних карток, тому використання безкоштовного ресурсу з даними є доцільним вирішенням завдання розробки ігрових карток.

1.3 Аналіз функціональних вимог

Наступним кроком до реалізації вебзастосунку є визначення вимог до нього, а також етапів розробки для спрощення та систематизації роботи. Спочатку необхідно скласти список завдань, які необхідно виконати на шляху до створення додатку, після чого визначити всі функціональні вимоги та можливості користувача.

Для розробки вебсайту для колекціонування гральних карток та гри ними потрібно виконати такі завдання:

- створити макет ігрових карток, попередньо визначивши їх наповнення;
- приєднати до додатку PokeApi, звідки братиметься наповнення карток;
- створити сторінку де будуть відображені всі існуючі картки;
- створити сторінку реєстрації та авторизації користувачів;
- розробити та приєднати базу даних для реєстрації та авторизації;
- додати персональну сторінку користувача, де буде відображена інформація про нього, а також його особиста колекція карток;
- розробити сторінку ігрового магазину, де через певні проміжки часу буде доступним випадковий набір карток для покупки;
- надати користувачу можливість за ігрову валюту придбати бажані картки;

– створити сторінку для обміну картками, надати можливість користувачам додавати на неї свої картки та задавати ціну або бажані картки для обміну;
– розробити правила та систему гри картками;
– додати сторінку для гри та програмно реалізувати ігровий процес;
– додати функціонал для гри онлайн;
– додати нагороди за перемогу або поразку в грі;
– створити сторінку з рейтингом найкращих гравців.

Виконавши всі поставлені завдання можна розробити функціонуючий додаток з виконанням усіх вимог.

Також необхідно скласти перелік можливостей користувача, тобто що саме він зможе робити в додатку, які функції використовувати, що пізніше буде відображено на діаграмі прецедентів в розділі 2. В готовому застосунку користувач повинен мати такі можливості:

– зареєструватися на сайті;
– авторизуватися на сайті;
– переглянути свою особисту сторінку;
– переглянути всі існуючі картки на відповідній сторінці;
– придбати картки в ігровому магазині;
– на своїй сторінці переглянути власну колекцію карток;
– виставити свою картку на обмін або продаж;
– обмінятись з кимось або придбати чиюсь картку;
– переглянути правила гри;
– зіграти в гру з іншими гравцями використовуючи свою колекцію карток;
– отримати нагороди за перемогу або поразку в грі;
– переглянути сторінку з рейтингом гравців.

Після реалізації всіх зазначених вище вимог до застосунку та можливостей користувача вебдодаток буде повноцінним.

Таким чином було визначено кроки до реалізації додатку та

функціональні вимоги та можливості користувача майбутнього програмного продукту.

1.4 Висновки до розділу 1

В розділі 1 було наведено аналіз аналогових додатків, обґрунтовано актуальність та необхідність проведення дослідження. Після цього було проведено порівняння засобів розробки вебсайтів, описано та аргументовано обраний стек технологій. Також було визначено завдання, що необхідно виконати протягом проведення дослідження та функціональних вимог і можливостей користувача, які необхідно реалізувати.

2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1 Проєктування структури застосунку

Наступним кроком буде перехід до етапу проєктування додатку після вибору технологій для реалізації та визначення функціональних вимог. Для цього буде використано UML-діаграми, щоб розробити відповідні діаграми для полегшення та прискорення розробки, користуючись правилами та рекомендаціями до розробки з джерела [10].

Для відображення всіх можливостей користувача необхідно створити діаграму прецедентів, використовуючи список з розділу 1. Діаграма прецедентів, зображена на рисунку 2.1, показує, які можливості користувач буде мати для взаємодії з додатком, та що зможе робити при його використанні.



Рисунок 2.1 – Діаграма прецедентів

Після розробки діаграми прецедентів, наступним кроком є визначення структури додатку за допомогою діаграми компонентів (рис. 2.2). Діаграма відображає структуру додатку – клієнтська частина використовує Vite для збірки, React, JavaScript, HTML, CSS для візуальної частини застосунку, серверна частина працює з використанням NodeJS, Express та SocketIO, а також під'єднується до бази даних. Окрім цього, до клієнтської та серверної частини підключається PokeApi для розробки карток.

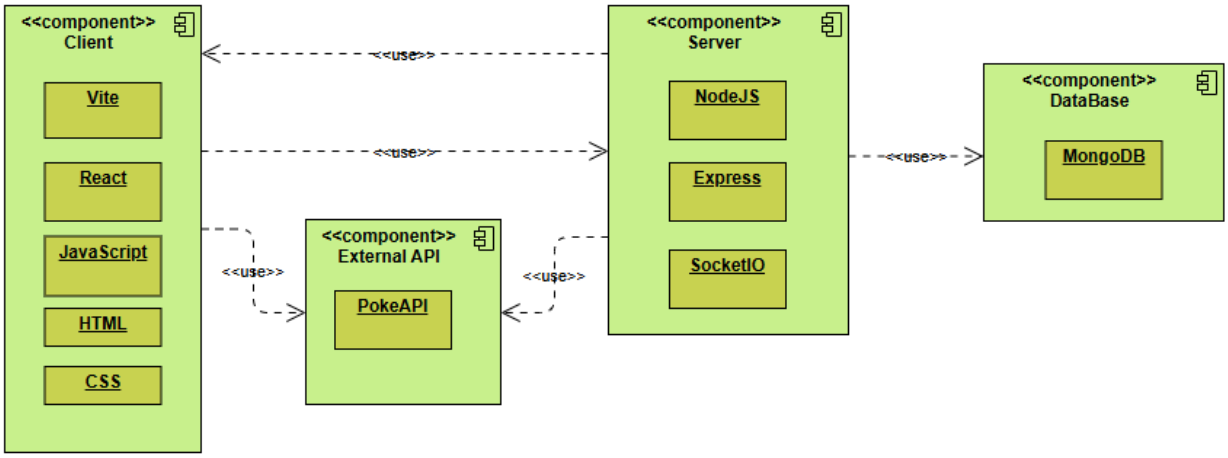


Рисунок 2.2 – Діаграма компонентів

Оскільки додаток використовує базу даних, то визначимо її схему, а саме які буде містити сутності та їх параметри. В першу чергу необхідна сутність користувача, що містить всю необхідну інформацію про гравця, включаючи особисті дані, колекцію карток, колоду та інше.

Також для кожного користувача створюється нова індивідуальна сутність ігрового магазину, що кожен день пропонує гравцю новий вибір карток для покупки.

Окрім цього потрібно створити сутність «Пропозиція», що необхідна для продажу та обміну картками. Кожен гравець може створити будь-яку кількість пропозицій, що міститимуть інформацію про картку для продажу, її ціну та дані про бажану картку, на яку користувач згодний обмінятися з іншими.

Схема бази даних зображена на рисунку 2.3.

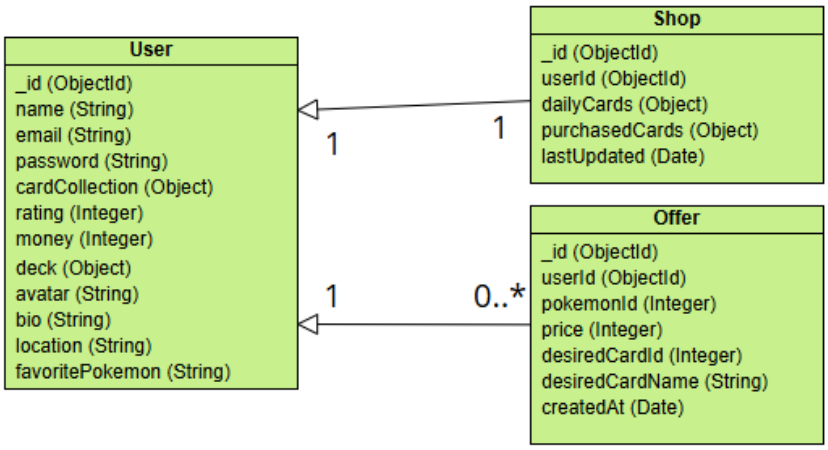


Рисунок 2.3 – Схема бази даних

Таким чином було створено діаграми прецедентів, компонентів та схему бази даних для пришвидшення та полегшення розробки майбутнього додатку.

2.2 Висновки до розділу 2

В розділі 2 було спроєктовано майбутній застосунок, розроблено необхідні UML-діаграми, такі як діаграма прецедентів, діаграма компонентів, також розроблено схему бази даних. До кожної з діаграм надано коротке пояснення для висвітлення їх змісту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Програмна реалізація додатку

Після визначення всіх функціональних вимог та проєктування застосунку можна переходити до його програмної реалізації. Сам додаток повинен містити клієнтську частину (фронт-енд), що відповідає за візуальну та інтерактивну частину додатку, а також серверну частину (бек-енд), що обробляє та надсилає відповіді на запити клієнта та працює з даними (в тому числі й з базою даних).

В першу чергу буде створено реєстрацію та авторизацію на сайті, а також під’єднано базу даних для збереження інформації користувача. Спочатку необхідно розробити серверну частину, для роботи якої буде використовуватися Express. На рисунках 3.1 – 3.2 послідовно наведено приклад коду для ініціалізації додатку та запуску серверу. В коді було створено додаток, використано бібліотеку «cors» для можливості передачі даних між двома різними доменами (а саме клієнтом та сервером), підключено базу даних, а також запущено сервер за зазначеною адресою.

```
const express = require('express');
const mongoose = require('mongoose');
const cookieParser = require('cookie-parser');

//надає можливість використання файлу .env
require('dotenv').config();
//ініціалізація додатку
const app = express();
const port = 3000;

//бібліотека що дозволяє та контролює обмін даними між різними доменами (клієнт та сервер)
let cors = require('cors')
app.use(cors({
```

Рисунок 3.1

```
  credentials: true,
  origin: "http://localhost:5173",
  methods: ["GET", "POST"],
}))

//рядок для підключення до БД
const connectionString = process.env.MONGO_URL;

//підключення до бази даних
mongoose.connect(connectionString)
.then(() => {
  console.log('connection sucessful');
})
.catch(err => {
  console.error('Connection error', err);
});

//запускаємо сервер за вказаним портом
app.listen(port, () => {
  console.log(`Express server running at http://localhost:${port}`);
});
```

Рисунок 3.2

Після ініціалізації сервера можна створити роути для реєстрації та авторизації користувача (приклад на рис. 3.3), які пізніше будуть підключені до сервера. Ці роути відповідають за запити до сервера, що надходитимуть з клієнтської частини (їх використання буде показано далі).

```
const router = express.Router();

//роути
//get, post - методи що викликаються для цих роутів (запити до сервера)
//параметри - шлях, функція що викликається при запиті до сервера

router.post('/register', registerUser)
router.post('/login', loginUser)
router.get('/profile', getProfile)
router.get('/rating', getRating)
module.exports = router
```

Рисунок 3.3 – Роути для реєстрації та авторизації користувача

Для збереження інформації користувача в базі даних необхідно створити схему даних, як це показано на рисунку 3.4. Наведений код визначає схему документу з даними користувача, а саме параметри (ім’я, електронна пошта та пароль) та їх типи даних.

```
const mongoose = require('mongoose');

//модель сутності користувача для БД
//експортується та використовується далі у програмі
const userSchema = new mongoose.Schema({
  name: String,
  email: {
    type: String,
    unique: true,
  },
  password: String,
});

module.exports = mongoose.model('User', userSchema);
```

Рисунок 3.4 – Модель даних користувача

Після всіх попередніх кроків можна перейти до розробки контролера реєстрації та авторизації, а саме функцій, що надсилатимуть відповіді на запити від клієнта та працюватимуть з даними. В цьому контролері присутні такі функції як реєстрація, логін, отримання даних поточного користувача (який зараз авторизований).

Рисунок 3.5 показує функцію реєстрації користувача. Спочатку функція перевіряє дані, надіслані в запиті з клієнта – поля не повинні бути пустими, а пароль має бути не коротший за 6 символів. Після цього перевіряється, чи існує в базі даних користувач з такою самою електронною поштою. В разі помилки в будь-якому з пунктів функція повертає відповідь з помилкою, а якщо помилок не виявлено то продовжує свою роботу. Далі відбувається шифрування пароля за допомогою бібліотеки «bcrypt» (функція шифрування наведена в додатку А), а потім створюється документ в базі даних з інформацією користувача.

```
const registerUser = async (req, res) => {
    try{
    //отримуємо дані користувача із запиту
    const {name, email, password} = req.body;
    //перевіряємо дані та повертаємо помилку якщо вони невідповідні
    if(!name){
       return res.json({
          error: "Name is required"
       })
    }
    if(!password || password.length < 6){
       return res.json({
          error: "Incorrect password"
       })
    }
    if(!email){
       return res.json({
          error: "Email is required"
       })
    }
    //перевіряємо чи існує в БД користувач з такою поштою
    const taken = await User.findOne({email});
    if(taken){
       return res.json({
          error: "Account with this email already exists"
       })
    }

    //шифруємо пароль
    const encryptedPassword = await encryptPassword(password);
    //створюємо користувача
    const user = await User.create({name, email, password: encryptedPassword});

    //створбємо унікальний токен
    const token = await createToken(
       { email: user.email, id: user._id, name: user.name },
       process.env.JWT_SECRET
     );

      //зберігаємо токен як кукі
    res.cookie('token', token);
      //повертаємо відповіть з даними нового користувача
    return res.json(user);
  }
  catch(error){
    console.log(error);
  }
}
```

Рисунок 3.5 – Реєстрація нового користувача

В кінці створюється особистий токен користувача з використанням «jsonwebtoken» (функція створення токену наведена в додатку А), що буде необхідний для контексту користувача (для розуміння програмою, який саме користувач зараз авторизований), та зберігається як кукі. Після успішного створення нового користувача надсилається відповідь до клієнта з даними цього користувача.

Після реєстрації користувача необхідно розробити відповідну функцію для авторизації або логіну користувача (рис. 3.6).

```
        //авторизація користувача
        const loginUser = async (req, res) => {
            try{
                //отримуємо дані із запиту
                const {email, password} = req.body;

                //шукаємо користувача в БД за поштою
                const user = await User.findOne({email});
                if(user){ //якщо користувач існує
                    //порівнюємо введений пароль з паролем з БД
                    const equals = await comparePassword(password, user.password);
                    if(equals){ //якщо паролі співпадають створюємо токен
                        const token = await createToken(
                            { email: user.email, id: user._id, name: user.name },
                            process.env.JWT_SECRET
                        );
                        //зберігаємо токен як кукі
                        res.cookie('token', token);
                    }
                }
                else{ //якщо користувача не існує або пароль не співпадає повертаємо помилку
                    return res.json({
                        error: "Invalid email or password"
                    })
                }
                return  res.json(user);

            }
            catch(error){
                console.log(error);
            }
        }
```

Рисунок 3.6 – Авторизація користувача

Функція перевіряє надіслані в запиті дані користувача – шукає в базі даних користувача із заданою електронною поштою, після чого порівнює надісланий пароль із паролем з бази даних. В разі помилки функція надсилає відповідь з помилкою, а в разі успішної авторизації, як і функція реєстрації, створює токен, зберігає його як кукі та повертає у відповіді дані авторизованого користувача.

Також даний контролер містить функцію отримання даних поточного користувача (того що авторизований в даний момент). Ця функція отримує токен користувача з кукі, після чого перевіряє, якому саме користувачу він належить, та повертає дані цього користувача до клієнтської частини, як показано на рисунку 3.7.

```
//отримати дані про користувача що зараз авторизований
const getProfile = async (req, res) => {
    //отримуємо токен з кукі
    const {token} = req.cookies;
    //якщо токен є
    if(token){
        //перевіряємо якому користувачу він належить
        //повертаємо помилку або користувача
        jwt.verify(token, process.env.JWT_SECRET, {}, (error, user) => {
            if(error) throw error;
            res.json(user);
        })
    }
    else {
        res.json(null);
    }
}
```

Рисунок 3.7 – Отримання даних поточного користувача

Після розробки серверної частини для реєстрації та авторизації необхідно створити клієнтську частину. Для цього потрібно ініціалізувати додаток за допомогою Vite, та при створенні проєкту в якості головного фреймворку вказати React, після чого більшість необхідних файлів буде створено автоматично.

В першу чергу потрібно розробити головний компонент додатку, рис. 3.8. Перед створенням компоненту підключається бібліотека «axios», щоб надсилати запити до серверної частини. В самому компоненті використовується спочатку контекст користувача, що буде описано далі, «тостер» для спливаючих повідомлень (бібліотека «react-hot-toast»), компонент навігаційного меню (код компонента наведено в додатку Б), а також роутер за бібліотеки «react-router-dom» для визначення шляхів до різних сторінок, що будуть розроблені пізніше.

```
//для запитів до сервера
axios.defaults.baseURL = 'http://localhost:3000';
axios.defaults.withCredentials = true;

function App() {
 //використовується контекст користувача для використання даних поточного користувача (див. ./context/userContext)
 //тостер - для спливаючих сповіщень на сайті
 //роути - показують, за яким шляхом знаходяться сторінки (./pages)
 return (
  <UserContextProvider>
  <Toaster position='bottom-right' toastOptions={{duration: 3000}}/>
   <Navbar>   </Navbar>
   <Routes>
    <Route path='/' element={<Home />}/>
    <Route path='/register' element={<Register />}/>
    <Route path='/login' element={<Login />}/>
    <Route path='/dashboard' element={<Dashboard/>}/>
   </Routes>
  </UserContextProvider>
 )
}
```

Рисунок 3.8 – Головний компонент додатку

На рисунках 3.9 – 3.10 послідовно наведено код сторінки реєстрації користувача. Сторінка реєстрації містить об’єкт для збереження даних користувача, а також внутрішню функцію, що надсилає запит до сервера для реєстрації. Якщо сервер надсилає відповідь з помилкою, то користувач бачить

спливаюче повідомлення з текстом помилки, а в разі успішної реєстрації – відповідне повідомлення, після чого перенаправляється на головну сторінку сайту. Сама сторінка реєстрації складається з форми, поля якої прив’язані до об’єкту з даними, та при підтвердженні якої надсилається запит до сервера з усіма введеними даними.

```
export default function Register(){
    //навігатор - для перенаправлення на іншу сторінку
    const navigation = useNavigate();
    //дані користувача
    const [userData, setData] = useState({
        name: '',
        email: '',
        password: '',
    })

    //функція реєстрації викликається при підтвердження форми нижче
    const register = async (e) => {
        e.preventDefault();
        const {name, email, password} = userData;
        try{
            //запит до сервера post надсилає введені користувачем дані
            const {data} = await axios.post('/register', {name, email, password});
            //якщо відповідь від сервера містить помилку
            if(data.error){
                //виводимо спливаюче повідомлення про помилку
                toast.error(data.error);
            } else {
                //якщо користувач зареєстрований видаляємо його дані з форми
                setData({});
                //повідомлення про успішну реєстрацію
                toast.success('Registration successful!');
                //перенаправлення на іншу сторінку сайту
                navigation('/dashboard');
            }
        }
        catch(error){
            console.log(error);
        }

    }
```

Рисунок 3.9

```
        //форма реєстрації
        //при підтвердженні викликається функція реєстрації
        //всі поля привязуються до відповідних змінних
        //при зміні значень полей вони записуються в змінні
        return (
          <form onSubmit={register}>
            <label>Name</label>
            <input type='text' placeholder="Enter name" value={userData.name}
onChange={(e) => setData({...userData, name: e.target.value})}></input>
            <label>Email</label>
            <input type='email' placeholder="Enter Email" value={userData.email}
onChange={(e) => setData({...userData, email: e.target.value})}></input>
            <label>Password</label>
            <input type='password' placeholder="Enter password" value={userData.password}
onChange={(e) => setData({...userData, password: e.target.value})}></input>
            <button type="submit">Register</button>
          </form>
        )
      }
```

Рисунок 3.10

Аналогічно до сторінки реєстрації створюється сторінка авторизації користувача (функція логіну на рисунку 3.11). Наведена функція також надсилає запит до сервера з введеними даними користувача, та при успішній авторизації вітає користувача та перенаправляє на сторінку з рейтингом користувачів.

```
      const login = async (e) => {
        e.preventDefault();
        const {email, password} = userData;
        try{
          const {data} = await axios.post('/login', {email, password});
          if(data.error){
            toast.error(data.error);
          } else {
            setData({});
            toast.success("Welcome, " + data.name + "!");
            navigation('/dashboard');
          }
        }
        catch(error){
          console.log(error);
        }
      }
```

Рисунок 3.11 – Функція авторизації користувача

Окрім вищезазначених сторінок необхідно створити компонент контексту користувача, який буде надавати додатку інформацію про те, який саме користувач зараз авторизований в системі. На рисунку 3.12 показано реалізацію цього компоненту. Функція фреймворку React «useEffect» викликається щоразу при завантаженні сторінки, і якщо ще не було визначено поточного користувача, вона робить запит до сервера на отримання даних авторизованого користувача (ця функція наведена раніше на рис. 3.7).

```
//контекст користувача - який користувач зараз авторизований
export const UserContext = createContext({});

export function UserContextProvider({children}) {

   const [user, setUser] = useState(null);
   //ця функція викликається щоразу при завантаженні сторінки
   useEffect(() => {
      if(!user){
         //якщо ще не встановлено який зараз користувач авторизований
         //робиться запит до сервера для отримання даних поточного користувача
         axios.get('/profile').then(({data}) => {
            //запис даних користувача в змінну
            setUser(data);
         })
      }
   })

   return(
      <UserContext.Provider value={{user, setUser}}>
         {children}
      </UserContext.Provider>
   )
}
```

Рисунок 3.12 – Визначення контексту користувача

Після цього розробляється головна сторінка сайту (рис. 3.13), яка містить привітання нового або авторизованого користувача з використанням контексту користувача (контекст повертає дані поточного користувача, а сторінка виводить привітання з іменем), а також таблицю зі списком перших десяти користувачів з найвищим рейтингом. Таблиця є окремим компонентом, що також робить запит до сервера, де всі користувачі сортуються за рейтингом та виводяться перші десять гравців.

```
export default function Dashboard(){

  //використовуємо контекст користувача для використання його даних
  const {user} = useContext(UserContext);

  //з контексту отримуємо імя поточного користувача й виводимо його
  return (
    <div>
      <h1>Dashboard</h1>
      {!!user && (<h2>Welcome, {user.name}!</h2>)}

      <DataTable></DataTable>
    </div>


  );
}
```

Рисунок 3.13 – Головна сторінка сайту

Далі необхідно створити особисту сторінку користувача, що відображатиме дані про нього. Для завантаження даних використовується контекст користувача, а для їх відображення – компонент UserInfo (рис. 3.14), що також використовується для перегляду профілів інших користувачів. На сторінці також містяться кнопки для редагування особистої інформації та перегляду власної колекції карток.

```
const UserInfo = ({ user }) => {

  return (
    <div className="profile-page">
      <div className="profile-header">
   <img src={user.avatar || 'https://cdn-icons-png.flaticon.com/128/18020/18020062.png'}
          alt={user.name}
          className="avatar"
        />
        <h1>{user.name}</h1>
        <p className="profile-email">{user.email}</p>
      </div>
      <div className="profile-details">
        <p><strong>Bio:</strong> {user.bio || 'No bio available'}</p>
        <p><strong>Location:</strong> {user.location || 'No location provided'}</p>
          <p><strong>Favorite Pokémon:</strong> {user.favoritePokemon || 'No favorite
Pokémon listed'}</p>
        <p><strong>Balance:</strong> {user.money || 'No money'}</p>
        <p><strong>Rating:</strong> {user.rating || 'No rating'}</p>
      </div>
    </div>
  );
};
```

Рисунок 3.14 – Компонент для відображення даних користувача

Дуже важливим кроком є розробка компоненту ігрової картки, оскільки він буде використовуватися дуже часто. Дані для карток завантажуються за запитом до PokeApi та використовуються при створенні картки (рис. 3.15).

```
        useEffect(() => {
          const fetchPokemonData = async () => {
            try {
                                              const response = await
axios.get(`https://pokeapi.co/api/v2/pokemon/${pokemonId}`, {
              baseURL: '',
              withCredentials: false,
            });
            setPokemonData(response.data);

            const speciesResponse = await axios.get(response.data.species.url, {
              baseURL: '',
              withCredentials: false,
            });
            setPokemonSpecies(speciesResponse.data);

              setLoading(false);
            } catch (error) {
              console.error('Error fetching Pokemon or species data:', error);
              setError(true);
              setLoading(false);
            }
          };

          fetchPokemonData();
        }, [pokemonId]);
```

Рисунок 3.15 – Завантаження даних про покемона в картку

Макет кожної картки всередині містить ім’я покемона, його зображення та короткий опис. В трьох кутах картки трохи виходячи за її межі знаходяться кола з характеристиками – червоне для атаки, блакитне для захисту та зелене для здоров’я. Також додаються окремі стилі для картки, якщо вона буде знаходитися в грі, щоб кола з характеристиками знаходилися всередині картки, а опис не показувався зовсім для зручності використання. Зовнішній вигляд картки показано на рисунку 3.16.

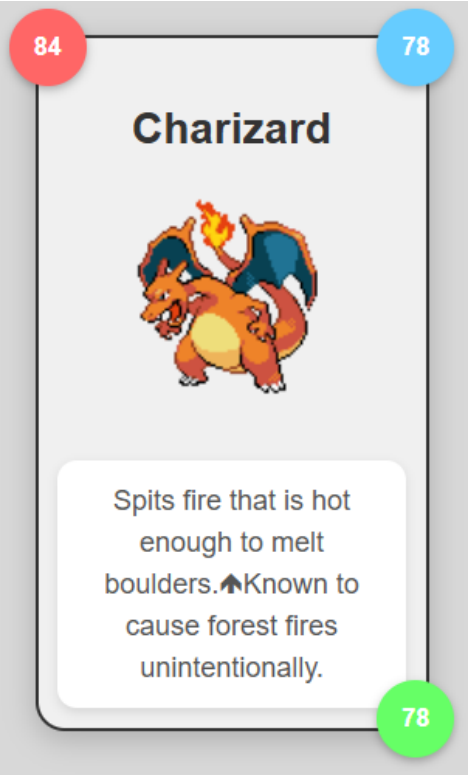


Рисунок 3.16 – Зовнішній вигляд картки

Тепер з розробленим компонентом для карток можна створити сторінку бібліотеки, де будуть відображені всі існуючі картки, що гравець може зібрати (повний код наведено в додатку Б). Для зручності на етапі розробки додаток використовує лише перші 151 покемонів (перше покоління). На сторінку бібліотеки також додається розподілення на сторінки (по 20 карток на кожній) з відповідними кнопками для навігації та пошукове поле для фільтрування карток за іменем покемонів.

Аналогічно до бібліотеки реалізується сторінка з картковою колекцією гравця, однак замість всіх карток відображаються лише ті, якими володіє користувач. Важливим елементом цієї сторінки є можливість користувача збирати свою колоду не більше ніж з 15 карток. При натисканні картка виділяється блакитним кольором, при повторному кліку виділення знімається, а при спробі виділити більше 15 карток картки перестануть виділятися. При натисканні відповідної кнопки, що також показує кількість вибраних карток, надсилається запит на сервер для збереження в базі даних колоди гравця. Функції виділення карток та збереження колоди наведено на рисунку 3.17.

```
const handleCardSelect = (pokemonId) => {
    if (selectedCards.includes(pokemonId)) {
      setSelectedCards(selectedCards.filter((id) => id !== pokemonId));
    } else if (selectedCards.length < maxDeckSize) {
      setSelectedCards([...selectedCards, pokemonId]);
    }};
  const handleSaveDeck = async () => {
    try {
      const response = await axios.post(`/save-deck/${user._id}`, {
        deck: selectedCards,
      });
      const updatedUser = { ...user, deck: response.data.deck };
      setUser(updatedUser);
      console.log(user)
      toast.success('Deck saved successfully');
    } catch (error) {
      console.error('Error saving deck:', error);
      toast.error('Failed to save deck');
    }};
```

Рисунок 3.17 – Виділення карток та збереження колоди

Наступним кроком потрібно розробити магазин з ігровими картками, що створюється для кожного користувача індивідуально щодня та містить випадкові 10 карток для покупки. Кожна картка має випадкову ціну, що написана під нею, та кнопку «Придбати», що стає неактивною, якщо гравець вже придбав цю картку для запобігання повторної покупки. В нижній частині сторінки знаходиться таймер для відліку часу до наступного оновлення магазину. На стороні серверу створюється модель магазину для бази даних та функції для випадкового вибору карток, підрахунку часу до наступного оновлення та покупки картки. На рисунку 3.18 показано функцію для покупки картки.

Схожим чином реалізується і сторінка для продажу та обміну картками між гравцями. В першу чергу створюється модель для пропозиції, як це показано на схемі бази даних в розділі 2. Після цього потрібно розробити відповідну сторінку на сайті, на якій будуть відображатися всі пропозиції та буде форма для створення нової.

```
const purchaseCard = async (req, res) => {
  const { userId } = req.params;
  const { pokemonId } = req.body;
  try {
    const shop = await Shop.findOne({ userId });
    if (!shop) {
      return res.status(404).json({ error: 'Shop not found' });
    }
      const cardToPurchase = shop.dailyCards.find(card => card.pokemonId === pokemonId);
    if (!cardToPurchase) {
      return res.status(404).json({ error: 'Card not available for purchase' });
    }
     const isAlreadyPurchased = shop.purchasedCards.some(card => card.pokemonId === pokemonId);
    if (isAlreadyPurchased) {
      return res.status(400).json({ error: 'Card already purchased' });
    }
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }
    if (user.money < cardToPurchase.price) {
      return res.status(400).json({ error: 'Not enough money' });
    }
    user.money -= cardToPurchase.price;
    user.cardCollection.push({ pokemonId: cardToPurchase.pokemonId });
    shop.purchasedCards.push(cardToPurchase);

    await user.save();
    await shop.save();

    res.json({
      message: 'Card purchased successfully',
      purchasedCard: cardToPurchase,
      money: user.money
    });
  } catch (error) {
    console.error('Error purchasing card:', error);
    res.status(500).json({ error: 'Failed to purchase card' });
  }
};
```

Рисунок 3.18 – Функція для покупки картки

Форма представляє собою модальне вікно, в якому також відображається колекція карт гравця, де він може виділити лише одну яку бажає продати чи обміняти. Внизу форми є поле для вводу бажаної ціни, а

також можливість обрати бажану картку для обміну. При підтвердженні форми створюється нова пропозиція та відображається на основній сторінці. Кожна пропозиція складається з картки та двох кнопок – придбати та обміняти, при чому друга кнопка є неактивною, якщо гравець не має в своїй колекції потрібної картки. Також важливо зазначити, що власник пропозиції має кнопку «Видалити пропозицію», яка відповідно за запитом до серверу видаляє пропозицію з бази даних.

На сервері функція обміну картками працює наступним чином – спершу проходять всі необхідні перевірки на існування користувачів, існування картки в колекції продавця, достатньої кількості грошей або наявності необхідної для обміну картки в колекції покупця. При обміні картка видаляється з колекції та колоди продавця та додається в колекцію покупця, аналогічно необхідна для обміну картка видаляється з колекції та колоди покупця та додоється в колекцію продавця (або при покупці за гроші вони віднімаються від балансу покупця та додаються до балансу продавця).

Після розробки всіх попередніх сторінок та компонентів можна переходити до розробки функціоналу гри з використанням карток. В першу чергу програма перевіряє, чи не пуста колода гравця, оскільки без неї неможливо грати. Якщо жодної картки не додано до колоди показується модальне вікно з відповідним повідомленням та кнопкою, що перенаправляє користувача до сторінки з його колекцією карток для заповнення колоди. Якщо колода не пуста, то модальне вікно містить дві кнопки для вибору режиму гри – проти комп’ютера або іншого гравця.

Сторінка з грою має такі елементи: ілюстративні елементи колод гравців, інформація про користувачів (аватар, ім’я та полоса здоров’я), їх «руки» (набір взятих карток), ігрове поле з п’ятьма місцями для покемонів для кожного гравця, індикатор ходу та дві кнопки для витягування нової картки та завершення ходу.

За правилами гри гравці під час свого ходу можуть взяти картку (не більше 5), покласти картку на поле та завершити хід нічого не зробивши. В

кінці ходу покемони на полі атакують або покемонів навпроти них (якщо такі є), або за їх відсутності атакують противника зменшуючи його здоров'я. Якщо в одного з гравців здоров'я дорівнює нулю гра завершується.

Для реалізації гри створюється React-компонент з усіма необхідними елементами та логікою для підтримки дій гравця, а також JavaScript клас з рештою ігрової логіки. На початку гри для гравця завантажується його колода, а для комп'ютера – створюється з випадкового набору карток, після чого колода кожного гравця перемішується. При грі з комп'ютером програма очікує дії гравця, після чого хід передається комп'ютеру. Його дії обираються випадково для елементу непередбачуваності та цікавості під час гри – він може або взяти картку якщо не має п'яти, або покласти випадкову картку на випадкове місце на полі. Після цього хід повертається назад до гравця.

Дуже важливим аспектом гри є саме мультиплеєр. Для його реалізації використовується SocketIO для швидкої та зручної передачі даних між двома користувачами. Коли гравець обирає гру з іншим гравцем відображається повідомлення про очікування другого гравця. Коли інший гравець обирає ту саму опцію та підключається до гри сокет запускає подію початку гри та надсилає гравцям дані свого опонента, такі як ім'я, аватар, колода та набрані в руку карти (рис. 3.19).

Коли гравець виконує дію на сервері викликається відповідна подія. На рисунку 3.20 показано функцію викладання гравцем картки на поле та виклику відповідної події на сервері.

```
io.on("connection", (socket) => {
  socket.on("joinGame", (playerData) => {
    players[socket.id] = {
      id: socket.id,
      name: playerData.name,
      avatar: playerData.avatar,
      deck: playerData.deck,
      hand: [],
      placedCards: [],
      health: 500,
      turn: false
```

```
        };

        socket.join("gameRoom");
        if (Object.keys(players).length === 2) {
            const [player1, player2] = Object.keys(players);
            io.to("gameRoom").emit("playersReady", { player1, player2 });
            players[player1].turn = true;
            io.to(player1).emit("startGame", {
                opponentData: {
                    name: players[player2].name,
                    avatar: players[player2].avatar,
                    deck: players[player2].deck
                },
                turn: true
            });
            io.to(player2).emit("startGame", {
                opponentData: {
                    name: players[player1].name,
                    avatar: players[player1].avatar,
                    deck: players[player1].deck
                },
                turn: false
            });
        }
    });
```

Рисунок 3.19 – Функція початку гри й передачі даних

```
    placePlayerCard(index, selectedCard) {
        if (this.playerPlacedCards[index] === null && selectedCard !== null) {
            this.playerPlacedCards[index] = selectedCard;
            this.playerHand = this.playerHand.filter(card => card !== selectedCard);

              if(this.isMultiplayer) this.socket.emit('playerPlaceCard', { index: index, card:
selectedCard });
            return true;
        }
        return false;
    }
```

Рисунок 3.20 – Викладання картки на ігрове поле

Оскільки в кінці кожного ходу відбувається атака викладеними картками, то саме за цією подією відбувається передача ходу опоненту. На рисунку 3.21 відображено функцію, що у відповідь на атаку іншого гравця

оновлює ігровий компонент та відображає зміни, а також перевіряє чи закінчена гра шляхом перевірки здоров’я гравців.

```
socketRef.current.on('opponentAttack', async ()=>{
        forceUpdate();
        await animateCardsOnField(gameRef.current.opponentPlacedCards, 'opponent');
        setOpponentTurn(false);
        checkGameOver();
    })
```

Рисунок 3.21 – Відповідь на атаку іншого гравця

Після завершення гри з’являється модальне вікно з її результатами, та за перемогу гравець отримує ігрову валюту та рейтинг, та меншу кількість валюти за поразку. Альтернативним варіантом завершення гри є відключення опонента, за що гравець також отримує невелику кількість валюти. Повний код гри наведено у додатках.

Останнім кроком необхідно створити просту сторінку з правилами гри для зручності користування вебсайтом. Реалізація такої сторінки є дуже простою, тому її код наведено в додатку Б.

Після реалізації всіх необхідних функцій та виконання функціональних вимог до додатку його необхідно протестувати та за необхідності виправити помилки. На наступних зображеннях 3.22 – 3.27 показано зовнішній вигляд та приклади сторінок розробленого вебзастосунку.

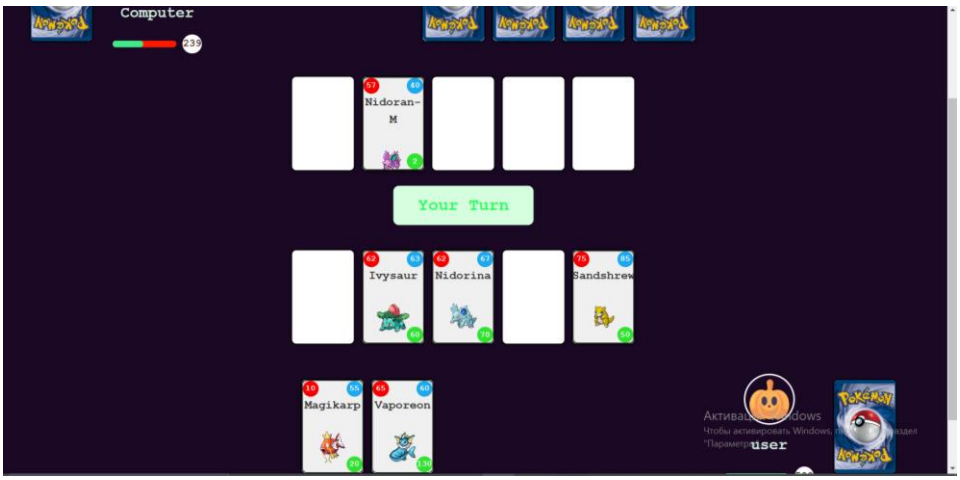


Рисунок 3.22 – Процес гри



Рисунок 3.23 – Таблиця рейтингу гравців

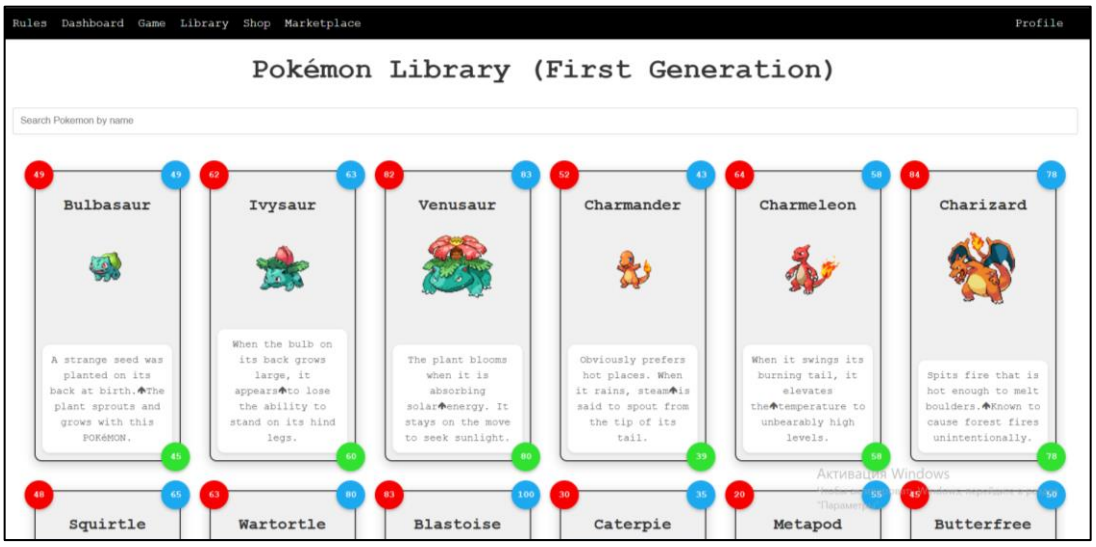


Рисунок 3.24 – Бібліотека усіх карток

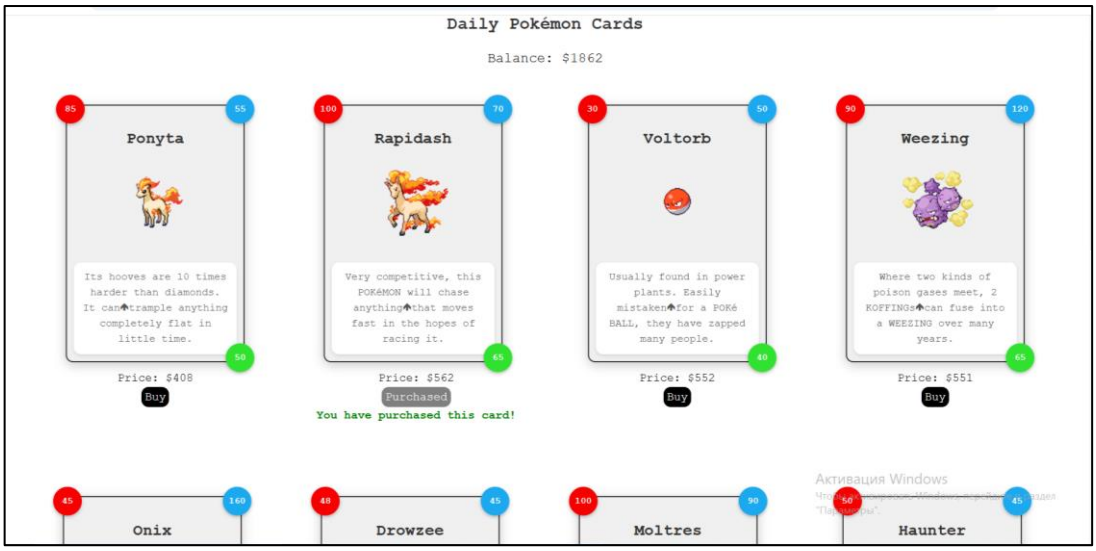


Рисунок 3.25 – Магазин карток

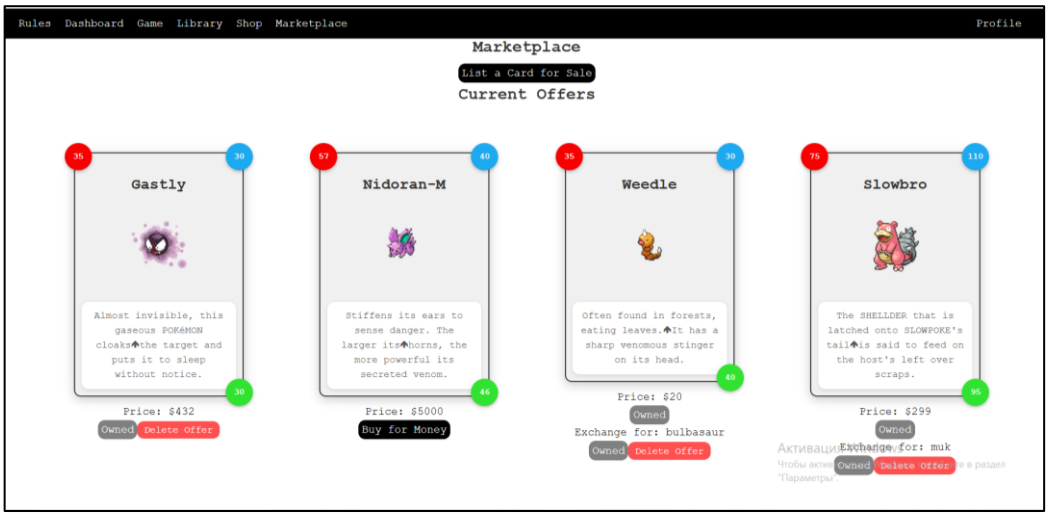


Рисунок 3.26 – Продаж або обмін картками між гравцями

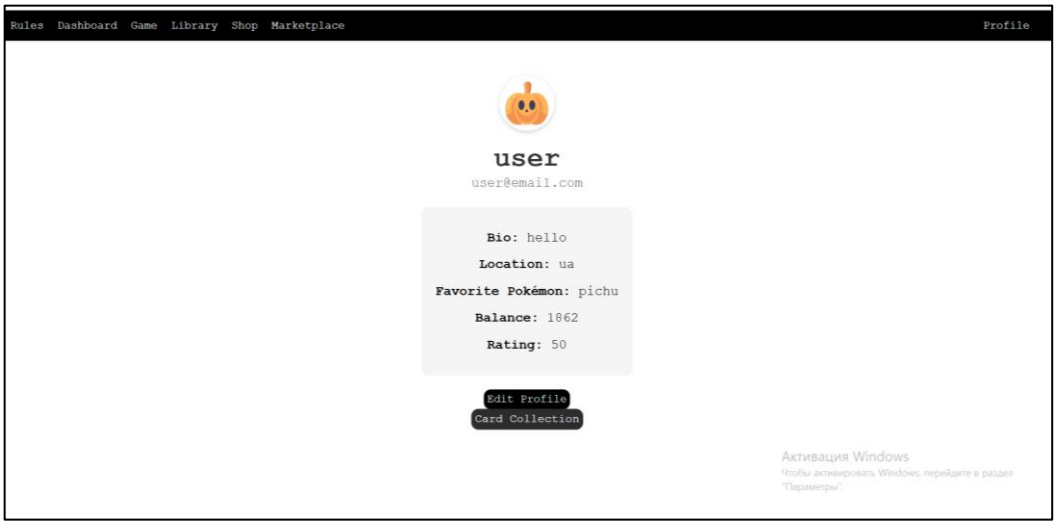


Рисунок 3.27 – Сторінка профілю

Таким чином було розроблено механізм реєстрації та авторизації користувача зі збереженням інформації в базі даних, а також використанням контексту користувача.

3.2 Висновки до розділу 3

В розділі 3 було розроблено серверну та клієнтську частину додатку, пояснено принципи роботи вебзастосунку, наведено приклади основних функцій та частин програми, а також показано приклади роботи сайту.

ВИСНОВКИ

В процесі дослідження було розроблено вебсайт для колекціонування гральних карток з використанням MERN та PokeApi. В роботі було проведено аналіз предметної області, спроєктовано додаток та розроблено необхідні діаграми, також реалізовано програмну частину та описано її роботу.

В розділі 1 проведено аналіз та порівняння аналогових продуктів, проаналізовано актуальність програми, після чого було проведено порівняння найпоширеніших засобів розробки вебсайтів, пояснено вибір технологій. Окрім цього було розроблено список функціональних вимог додатку.

В розділі 2 було спроєктовано застосунок та розроблено діаграми для зручності подальшої розробки. Для усіх діаграм було надано пояснення щодо її змісту.

В розділі 3 було наведено опис процесу розробки застосунку та пояснено логіку роботи програми та її головних частин. Після цього було протестовано готовий додаток та показано зовнішній вигляд сторінок вебсайту.

У майбутньому гру можна покращувати, наприклад додати більше гральних карток з особливими ефектами та внутрішньо ігровий чат, щоб гравці мали можливість спілкуватися між собою під час гри.

Таким чином в результаті роботи було розроблено та спроєктовано вебзастосунок для колекціонування гральних карток з використанням MERN та PokeApi, а також просту карткову гру з виконанням всіх функціональних вимог та розробкою необхідних можливостей користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1. Brown E. Web Development with Node and Express. Sebastopol : O’Reilly Media Inc., 2014. 306 p.

2. D'mello B. J., Satheesh M., Krol J. Web Development with MongoDB and Node : 3rd ed. Birmingham : Packt Publishing Ltd., 2017. 304 p.

3. Sharma A. Full-Stack Web Development with Vue.js and Node: Build scalable and powerful web apps with modern web stack: MongoDB, Vue, Node.js, and Express. Birmingham : Packt Publishing Ltd., 2018. 366 p.

4. Dickey J. Write Modern Web Apps with the MEAN Stack: Mongo, Express, AngularJS, and Node.js. Peachpit Press, 2015. 238 p.

5. Wilson E., Koroliova I. MERN Quick Start Guide: Build web applications with MongoDB, Express.js, React, and Node. Birmingham : Packt Publishing Ltd., 2018. 278 p.

6. Biswas N. MERN Projects for Beginners: Create Five Social Web Apps Using MongoDB, Express.js, React, and Node. Apress, 2021. 285 p.

7. Subramanian V. Pro MERN Stack. Apress, 2017. 328 p.

8. Hoque S. Full-Stack React Projects : 2nd ed. Birmingham : Packt Publishing Ltd., 2020. 682 p.

9. Півняк Г. Г., Бусигін Б. С., Дівізінюк М. М. та ін. Тлумачний словник з інформатики. Дніпро : Нац. гірнич. ун-т, 2010. 600 с.

10. Patrick G. Henriette B. Philippe B. UML 2.0 in Action. Birmingham : Packt Publishing, 2005. 229 p.

ДОДАТОК А

Код серверної частини додатку

```
const User = require('../models/UserModel')
const {encryptPassword, comparePassword, createToken} = require('../utils/auth')
const jwt = require('jsonwebtoken')

//реєстрація нового користувача
const registerUser = async (req, res) => {


   try{
      //отримуємо дані користувача із запиту
      const {name, email, password} = req.body;
      //перевіряємо дані та повертаємо помилку якщо вони невідповідні
      if(!name){
         return res.json({
            error: "Необхідне імя"
         })
      }
      if(!password || password.length < 6){
         return res.json({
            error: "Пароль не підходить"
         })
      }
      if(!email){
         return res.json({
            error: "Необхідно ввести пошту"
         })
      }
      //перевіряємо чи існує в БД користувач з такою поштою
      const taken = await User.findOne({email});
      if(taken){
         return res.json({
            error: "Такий користувач вже існує"
         })
      }

      //шифруємо пароль
      const encryptedPassword = await encryptPassword(password);
      //створюємо користувача
      const user = await User.create({name, email, password: encryptedPassword});

      //створбємо унікальний токен
      const token = await createToken(
         { email: user.email, id: user._id, name: user.name },
         process.env.JWT_SECRET
        );

        //зберігаємо токен як кукі для подальшого відстеження, який користувач зараз авторизований
      res.cookie('token', token);
        //повертаємо відповіть з даними нового користувача
      return res.json(user);
   }
   catch(error){
      console.log(error);
   }
```

```
}

//авторизація користувача
const loginUser = async (req, res) => {
   try{
      //отримуємо дані із запиту
      const {email, password} = req.body;

      //шукаємо користувача в БД за поштою
      const user = await User.findOne({email});
      if(user){ //якщо користувач існує
         //порівнюємо введений пароль з паролем з БД
         const equals = await comparePassword(password, user.password);
         if(equals){ //якщо паролі співпадають створюємо токен
            const token = await createToken(
               { email: user.email, _id: user._id, name: user.name },
               process.env.JWT_SECRET
             );
             //зберігаємо токен як кукі
            res.cookie('token', token);
         }
      }
      else{ //якщо користувача не існує або пароль не співпадає повертаємо помилку
         return res.json({
            error: "Невірний пароль або пошта"
         })
      }
      return  res.json(user);

   }
   catch(error){
      console.log(error);
   }
}

//отримати дані про користувача що зараз авторизований
const getProfile = async (req, res) => {
   //отримуємо токен з кукі
   const { token } = req.cookies;

   //якщо токен є
   if (token) {
      jwt.verify(token, process.env.JWT_SECRET, async (error, decoded) => {
         if (error) {
            return res.status(401).json({ message: 'немає доступу' });
         }

         try {

            const user = await User.findById(decoded._id);
            if (!user) {
                  return res.status(404).json({ message: 'Користувач не знайдений' }); //повертає 404, якщо
користувача не знайдено
               }
               //повертає дані користувача, за винятком пароля
               const { password, ...userData } = user.toObject(); //toObject, щоб отримати об’єкт JavaScript
               res.json(userData); //надсилає повні дані користувача
            } catch (dbError) {
               console.error(dbError);
               res.status(500).json({ message: 'помилка сервера' }); //обробка помилок бази даних
            }
         });
      } else {
```

```
        res.status(401).json({ message: 'токен не знайдено' }); //поверне 401, якщо немає токену
    }
};

//отримати рейтинг користувачів
const getRating = async (req, res) => {
    try{
        //беремо всіх користувачів з БД та сортуємо за іменем
        const users = await User.find().sort({ rating: -1 }).limit(10);

        //якщо користувачів немає
        if(!users){
            return res.json({ error: "Користувачів не знайдено"});
        }
        //повертаємо користувачів
        return res.json(users);
    }
    catch(error){
        console.log(error);
    }
}

//отримати дані про будь якого користувача за айді
const getUser = async (req, res) => {
    const { userId } = req.params;
    try {
        const user = await User.findById(userId);
        if (!user) {
            return res.status(404).json({ error: 'Користувач не знайдений' });
        }
        res.json(user);
    } catch (err) {
        res.status(500).json({ error: 'Помилка сервера' });
    }
}

//оновити профіль
const updateProfile = async (req, res) => {
    const { userId, avatar, bio, location, favoritePokemon } = req.body;

    try {
      //знаходить користувача за айді і оновляє поля
      const updatedUser = await User.findByIdAndUpdate(
        userId,
        { avatar, bio, location, favoritePokemon },
        { new: true, runValidators: true }
      );

      if (!updatedUser) {
        return res.status(404).json({ message: 'Користувавч не знайдений' });
      }

      res.json(updatedUser);
    } catch (error) {
      res.status(500).json({ message: 'Помилка при оновлені профілю', error });
    }
}

//оновити дані користувача
const updateUser = async (req, res) => {
    const { userId, rewardAmount, newRating } = req.body;

    try {
```

```
        //знаходить користувача за айді
        const user = await User.findById(userId);
        if (!user) {
            return res.status(404).json({ error: 'Користувач не знайдений' });
        }

        //оновляє гроші та рейтинг користувача
        user.money += rewardAmount;
        user.rating = newRating;

        await user.save();

        res.status(200).json({
            message: 'оновлення успішне'
        });
    } catch (error) {
        console.error('помилка', error);
        res.status(500).json({ error: 'не вийшло оновити дані' });
    }
}

module.exports = {
    registerUser, loginUser, getProfile, getUser, updateProfile, getRating, updateUser
}

const User = require('../models/UserModel')

//зберегти колоду
const saveDeck = async (req, res) => {
    const { userId } = req.params;
    const { deck } = req.body;

    if (!Array.isArray(deck) || deck.length > 15) {
        return res.status(400).json({ error: 'Колода може мітсити не більше 15 карток' });
    }

    try {
        //знаходить користувача по айді
        const user = await User.findById(userId);

        if (!user) {
            return res.status(404).json({ error: 'Користувач не знайдений' });
        }

        user.deck = [];
        deck.forEach(card => {
            user.deck.push({pokemonId: card})
        });
        //зберігає колоду в профілі користувача перезаписуючи попередню колоду

        await user.save();

        res.json({ message: 'Колода збережена', deck: user.deck });
    } catch (error) {
        console.error('помилка ', error);
        res.status(500).json({ error: 'Не вийшло зберегти колоду' });
    }
};

//отримати колоду користувача
const getDeck = async (req, res) => {
    try {
        const user = await User.findById(req.params.userId);
```

```
            if (!user) {
                return res.status(404).json({ message: 'Користувач не знайдений' });
            }
            res.json({ deck: user.deck });
        } catch (error) {
            console.error(error);
            res.status(500).json({ message: 'Помилка серверу' });
        }
    }
    module.exports = {saveDeck, getDeck}

    const Shop = require('../models/ShopModel');
    const User = require('../models/UserModel')
    const Offer = require('../models/OfferModel')

    //функція для створення випадкових покемонів і цін
    const generateDailyCards = () => {
      const dailyCards = [];
      const availablePokemonIds = Array.from({ length: 151 }, (_, i) => i + 1); //беремо покемонів першого покоління

      //вибрав 10 карток на день
      const shuffled = availablePokemonIds.sort(() => 0.5 - Math.random()).slice(0, 10);

      //випадкові ціни для вибраних покемонів
      shuffled.forEach(pokemonId => {
        dailyCards.push({
          pokemonId,
          price: Math.floor(Math.random() * 500) + 100, //ціна випадкова від 100 до 500
        });
      });

      return dailyCards;
    };

    //оновлення магазину щодня
    const updateDailyShop = async (userId) => {
      try {
        let shop = await Shop.findOne({ userId });

        //перевірка чи пойшло 24 години
        const now = new Date();
        const oneDay = 24 * 60 * 60 * 1000;

        if (!shop || now - new Date(shop.lastUpdated) > oneDay) {
          const dailyCards = generateDailyCards();

          //якщо магазину нема створити новий
          if (!shop) {

            shop = new Shop({
              userId,
              dailyCards,
              purchasedCards: [],
            });
          } else {
            //інакше оновити набір карток
            shop.dailyCards = dailyCards;
            shop.lastUpdated = now;
          }

          await shop.save();
        }
```

```
    return shop;
  } catch (error) {
    console.error('Помилка оновлення магазину', error);
  }
};

//отримати дані магазину
const getShopSelection = async (req, res) => {
    const { userId } = req.params;

    try {
      const shop = await updateDailyShop(userId);
      res.json(shop);
    } catch (error) {
      console.error('Помилка завантаження магазину', error);
      res.status(500).json({ error: 'Помилка завантаження магазину' });
    }
};

//купити картку
const purchaseCard = async (req, res) => {
  const { userId } = req.params;
  const { pokemonId } = req.body;

  try {
      const shop = await Shop.findOne({ userId });
      if (!shop) {
          return res.status(404).json({ error: 'Магазин не знайдено' });
      }

      //перевірка чи доступна картка
      const cardToPurchase = shop.dailyCards.find(card => card.pokemonId === pokemonId);
      if (!cardToPurchase) {
          return res.status(404).json({ error: 'Картку неможливо придбати' });
      }

      //перевірка чи ввже придбали картку
      const isAlreadyPurchased = shop.purchasedCards.some(card => card.pokemonId === pokemonId);
      if (isAlreadyPurchased) {
          return res.status(400).json({ error: 'Картка вже куплена' });
      }

      //отримуємо та підтверджуємо користувача
      const user = await User.findById(userId);
      if (!user) {
          return res.status(404).json({ error: 'Користувач не знайдений' });
      }

      if (user.money < cardToPurchase.price) {
          return res.status(400).json({ error: 'Недостатньо грошей' });
      }

      //знімаємо гроші з балансу і додаємо картку до колекції
      user.money -= cardToPurchase.price;
      user.cardCollection.push({ pokemonId: cardToPurchase.pokemonId });

      //додаємо картку до карток, придбаних у магазині, що б не купувати 2 рази те ж саме
      shop.purchasedCards.push(cardToPurchase);

      await user.save();
      await shop.save();

      res.json({
```

```
            message: 'Куртка придбана успішно',
            purchasedCard: cardToPurchase,
            money: user.money
        });
    } catch (error) {
        console.error('Помилка при покупці', error);
        res.status(500).json({ error: 'Не вийшло купити картку' });
    }
};

//створити пропозицію
const createOffer = async (req, res) => {
    const { userId, pokemonId, price, desiredCardId, desiredCardName } = req.body;

    try {
        const newOffer = new Offer({ userId, pokemonId, price, desiredCardId, desiredCardName });
        await newOffer.save();
        res.status(201).json(newOffer);
    } catch (error) {
        res.status(400).json({ message: error.message });
    }
}

//отримати всі пропозиції
const getOffers = async (req, res) => {
    try {
        const offers = await Offer.find().populate('userId', 'username');
        res.status(200).json(offers);
    } catch (error) {
        res.status(400).json({ message: error.message });
    }
}

//придбати картку за пропозицією
const purchaseOffer = async (req, res) => {
    const { offerId, buyerId } = req.params;
    //cardId надіслано в тілі запиту
    const { cardId } = req.body;

    try {
        //знаходимо пропозицію на основі offerId
        const offer = await Offer.findById(offerId).populate('userId');
        if (!offer) {
            return res.status(404).json({ error: 'Пропозицію не знайдено' });
        }

        const { userId, price } = offer;

        //покупець не є продавцем
        if (buyerId === userId.toString()) {
            return res.status(400).json({ error: 'Не можна купити власну картку' });
        }

        //отримуємо та перевіряємо покупця
        const buyer = await User.findById(buyerId);
        if (!buyer) {
            return res.status(404).json({ error: 'Покупець не знайдений' });
        }

        if (buyer.money < price) {
            return res.status(400).json({ error: 'Недостатньо грошей' });
        }
```

```
            const seller = await User.findById(userId);
            if (!seller) {
              return res.status(404).json({ error: 'Продавець не знайдений' });
            }

            //оновлюємо колекцію карток покупця та гроші
            buyer.money -= price;
            buyer.cardCollection.push({ pokemonId: cardId });

           //саме оновлення колоди та грошей продавця
            seller.money += price;
            seller.cardCollection = seller.cardCollection.filter(card => card.pokemonId !== cardId); //видаляємо вже придбану картку з колекції продавця
            seller.deck = seller.deck.filter(card => card.pokemonId !== cardId);
            //прибираємо пропозицію з бази даних
            await Offer.findByIdAndDelete(offerId); //видалю пропозицію після покупки

            //зберігаю оновлені дані покупця та продавця
            await buyer.save();
            await seller.save();

            res.json({
              message: 'Картка успішно придбана',
              purchasedCard: { pokemonId: cardId, price },
              buyerMoney: buyer.money,
              sellerMoney: seller.money
            });
          } catch (error) {
            console.error('Помилка при покупці ', error);
            res.status(500).json({ error: 'Не вийшло придбати картку' });
          }
        };

        //обмінятися картками
        const exchangeOffer = async (req, res) => {
          const { offerId, buyerId } = req.params;
          const { cardId, exchangeCardId } = req.body;

          try {
            //знаходимо пропозицію по offerId
            const offer = await Offer.findById(offerId).populate('userId'); //інфо  про продавця
            if (!offer) {
              return res.status(404).json({ error: 'Пропозиція не знайдена' });
            }

            const { userId, pokemonId, price, desiredCardId, desiredCardName } = offer;
            console.log(offer)
            console.log(desiredCardId)
            //якщо  покупець є продавцем
            if (buyerId === userId.toString()) {
              return res.status(400).json({ error: 'Не можна обмінятися з самим собою' });
            }

            //підтверджуємо покупця
            const buyer = await User.findById(buyerId);
            if (!buyer) {
              return res.status(404).json({ error: 'Покупець не знайдений' });
            }

            // перевіряємо чи в колекції покупця є потрібна картка для обміну
            if (desiredCardId !== 0 && !buyer.cardCollection.some(card => card.pokemonId === desiredCardId)) {
              return res.status(400).json({ error: 'В покупця немає необхідної картки' });
            }
```

```
      const seller = await User.findById(userId);
      if (!seller) {
         return res.status(404).json({ error: 'Подавець не знайдений' });
      }

      //обмін картками
      //видаляється карта з колекції покупця
      buyer.cardCollection = buyer.cardCollection.filter(card => card.pokemonId !== desiredCardId);
      buyer.deck = buyer.deck.filter(card => card.pokemonId !== desiredCardId);
      //додається картка від продавця в колекцію покупця
      buyer.cardCollection.push({ pokemonId: cardId });

      //видаляємо обміняну картку з колекції продавця
      seller.cardCollection = seller.cardCollection.filter(card => card.pokemonId !== cardId);
      seller.deck = seller.deck.filter(card => card.pokemonId !== cardId);

      seller.cardCollection.push({ pokemonId: desiredCardId });

      await buyer.save();
      await seller.save();

     //прибираю  пропозицію з бд
      await Offer.findByIdAndDelete(offerId);

      res.json({
         message: 'Обмін картками пройшов успішно',
         exchangedCards: [
           { pokemonId: cardId, exchangedFor: desiredCardId },
           { pokemonId: desiredCardId, exchangedFor: cardId }
         ],
         buyerCardCollection: buyer.cardCollection,
         sellerCardCollection: seller.cardCollection
      });
  } catch (error) {
      console.error('Помилка при обміні картками ', error);
      res.status(500).json({ error: 'Не вийшло обмінятися картками' });
  }
};

//видалити пропозицію
const deleteOffer = async (req, res) => {
  const { offerId } = req.params;
  const { userId } = req.body;  //передає ідентифікатор користувача, що ввійшов у систему, у тілі запиту

  try {
    const offer = await Offer.findById(offerId);

    //и є користувач власником пропозиції
    if (offer.userId.toString() !== userId) {
      return res.status(403).json({ error: 'Ви не маєте права на видалення цієї пропозиції' });
    }

    //видалити пропозицію
    await Offer.findByIdAndDelete(offerId);

    res.status(200).json({ message: 'Пропозицію успішно видалено' });
  } catch (error) {
    console.error("Помилка при видаленні пропозиції", error);
    res.status(500).json({ error: 'Не вийшло видалити пропозицію' });
  }
}
```

```
  module.exports = {
    getShopSelection,
    purchaseCard,
    createOffer,
    getOffers,
    purchaseOffer,
    exchangeOffer,
    deleteOffer
  };

const mongoose = require('mongoose');

const offerSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  pokemonId: { type: Number, required: true },
  price: { type: Number, required: true },
  desiredCardId: {type: Number, default: 0},
  desiredCardName: {type: String, default: ''},
  createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Offer', offerSchema);

const mongoose = require('mongoose');

//схема для зберігання індивідуальних даних магазину
const shopSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User', //посилання на модель користувача
    required: true,
  },
  dailyCards: [
    {
      pokemonId: Number,
      price: Number,
    }
  ],
  purchasedCards: [
    {
      pokemonId: Number,
      price: Number,
    }
  ],
  lastUpdated: {
    type: Date,
    default: Date.now,
  }
});

module.exports = mongoose.model('Shop', shopSchema);

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    unique: true,
    required: true,
```

```
    },
    password: {
      type: String,
      required: true,
    },

    avatar: {
      type: String,
      default: '',
    },
    bio: {
      type: String,
      default: '',
    },
    location: {
      type: String,
      default: '',
    },
    favoritePokemon: {
      type: String,
      default: '',
    },
    money: {
      type: Number,
      default: 2000,
    },
    rating: {
      type: Number,
      default: 0,
    },
    cardCollection: [
      {
        pokemonId: Number,
      }
    ],

    deck: [
      {
        pokemonId: Number,
      }
    ],
  }, { timestamps: true });

  module.exports = mongoose.model('User', userSchema);

  const express = require('express');
  //функції з контролеру
  const {registerUser, loginUser, getProfile, getUser, getRating, updateProfile, updateUser} =
require('../controllers/AuthController')
  const {getShopSelection, purchaseCard, createOffer, getOffers, purchaseOffer, exchangeOffer, deleteOffer}
= require('../controllers/ShopController');
  const {saveDeck, getDeck} = require('../controllers/CardsController')
  const router = express.Router();

  //роути
  //get, post - методи що викликаються для цих роутів (запити до сервера)
  //параметри - шлях, функція що викликається при запиті до сервера

  router.post('/register', registerUser)
  router.post('/login', loginUser)
  router.get('/profile', getProfile)
  router.post('/profile', updateProfile);
  router.get('/rating', getRating)
```

```
router.post('/update-user-stats', updateUser);
router.get('/user/:userId', getUser);

router.get('/shop/:userId', getShopSelection);
router.post('/shop/:userId/purchase', purchaseCard);

router.get('/user-deck/:userId', getDeck);
router.post('/save-deck/:userId', saveDeck);

router.post('/offers/create', createOffer);
router.get('/offers', getOffers);
router.post('/offers/:offerId/purchase/:buyerId', purchaseOffer);
router.post('/offers/:offerId/exchange/:buyerId', exchangeOffer);
router.delete('/offers/:offerId', deleteOffer);

module.exports = router

const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

//шифрування паролю з використанням bcrypt
const encryptPassword = (password) => {
   return new Promise((resolve, reject) =>{
      bcrypt.genSalt(10, (err, salt) => {
         if(err){
            reject(err);
         }
         bcrypt.hash(password, salt, (err, hash) => {
            if(err){
               reject(err);
            }
            resolve(hash);
         })
      })
   })
}

//порівняння звичайного паролю з зашифрованим паролем
const comparePassword = (password, encrypted) => {
   return bcrypt.compare(password, encrypted)
}

//створення токену користувача з використанням його даних
//для подальшого відстеження, який користувач зараз авторизований та використовує додаток
const createToken = (payload, secret, options = {}) => {
   return new Promise((resolve, reject) => {
     jwt.sign(payload, secret, options, (error, token) => {
       if (error) {
         reject(error);
       } else {
         resolve(token);
       }
     });
   });
 };

module.exports = {encryptPassword, comparePassword, createToken}

const express = require('express');//для створення роутів для запросів до серверу
const mongoose = require('mongoose');//підключення до бази даних
const cookieParser = require('cookie-parser');
const { Server } = require('socket.io');
const http = require('http');
```

```
//можливість використовувати глобальні змінні
require('dotenv').config();

//ініціалізація додатку
const app = express();
const port = 3000;

//сервер для сокет io
const server = http.createServer(app);
const io = new Server(server, {
  cors: {
    origin: "http://localhost:5173", //підключення до фронтенд частини
    methods: ["GET", "POST", "DELETE"],
    credentials: true
  }
});

//дозволити обмін даними між клієнтом та сервером
const cors = require('cors');
app.use(cors({
  credentials: true,
  origin: "http://localhost:5173",
  methods: ["GET", "POST", "DELETE"],
}));

//підключення до бд
const connectionString = process.env.MONGO_URL;

mongoose.connect(connectionString)
  .then(() => {
    console.log('Database connection successful');
  })
  .catch(err => {
    console.error('Database connection error', err);
  });

//додаткові утиліти
app.use(express.json());
app.use(cookieParser());
app.use(express.urlencoded({ extended: false }));

//підключити роути
app.use('/', require('./routes/AuthRoutes'));


const players = {}; //підключені гравці

//функція коли гравець підключається
//всі функції всередині працюють доки гравець підключений
io.on("connection", (socket) => {
    console.log(`Player connected: ${socket.id}`);

    //при підключенні гравця його дані зберігаються
    socket.on("joinGame", (playerData) => {
        players[socket.id] = {
            id: socket.id,
            name: playerData.name,
            avatar: playerData.avatar,
            deck: playerData.deck,
            hand: [],
            placedCards: [],
            health: 500,
            turn: false
```

```
    };

    socket.join("gameRoom");

    //коли два гравці підключені почати гру
    if (Object.keys(players).length === 2) {
        const [player1, player2] = Object.keys(players);
        io.to("gameRoom").emit("playersReady", { player1, player2 });

        players[player1].turn = true; //першим ходить той хто першим підключився

        //на початку гри надіслати гравцям дані противника
        io.to(player1).emit("startGame", {
            opponentData: {
                name: players[player2].name,
                avatar: players[player2].avatar,
                deck: players[player2].deck
            },
            turn: true
        });
        io.to(player2).emit("startGame", {
            opponentData: {
                name: players[player1].name,
                avatar: players[player1].avatar,
                deck: players[player1].deck
            },
            turn: false
        });
    }
});

//коли гравець набирає початкову "руку" з картками
socket.on('drawHand', ({hand}) => {
    if (players[socket.id]) {
      players[socket.id].hand = hand;
  }
  const opponentId = Object.keys(players).find(id => id !== socket.id);
  io.to(opponentId).emit("opponentDrawHand", { hand });
})

//коли гравець кладе картку на поле
socket.on("playerPlaceCard", ({ index, card }) => {

  if (players[socket.id]) {
      players[socket.id].placedCards[index] = card;
  }
  console.log(card)
  //надіслати противнику інформацію про розміщену гравцем картку
  const opponentId = Object.keys(players).find(id => id !== socket.id);
  io.to(opponentId).emit("opponentPlaceCard", { index: index, card: card });

});

//коли гравець бере картку
socket.on("playerDraw", (drawData) => {
  const opponentId = Object.keys(players).find(id => id !== socket.id);
  players[socket.id].hand.push(drawData.card);

  socket.to(opponentId).emit("opponentDraw", { card: drawData.card });
  //socket.emit("updateTurn", { turn: false });
});

//коли гравець атакує (в кінці кожного ходу)
```

```
    socket.on("playerAttack", (attackData) => {
      const opponentId = Object.keys(players).find(id => id !== socket.id);
      players[opponentId].placedCards = attackData.cards;
      players[opponentId].health = attackData.health;

      io.to(opponentId).emit("opponentAttack", { health: attackData.health, pokemonHealth: attackData.pokemon, cards: attackData.cards});
      //socket.emit("updateTurn", { turn: false });
    });

    //функція для передачі ходу
    socket.on('switchTurn', ({ turn }) => {
      const opponentId = Object.keys(players).find(id => id !== socket.id);

      players[socket.id].turn = !turn;
      players[opponentId].turn = turn;

      //надіслати противнику інформацію що зараз його хід
      socket.to(opponentId).emit("updateTurn", { turn: turn });
    });

    //коли гра завершується
    socket.on('gameOver', ({won}) => {
      const opponentId = Object.keys(players).find(id => id !== socket.id);
      socket.to(opponentId).emit("gameOver", { won: won });
    })

    //якщо хтось з гравців відключився
    socket.on("disconnect", () => {
      console.log(`Player disconnected: ${socket.id}`);
      delete players[socket.id];

      //повідомити противнику про відключення другого гравця
      socket.to("gameRoom").emit("playerDisconnected", socket.id);
    });
});


//запустити сервер
server.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

ДОДАТОК Б

Код фронтенду додатку

```
import axios from "axios";
import { createContext, useState, useEffect } from "react";

//зберігає дані користувача що зараз авторизований
export const UserContext = createContext({});

export function UserContextProvider({ children }) {
   const [user, setUser] = useState(null);
   const [loading, setLoading] = useState(true);
   const [error, setError] = useState(null);

   useEffect(() => {
      const fetchUser = async () => {
         try {
            const response = await axios.get('/profile');
            if (response.status === 200) {
               setUser(response.data); //зберігти дані користувача
            } else {
               setUser(null);
            }
         } catch (error) {
            console.error('Помилка при завантаженні даних', error);
            setUser(null);
            setError('Не вийшло завантажити дані користувача');
         } finally {
            setLoading(false);
         }
      };

      fetchUser();
   }, []);

   return (
      <UserContext.Provider value={{ user, setUser, loading, error }}>
         {children}
      </UserContext.Provider>
   );
}


import React, { useState, useContext, useEffect } from 'react';
import axios from 'axios';
import '../css/index.css';
import { UserContext } from '../../context/userContext';
import PlayingCard from './PlayingCard';
import SearchBar from '../components/SearchBar';

const CreateOffer = ({ refreshOffers }) => {
  const { user } = useContext(UserContext);
  const [userCards, setUserCards] = useState([]);
  const [allPokemon, setAllPokemon] = useState([]);
  const [filteredPokemon, setFilteredPokemon] = useState([]);
  const [selectedCard, setSelectedCard] = useState(null);
  const [desiredCard, setDesiredCard] = useState({ id: 0, name: '' });
```

```
  const [price, setPrice] = useState('');
  const [error, setError] = useState(null);
  const [isModalOpen, setModalOpen] = useState(false);
  const [searchTerm, setSearchTerm] = useState('');

  useEffect(() => {
   //завантажити колекцію карток користувача
   const loadUserCards = async () => {
    if (user && user._id) {
     try {
      const userCards = user.cardCollection ? user.cardCollection.map(card => card.pokemonId) : [];
      setUserCards(userCards);
     } catch (error) {
      console.error("Помилка при завантажені карток", error);
      setError('Не вицшло завантажики картки');
     }
    }
   };

   //завантажити дані про покемонів для коректного відображення карток користувача та списку всіх
карток
   const fetchAllPokemon = async () => {
    try {
     const response = await axios.get('https://pokeapi.co/api/v2/pokemon?limit=151', {
      baseURL: '',
      withCredentials: false,
     });
     const pokemonData = response.data.results.map((pokemon, index) => ({
      id: index + 1,
      name: pokemon.name,
     }));
     setAllPokemon(pokemonData);
     setFilteredPokemon(pokemonData);
    } catch (error) {
     console.error('Помилка при завантаженні даних покемонів', error);
    }
   };

   loadUserCards();
   fetchAllPokemon();
  }, [user]);

  useEffect(() => {
   //фільтрування списку всіх карток за іменем покемона
   const filtered = allPokemon.filter((pokemon) =>
    pokemon.name.toLowerCase().includes(searchTerm.toLowerCase())
   );
   setFilteredPokemon(filtered);
  }, [searchTerm, allPokemon]);

  //обробник натискання на картку
  const handleCardSelect = (pokemonId) => {
   setSelectedCard(pokemonId);
  };

  //збереження бажаної картки для обміну
  const handleDesiredCardSelect = (pokemonId) => {
   const selectedPokemon = allPokemon.find((pokemon) => pokemon.id === pokemonId);
   if (selectedPokemon) {
    setDesiredCard({ id: pokemonId, name: selectedPokemon.name });
   } else {
    setDesiredCard({ id: 0, name: '' });
   }
```

```
};

//збереження пропозиції
const handleOfferSubmit = async (e) => {
  e.preventDefault();
  if (!selectedCard || !price) return;

  //дані про пропозицію
  try {
    const offerData = {
      userId: user._id,
      pokemonId: selectedCard,
      price: Number(price),
      desiredCardId: desiredCard.id,
      desiredCardName: desiredCard.name
    };
    //створити пропозицію
    await axios.post('/offers/create', offerData);

    //оновити стлрінку з пропозиціями
    await refreshOffers();
    setSelectedCard(null);
    setPrice('');
    setError(null);
    setModalOpen(false);
  } catch (error) {
    console.error('Помилка при створення пропозиції', error);
    setError('Не вийшло створити пропозицію');
  }
};

return (
  <div>
    <button onClick={() => setModalOpen(true)}>List a Card for Sale</button>

    {isModalOpen && (
      <div className="modal-overlay" onClick={() => setModalOpen(false)}>
        <div className="modal-content" onClick={(e) => e.stopPropagation()}>
          <button className="modal-close" onClick={() => setModalOpen(false)}>×</button>
          <h2>Create Offer</h2>
          <h3>Select card from your collection that you want to trade</h3>
          <form onSubmit={handleOfferSubmit}>
            <h3>Your Cards</h3>
            <div className="cards-grid">
              {userCards.length > 0 ? (
                userCards.map((card) => (
                  <div
                    key={card}
                    className={`card-item ${selectedCard === card ? 'selected' : ''}`}
                    onClick={() => handleCardSelect(card)}
                  >
                    <PlayingCard pokemonId={card} />
                  </div>
                ))
              ) : (
                <p>No cards in your collection.</p>
              )}
            </div>

            <h3>Choose desired card to trade</h3>
            <SearchBar searchTerm={searchTerm} setSearchTerm={setSearchTerm} />

            <select
```

```
            onChange={(e) => handleDesiredCardSelect(Number(e.target.value))}
            value={desiredCard.id || ''}
          >
            <option value="" disabled>Select Pokémon</option>
            {filteredPokemon.length > 0 ? (
              filteredPokemon.map((pokemon) => (
                <option key={pokemon.id} value={pokemon.id}>
                  {pokemon.name} (ID: {pokemon.id})
                </option>
              ))
            ) : (
              <option disabled>No Pokémon found</option>
            )}
          </select>
          <h3>Set a price for your card</h3>
          <input
            type="number"
            placeholder="Price"
            value={price}
            onChange={(e) => setPrice(e.target.value)}
            required
          />
          <button type="submit">List for Sale</button>
          {error && <div className="error">{error}</div>}
        </form>
      </div>
    </div>
  )}
 </div>
);
};

export default CreateOffer;

import { useState, useEffect } from 'react';
import axios from 'axios';
import { toast } from 'react-hot-toast';
import { Link } from 'react-router-dom';
import '../css/index.css';

function DataTable() {
  const [users, setData] = useState([]);

  useEffect(() => {
    const fetchUsers = async () => {
      try {
        const { data } = await axios.get('/rating');
        if (data.error) {
          toast.error(data.error);
        } else {
          setData(data);
        }
      } catch (error) {
        toast.error('Помилка завантаження користувачів');
      }
    };

    fetchUsers();
  }, []);

  return (
    <div className="dashboard-container">
      <h1>User Rating</h1>
```

```
            <div className="table-container">
              <table>
                <thead>
                  <tr>
                    <th>Rank</th>
                    <th>Name</th>
                    <th>Rating</th>
                  </tr>
                </thead>
                <tbody>
                  {users.map((user, index) => (
                    <tr key={user._id}>
                      <td>{index + 1}</td>
                      <td>
                        <Link to={`/user/${user._id}`} className="user-link">
                          <span className="tooltip">
                            {user.name}
                            <span className="tooltiptext">See Profile</span>
                          </span>
                        </Link>
                      </td>
                      <td>{user.rating}</td>
                    </tr>
                  ))}
                </tbody>
              </table>
            </div>
          </div>
        );
}

export default DataTable;

/* eslint-disable react/prop-types */
import Modal from './Modal';
import axios from 'axios';

const EndGameModal = ({ isOpen, onClose, didWin, opponentDisconnected, userId, userRating }) => {
  let title, message, rewardAmount, newRating;

  if (opponentDisconnected) {
    title = "Opponent Disconnected!";
    message = "Your opponent left the game. You win by default. \n Your reward is 100$";
    rewardAmount = 100;
    newRating = userRating;
  } else if (didWin) {
    title = "Congratulations! You Won!";
    message = "Great job defeating your opponent! \n Your reward is 200$ and +10 to rating";
    rewardAmount = 200;
    newRating = userRating + 10;
  } else {
    title = "Game Over! You Lost!";
    message = "Better luck next time! \n Your reward is 50$";
    rewardAmount = 50;
    newRating = userRating;
  }

  const handleClose = async () => {
    try {
      await axios.post('/update-user-stats', { userId, rewardAmount, newRating });
      onClose(); //закрити модальне вікно
    } catch (error) {
      console.error('Помилка при оновлені даних', error);
```

```
    }
  };

  return (
    <Modal isOpen={isOpen} onClose={onClose}>
      <h2>{title}</h2>
      <p>{message}</p>
      <button onClick={handleClose}>Close</button>
    </Modal>
  );
};

export default EndGameModal;

/* eslint-disable react/prop-types */
import '../css/Modal.css';

const Modal = ({ isOpen, onClose, children }) => {
  if (!isOpen) return null;

  return (
    <div className="modal-overlay">
      <div className="modal-content">
        <button className="close-button" onClick={onClose}>×</button>
        {children}
      </div>
    </div>
  );
};

export default Modal;

import { Link, useNavigate } from "react-router-dom";
import { useContext, useState } from "react";
import { UserContext } from "../../context/userContext";
import "../css/Navbar.css";

export default function Navbar() {
  const { user, setUser } = useContext(UserContext);
  const [showProfileMenu, setShowProfileMenu] = useState(false);
  const navigate = useNavigate();

  //показувати чи закривати панель користувача
  const handleMouseEnter = () => setShowProfileMenu(true);
  const handleMouseLeave = () => setShowProfileMenu(false);

  //вихід з акаунту
  const handleLogout = () => {
    setUser(null);
    localStorage.removeItem("userToken");

    //очистити кукі
    const cookies = document.cookie.split(";");
    for (let cookie of cookies) {
      const eqPos = cookie.indexOf("=");
      const name = eqPos > -1 ? cookie.substr(0, eqPos) : cookie;
      document.cookie = name + "=;expires=Thu, 01 Jan 1970 00:00:00 GMT";
    }

    navigate("/login"); //перенаправлення на сторінку логіну
  };
```

```
    return (
      <nav className="navbar">
        <div className="nav-left">
          <Link to={'/rules'} className="nav-link">Rules</Link>
          <Link to={'/dashboard'} className="nav-link">Dashboard</Link>
          <Link to={'/game'} className="nav-link">Game</Link>
          <Link to={'/library'} className="nav-link">Library</Link>
          <Link to={'/shop'} className="nav-link">Shop</Link>
          <Link to={'/marketplace'} className="nav-link">Marketplace</Link>
        </div>
        <div className="nav-right">
          {user ? (
            <div
              className="profile-menu"
              onMouseEnter={handleMouseEnter}
              onMouseLeave={handleMouseLeave}
            >
              <Link to="/profile" className="nav-link">Profile</Link>
              {showProfileMenu && (
                <div className="profile-dropdown">
                  <img
                    src={user.avatar || 'https://cdn-icons-png.flaticon.com/128/18020/18020062.png'}
                    alt={user.name}
                    className="avatar"
                  />
                  <p><strong>{user.name}</strong></p>
                  <p>{user.email}</p>
                  <Link to='/profile' className="dropdown-link">Profile</Link>
                  <Link to="/profile-edit" className="dropdown-link">Edit Profile</Link>
                  <Link to="/profile-cards" className="dropdown-link">Card Collection</Link>
                  <button onClick={handleLogout} className="dropdown-link logout-
btn">Logout</button>
                </div>
              )}
            </div>
          ) : (
            <>
              <Link to={'/login'} className="nav-link">Login</Link>
              <Link to={'/register'} className="nav-link">Register</Link>
            </>
          )}
        </div>
      </nav>
    );
  }

  /* eslint-disable react/prop-types */
  import { useState, useEffect } from 'react';
  import axios from 'axios';
  import '../css/PlayingCard.css';

  const PlayingCard = ({ pokemonId, isInGame=false, currentHP=undefined }) => {
    const [pokemonData, setPokemonData] = useState(null);
    const [pokemonSpecies, setPokemonSpecies] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(false);

    useEffect(() => {
      const fetchPokemonData = async () => {
        try {
          //отримати дані про покемонів
          const response = await axios.get(`https://pokeapi.co/api/v2/pokemon/${pokemonId}`, {
            baseURL: '',
```

```
      withCredentials: false,
    });
    setPokemonData(response.data);

    //для отримання короткого опису
    const speciesResponse = await axios.get(response.data.species.url, {
      baseURL: '',
      withCredentials: false,
    });
    setPokemonSpecies(speciesResponse.data);

      setLoading(false);
    } catch (error) {
      console.error('Помилка завантаження даних:', error);
      setError(true);
      setLoading(false);
    }
  };

  fetchPokemonData();
}, [pokemonId]);

if (loading) {
  return <div className="loading">Loading...</div>;
}

if (error || !pokemonData || !pokemonSpecies) {
  return <div className="error">Error loading data!</div>;
}

//характеристики
const stats = {
  hp: currentHP !== undefined ? currentHP : pokemonData.stats[0].base_stat,
  attack: pokemonData.stats[1].base_stat,
  defense: pokemonData.stats[2].base_stat,
};

//опис
const descriptionEntry = pokemonSpecies.flavor_text_entries.find(
  (entry) => entry.language.name === 'en'
);
const description = descriptionEntry ? descriptionEntry.flavor_text : 'No description available.';

return (
  <div className={`playing-card ${isInGame ? 'ingame' : ''}`}>
    <div className="card-header">
      <h2 className="pokemon-name">{pokemonData.name}</h2>
      <img
        src={pokemonData.sprites.front_default}
        alt={pokemonData.name}
        className="pokemon-image"
      />
    </div>

    <div className={`card-stats top-left ${isInGame ? 'ingame' : ''}`}>
      <div className="stat-circle attack">{stats.attack}</div>
    </div>
    <div className={`card-stats top-right ${isInGame ? 'ingame' : ''}`}>
      <div className="stat-circle defense">{stats.defense}</div>
    </div>
    <div className={`card-stats bottom-right ${isInGame ? 'ingame' : ''}`}>
      <div className="stat-circle hp">{stats.hp}</div>
    </div>
```

```
        {!isInGame && (
          <div className="pokemon-description">
            <p>{description}</p>
          </div>
        )}
      </div>
    );
  };

  export default PlayingCard;

  import { useState, useContext } from 'react';
  import { UserContext } from "../../context/userContext";
  import axios from 'axios';
  import '../css/ProfileEditForm.css'; // Import the CSS file for styling
  import { useNavigate } from "react-router-dom";
  import {toast} from 'react-hot-toast';

  const ProfileEditForm = () => {
    const { user, setUser } = useContext(UserContext);
    const [bio, setBio] = useState(user.bio);
    const [location, setLocation] = useState(user.location);
    const [favoritePokemon, setFavoritePokemon] = useState(user.favoritePokemon);
    const [avatar, setAvatar] = useState(user.avatar);
    const navigate = useNavigate();
    const handleSave = async () => {
      try {
        const response = await axios.post('/profile', {
          userId: user._id,
          bio,
          location,
          favoritePokemon,
          avatar,
        });

        //оновити контекст користувача з оновленими даними
        setUser(response.data);
        toast.success('Профіль успішно оновлено');
        navigate('/profile')
      } catch (error) {
        console.error('Помилка оновлення профілю', error);
        toast.error('Не вийшло оновити профіль');
      }
    };

    return (
      <div className="profile-edit-form">
        <h2>Edit Profile</h2>
        <div className="form-group">
          <label>Avatar URL</label>
          <input
            type="text"
            value={avatar}
            onChange={(e) => setAvatar(e.target.value)}
            placeholder="Enter avatar URL"
          />
        </div>
        <div className="form-group">
          <label>Bio</label>
          <textarea value={bio} onChange={(e) => setBio(e.target.value)} placeholder="Tell us about
yourself..." />
        </div>
        <div className="form-group">
```

```
            <label>Location</label>
            <input value={location} onChange={(e) => setLocation(e.target.value)} placeholder="Your location"
/>
          </div>
          <div className="form-group">
            <label>Favorite Pokémon</label>
            <input value={favoritePokemon} onChange={(e) => setFavoritePokemon(e.target.value)}
placeholder="Your favorite Pokémon" />
          </div>
          <button className="save-button" onClick={handleSave}>Save Changes</button>
        </div>
      );
    };

    export default ProfileEditForm;

    /* eslint-disable react/prop-types */

    //поле для пошуку за іменем покумона
    const SearchBar = ({ searchTerm, setSearchTerm }) => {
      return (
        <input
          type="text"
          placeholder="Search Pokemon by name"
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
          style={{ marginBottom: '20px', padding: '10px', width: '100%' }}
        />
      );
    };

    export default SearchBar;

    /* eslint-disable react-hooks/rules-of-hooks */
    /* eslint-disable react/prop-types */
    import { useNavigate } from "react-router-dom";

    const StartGameModal = ({ isOpen, onSelectGameType, userDeck }) => {
      if (!isOpen) return null;
      const navigation = useNavigate();
      return (
        <div className="modal-overlay">
          {userDeck ? (
            <div className="modal-content">
              <h2>Select Game Mode</h2>
              <button onClick={() => onSelectGameType("pvp")}>Player vs Player</button>
              <button onClick={() => onSelectGameType("computer")}>Play vs Computer</button>
            </div>
          ) : (
            <div className="modal-content">
              <h2>Your deck is empty!</h2>
              <button onClick={() => navigation('/profile-cards')}>Fill your deck</button>
            </div>
          )}
        </div>
      );
    };


    export default StartGameModal;

    /* eslint-disable react/prop-types */
    const UserInfo = ({ user }) => {
```

```
        return (
            <div className="profile-page">
                <div className="profile-header">
                    <img
                        src={user.avatar || 'https://cdn-icons-png.flaticon.com/128/18020/18020062.png'}
                        alt={user.name}
                        className="avatar"
                    />
                    <h1>{user.name}</h1>
                    <p className="profile-email">{user.email}</p>
                </div>
                <div className="profile-details">
                    <p><strong>Bio:</strong> {user.bio || 'No bio available'}</p>
                    <p><strong>Location:</strong> {user.location || 'No location provided'}</p>
                    <p><strong>Favorite Pokémon:</strong> {user.favoritePokemon || 'No favorite Pokémon listed'}</p>
                    <p><strong>Balance:</strong> {user.money || 'No money'}</p>
                    <p><strong>Rating:</strong> {user.rating || 'No rating'}</p>
                </div>
            </div>
        );
    };

    export default UserInfo;

    import toast from "react-hot-toast";

    export class GameLogic {
        constructor(socket, playerDeck, opponentDeck, turn, playerHealth = 500, opponentHealth = 500, isMultiplayer = true) {
            this.socket = socket;
            this.playerDeck = this.shuffleArray(playerDeck);
            this.opponentDeck = opponentDeck;
            this.playerHand = [];
            this.opponentHand = [];
            this.playerPlacedCards = Array(5).fill(null);
            this.opponentPlacedCards = Array(5).fill(null);
            this.playerHealth = playerHealth;
            this.opponentHealth = opponentHealth;
            this.turn = turn ? 'player' : 'opponent';
            this.isMultiplayer = isMultiplayer;
            this.playerPokémonHealth = {};
            this.opponentPokémonHealth = {};
            this.cardStats = {};
            this.readyToStart = false;

            console.log(turn)
            if (isMultiplayer) {
                //функції з реакцією на дії противники

                //противник взяв карту
                this.socket.on('opponentDraw', (drawData) => {
                    this.opponentDeck = drawData.deck;
                    this.opponentHand.push(drawData.card);
                    this.handleEndTurn();
                });

                //противник розмістив картку на полі
                this.socket.on('opponentPlaceCard', ({ index, card }) => {
                    console.log(index, card)
                    this.opponentPlacedCards[index] = card;
                    const ind = this.opponentHand.indexOf(card);
                    this.opponentHand.splice(ind, 1);
```

```
            this.handleEndTurn();
        });

        //противник атакував
        this.socket.on('opponentAttack', (attackData) => {
            console.log('attack')
            this.playerHealth = attackData.health;
            this.playerPokémonHealth = attackData.pokemonHealth;
            this.playerPlacedCards = attackData.cards;
            this.checkGameOver();
        });

        //противник набрав руку карток
        this.socket.on('opponentDrawHand', (hand) => {
            this.opponentHand = hand;
        })
        //оновити хід
        this.socket.on('updateTurn', (turn) => {
            this.turn = turn ? 'player' : 'opponent';
        })
    }
}

//почати гру
async initializeGame() {
    //набрати картки в руку
    for (let i = 0; i < 5; i++) {
        this.playerDrawCard();
        //якщо противник компютер набрати для нього картки
        if(!this.isMultiplayer) this.opponentDrawCard();
    }

    //зберігти характеристики покемонів для подальших розрахунків
    await this.initializePokémonStats(this.playerDeck, this.playerPokémonHealth);
    await this.initializePokémonStats(this.opponentDeck, this.opponentPokémonHealth);
    await this.initializePokémonStats(this.playerHand, this.playerPokémonHealth);
    await this.initializePokémonStats(this.opponentHand, this.opponentPokémonHealth);
}

//взяти картку
playerDrawCard() {
    if (this.playerHand.length >= 5) return false;

    const previousDeck = this.playerDeck;
    this.playerDeck = this.drawCard(this.playerDeck, this.playerHand);

    //викликати подію взяття картки
    const drawnCard = this.playerHand[this.playerHand.length - 1];
    if(this.isMultiplayer) this.socket.emit('playerDraw', { card: drawnCard });
    return previousDeck !== this.playerDeck;
}

//розмістити картку на полі
placePlayerCard(index, selectedCard) {
    if (this.playerPlacedCards[index] === null && selectedCard !== null) {
        this.playerPlacedCards[index] = selectedCard;
        this.playerHand = this.playerHand.filter(card => card !== selectedCard);

        //викликати подію розміщення картки
        if(this.isMultiplayer) this.socket.emit('playerPlaceCard', { index: index, card: selectedCard });
        return true;
    }
    return false;
```

```
}

//зберегти дані покемонів
async initializePokémonStats(hand, healthObj) {
    if (!Array.isArray(hand) || hand.length === 0) {
        return;
    }

    for (let card of hand) {
        const stats = await this.fetchPokemonStats(card);
        this.cardStats[card] = stats;
        healthObj[card] = stats.hp;
    }
}

//завантажити дані про покемонів
async fetchPokemonStats(pokemonId) {
    const response = await fetch(`https://pokeapi.co/api/v2/pokemon/${pokemonId}`);
    const data = await response.json();
    return {
        hp: data.stats.find(stat => stat.stat.name === "hp").base_stat,
        atk: data.stats.find(stat => stat.stat.name === "attack").base_stat,
        def: data.stats.find(stat => stat.stat.name === "defense").base_stat,
    };
}

//перемішати масив (колоду)
shuffleArray(array) {
    return array.sort(() => Math.random() - 0.5);
}

//взяти картку
drawCard(deck, hand) {
    if (deck.length > 0 && hand.length < 5) {
        const drawnCard = deck[0];
        hand.push(drawnCard);
        return deck.slice(1);
    }
    return deck;
}

//компютер бере картку
opponentDrawCard() {
    this.opponentDeck = this.drawCard(this.opponentDeck, this.opponentHand);
}

//компютер розміщає картку на полі
opponentPlaceCard() {
    const emptySlots = this.opponentPlacedCards
        .map((card, index) => card === null ? index : null)
        .filter(index => index !== null);

    if (emptySlots.length > 0 && this.opponentHand.length > 0) {
        const randomIndex = Math.floor(Math.random() * emptySlots.length);
        const slotToFill = emptySlots[randomIndex];
        const cardToPlace = this.opponentHand.shift();
        this.opponentPlacedCards[slotToFill] = cardToPlace;
    }
}

//обробник дії гравця
handlePlayerTurn(action, index = null, selectedCard = null) {
    if (action === 'draw') {
```

```
            const cardDrawn = this.playerDrawCard();
            if (!cardDrawn) {
                return false;
            }
            this.handleEndTurn();
            return true;
        } else if (action === 'place' && index !== null && selectedCard) {
            const placed = this.placePlayerCard(index, selectedCard);
            if (placed) {
                this.handleEndTurn();
                return true;
            }
        }
        return false;
    }

    //компютер обирає дію
    opponentTakeTurn() {
        if (this.checkGameOver()) return;

        const shouldDraw = this.opponentHand.length < 5 && Math.random() > 0.5;
        if (shouldDraw) {
            this.opponentDrawCard();
        } else {
            this.opponentPlaceCard();
        }
        this.handleEndTurn();
    }

    //підрахувати урон при атаках
    calculateDamage(attacker, defender) {
        const attackerStats = this.cardStats[attacker];
        const defenderStats = defender ? this.cardStats[defender] : null;

        let damage = 0;
        if (defenderStats) {
            damage = Math.max(0, attackerStats.atk - defenderStats.def);
        } else {
            damage = attackerStats.atk; //якщо навпроти не стоїть покемон атакувати опонента
        }

        return damage;
    }

    //атакувати противника
    playerAttack() {
        this.playerPlacedCards.forEach((card, index) => {
            if (card !== null) {
                const defender = this.opponentPlacedCards[index];
                const damage = this.calculateDamage(card, defender);
                if (defender) {
                    this.opponentPokémonHealth[defender] -= damage;
                    if (this.opponentPokémonHealth[defender] <= 0) {
                        this.opponentPlacedCards[index] = null;

                    }
                } else {
                    this.opponentHealth -= damage;
                }
                //викликати подію атаки
                if(this.isMultiplayer) this.socket.emit('playerAttack', {health: this.opponentHealth, pokemon: this.opponentPokémonHealth, cards: this.opponentPlacedCards });
            }
```

```
    });
}

//підрахувати урон
calculateTotalDamage() {
    let totalDamage = 0;
    this.playerPlacedCards.forEach((card, index) => {
        if (card !== null) {
            const defender = this.opponentPlacedCards[index];
            const damage = this.calculateDamage(card, defender);
            totalDamage += damage;
        }
    });
    return totalDamage;
}

//компютер атакує гравця
opponentAttack() {
    this.opponentPlacedCards.forEach((card, index) => {
        if (card !== null) {
            const defender = this.playerPlacedCards[index];
            const damage = this.calculateDamage(card, defender);
            if (defender) {
                this.playerPokémonHealth[defender] -= damage;
                if (this.playerPokémonHealth[defender] <= 0) {
                    this.playerPlacedCards[index] = null;
                }
            } else {
                this.playerHealth -= damage;
            }
        }
    });
}

//перевірити чи гра завершена
checkGameOver() {
    if (this.playerHealth <= 0) {
        toast.success("Opponent wins! You have been defeated.");
        return true;
    } else if (this.opponentHealth <= 0) {
        toast.success("You win! The opponent has been defeated.");
        return true;
    }
    return false;
}

//кінець ходу
handleEndTurn() {
    if (this.turn === 'player') {
        this.playerAttack();
        if (!this.checkGameOver()) {
            if (this.isMultiplayer) this.switchTurn(); //передати хід опоненту
            else {
                this.turn = 'opponent';
                this.opponentTakeTurn(); //якщо противник компютер він робить хід

            }
        }
    }
    else {
        if(!this.isMultiplayer){
            this.opponentAttack();
            this.turn = 'player';
```

```
            }
        }
    }

    //завершити свій хід та передати його іншому гравцю
    switchTurn() {
        this.turn = 'opponent';
        this.socket.emit('switchTurn', { turn: true });
    }

    isPlayerTurn() {
        return this.turn === 'player';
    }

    isOpponentTurn() {
        return this.turn === 'opponent';
    }
}

/* eslint-disable no-unused-vars */
import { useContext, useState, useEffect } from "react";
import { UserContext } from "../../context/userContext";
import axios from "axios";
import '../css/Profile.css';
import '../css/index.css';
import PlayingCard from '../components/PlayingCard';
import toast from "react-hot-toast";
const CardCollection = () => {
    const { user, loading, setUser } = useContext(UserContext);
    const [selectedCards, setSelectedCards] = useState([]);
    const maxDeckSize = 15;

    useEffect(() => {
        const loadUserDeck = async () => {
            try {
                if (user && user._id) {
                    //завантажити дані про колоду користувача
                    const response = await axios.get(`/user-deck/${user._id}`);
                    const deckIds = response.data.deck ? response.data.deck.map(card => card.pokemonId) : [];
                    setSelectedCards(deckIds);
                }
            } catch (error) {
                console.error("Помилка при завантаженні колоди", error);
            }
        };
        loadUserDeck();
    }, [user]);

    if (loading) {
        return <div className="loading">Loading...</div>;
    }

    if (!user) {
        return <div className="error">User profile not found.</div>;
    }

    //обробник натисканні на картку
    const handleCardSelect = (pokemonId) => {
        if (selectedCards.includes(pokemonId)) {
            setSelectedCards(selectedCards.filter((id) => id !== pokemonId)); //при повторному натисканні знати вибір з картки
        } else if (selectedCards.length < maxDeckSize) {
            setSelectedCards([...selectedCards, pokemonId]); //виділити обрану картку
```

```
        }
    };

    //зберегти колоду в бд
    const handleSaveDeck = async () => {
        try {
            const response = await axios.post(`/save-deck/${user._id}`, {
                deck: selectedCards,
            });
            //оновити дані поточного користувача
            const updatedUser = { ...user, deck: response.data.deck };
            setUser(updatedUser);
            toast.success('Колода успішно збережена');
        } catch (error) {
            console.error('Помилка  при збереженні колоди', error);
            toast.error('Не вийшло зберегти колоду');
        }
    };

    return (
        <div className="profile-page">
            <div className="profile-cards">
                <h2>Your Cards</h2>
                <div className="cards-grid">
                    {user.cardCollection.length > 0 ? (
                        user.cardCollection.map((card) => (
                            <div
                                key={card.pokemonId}
                                className={`card-item ${selectedCards.includes(card.pokemonId) ? 'selected' : ''}`}
                                onClick={() => handleCardSelect(card.pokemonId)}
                            >
                                <PlayingCard pokemonId={card.pokemonId} />
                            </div>
                        ))
                    ) : (
                        <p>No cards in your collection.</p>
                    )}
                </div>
                <button onClick={handleSaveDeck} disabled={selectedCards.length === 0 || selectedCards.length
> maxDeckSize}>
                    Save Deck ({selectedCards.length}/{maxDeckSize})
                </button>
            </div>
        </div>
    );
};

export default CardCollection;

import { useContext } from "react";
import { UserContext } from "../../context/userContext";
import DataTable from "../components/DataTable";
import '../css/index.css';

export default function Dashboard(){

    const { user, loading, error } = useContext(UserContext);

    if (loading) {
        return <div>Loading...</div>;
    }

    if (error) {
```

```
        return <div>{error}</div>;
    }
    //з контексту отримуємо ім'я поточного користувача й виводимо його ім'я
    return (
        <div>
            <h1>Dashboard</h1>
            {!!user && (<h2>Welcome, {user.name}!</h2>)}

            <DataTable></DataTable>

        </div>

    );
}

import { useContext } from "react";
import { UserContext } from "../../context/userContext";
import '../css/Profile.css';
import '../css/index.css';
import ProfileEditForm from '../components/ProfileEditForm';

const EditProfile = () => {
    const { user, loading } = useContext(UserContext);

    if (loading) {
        return <div className="loading">Loading...</div>;
    }

    if (!user) {
        return <div className="error">User profile not found.</div>;
    }

    //на сторінці лише форма для редагування даних
    return (
        <div className="profile-edit">
            <ProfileEditForm />
        </div>
    );
};

export default EditProfile;

/* eslint-disable react-hooks/exhaustive-deps */
/* eslint-disable no-unused-vars */
/* eslint-disable react/prop-types */
import { useContext, useState, useEffect, useRef } from "react";
import { io } from "socket.io-client";
import { UserContext } from "../../context/userContext";
import { GameLogic } from "../gameLogic/game";
import { useSpring, animated } from "react-spring";
import '../css/Game.css';
import PlayingCard from '../components/PlayingCard';
import StartGameModal from '../components/StartGameModal';
import EndGameModal from "../components/EndGameModal";
import toast from "react-hot-toast";

const Game = () => {
    const { user, loading } = useContext(UserContext);
    const [selectedCard, setSelectedCard] = useState(null);
    const [isAnimating, setIsAnimating] = useState(false);
    const [, setForceRender] = useState(0);
    const gameRef = useRef(null);
    const [maxHealth, setMaxHealth] = useState(200);
```

```
    const [cardSprings, setCardSprings] = useState([]);
    const [isStartModalOpen, setIsStartModalOpen] = useState(true);
    const [isEndModalOpen, setIsEndModalOpen] = useState(false);
    const [didWin, setDidWin] = useState(false);
    const [opponentDisconnected, setOpponentDisconnected] = useState(false);
    const [isWaiting, setIsWaiting] = useState(false);
    const [opponentTurn, setOpponentTurn] = useState(false);
    const [opponentName, setOpponentName] = useState('Computer');
    const [opponentAvatar, setOpponentAvatar] = useState('https://cdn-icons-
png.flaticon.com/128/18020/18020043.png');
    const socketRef = useRef(null);

    useEffect(() => {
      if (loading || !user) return;

      //відключити сокет при покиданні сторінки
      return () => {
        if (socketRef.current) socketRef.current.disconnect();
      };
    }, [loading, user]);

    //ініціальзація мультиплеєрної гри
    const initializeGame = async (opponentData, turn) => {
      const playerDeck = user.deck ? user.deck.map(card => card.pokemonId) : [];
      setOpponentTurn(turn);

      const newGame = new GameLogic(socketRef.current, playerDeck, opponentData.deck, turn);
      gameRef.current = newGame;
      setMaxHealth(newGame.playerHealth);

      //почати гру
      await newGame.initializeGame();
      forceUpdate();
    };

    //вибір типу гри
    const handleGameTypeSelection = (gameType) => {
      setIsStartModalOpen(false);

      if (gameType === "pvp") {
        startMultiplayerGame();
      } else {
        startSinglePlayerGame();
      }
    };

    //почати мультплеєрну гру
    const startMultiplayerGame = () => {
      //підключення сокета
      socketRef.current = io("http://localhost:3000");

      //підключення до гри
      socketRef.current.on("connect", () => {
        console.log("Connected to server");
        const playerDeck = user.deck ? user.deck.map(card => card.pokemonId) : [];
        socketRef.current.emit("joinGame", { userId: user.id, name: user.name, avatar: user.avatar, deck:
playerDeck });
      });

      //початок гри коли два гравці приєдналися
      socketRef.current.on("startGame", async ({ opponentData, turn }) => {
        console.log("Both players connected, starting game");
        console.log(opponentData)
```

```
        setOpponentName(opponentData.name);
        setOpponentAvatar(opponentData.avatar);
        await initializeGame(opponentData, turn);
        setIsWaiting(false);
        forceUpdate();
    });

    //передача хода
    socketRef.current.on('switchTurn', () => {
        forceUpdate();
    });

    //атака опонента
    socketRef.current.on('opponentAttack', async ()=>{
        forceUpdate();
        await animateCardsOnField(gameRef.current.opponentPlacedCards, 'opponent');
        setOpponentTurn(false);
        checkGameOver();
    })
    //відключення опонента
    socketRef.current.on("playerDisconnected", () => {
        if(!checkGameOver())
            setOpponentDisconnected(true);
        setDidWin(true);
        setIsEndModalOpen(true);
        socketRef.current.disconnect();
    });
    //завершення гри
    socketRef.current.on("gameOver", (won)=> {
        setDidWin(won);
        setIsEndModalOpen(true);
        socketRef.current.disconnect();
    })

    setIsWaiting(true); //екран очікування
};

//початок гри з компютером
const startSinglePlayerGame = () => {
    const playerDeck = user.deck ? user.deck.map(card => card.pokemonId) : [];
    const computerDeck = generateComputerDeck();

    setOpponentTurn(false);
    const newGame = new GameLogic(null, playerDeck, computerDeck, true, 500, 500, false);
    gameRef.current = newGame;
    setMaxHealth(newGame.playerHealth);

    //початок гри
    newGame.initializeGame();
    forceUpdate();
};

const generateComputerDeck = () => {
    //генерація колоди компютера
    return Array.from({ length: 5 }, () => Math.floor(Math.random() * 151) + 1);
};

//примусове оновлення всіх елементів екрану
const forceUpdate = () => setForceRender(prev => prev + 1);

//обробник натискання на картку
const handleCardClick = (card) => {
    const game = gameRef.current;
```

```
    console.log(game.isPlayerTurn())
    if (game && game.isPlayerTurn()) {
        setSelectedCard(selectedCard === card ? null : card);
    }
};

//призупинка всіх дій на секунду для імітації реалістичності
const animateAction = async () => {
    setIsAnimating(true);
    await new Promise(resolve => setTimeout(resolve, 1000));
    setIsAnimating(false);
};

//переміщення карти з руки на поле
const moveCardToPlayZone = async (index) => {
    const game = gameRef.current;
    if (selectedCard && game.isPlayerTurn() && game.playerPlacedCards[index] === null) {
        await animateAction();
        const cardPlaced = game.placePlayerCard(index, selectedCard);
        if (cardPlaced) {
            setSelectedCard(null);
            forceUpdate();
            handleEndTurn();
        }
    }
};

//обробник взяття картки гравцем
const handleTurn = async (action) => {
    const game = gameRef.current;
    if (game.isPlayerTurn() && action === 'draw') {
        await animateAction();
        const cardDrawn = game.handlePlayerTurn('draw');
        if (cardDrawn) {
            forceUpdate();
            handleEndTurn();
        } else {
            toast.error("Не можна взяти більше карток!");
        }
    }
};

//обробник завершення ходу
const handleEndTurn = async () => {
    const game = gameRef.current;
    //if (!game.isPlayerTurn()) return;

    forceUpdate();
    setIsAnimating(true);
    setOpponentTurn(true);
    await animateAction();
    await animateCardsOnField(game.playerPlacedCards, 'player');

    game.handleEndTurn();
    forceUpdate();
    checkGameOver();
    if(!game.isMultiplayer){
      await animateCardsOnField(game.opponentPlacedCards, 'opponent');
      setOpponentTurn(false);
    }
    setIsAnimating(false);
};
```

```
//виклик анімації атак карток
const animateCardsOnField = async (cards, type) => {
   const occupiedSlots = cards.map((card, index) => (card ? index : null)).filter(index => index !== null);
   if (occupiedSlots.length === 0) return;

   for (let index of occupiedSlots) {
      await animateCard(index, type === 'player' ? -30 : 30);
   }
};

//анімація карток
const animateCard = async (index, yTranslate) => {
   await new Promise(resolve => {
      setCardSprings(prev => {
         const updated = [...prev];
         updated[index] = { transform: `translateY(${yTranslate}px)` };
         return updated;
      });
      setTimeout(resolve, 500);
   });

   await new Promise(resolve => {
      setCardSprings(prev => {
         const updated = [...prev];
         updated[index] = { transform: 'translateY(0)' };
         return updated;
      });
      setTimeout(resolve, 500);
   });
};

//перевірити чи гра завершена
const checkGameOver = () => {
   const game = gameRef.current;
   if (game.playerHealth <= 0 || game.opponentHealth <= 0) {
      handleEndGame(game.playerHealth > 0);
      return true;
   }
   else return false;
};
//початок гри
const handleStartGame = () => {
   setIsStartModalOpen(false);
   socketRef.current.emit("startGame", { userId: user.id });
};

//завершення гри
const handleEndGame = (won) => {

   setDidWin(won);
   setOpponentDisconnected(false);
   setIsEndModalOpen(true);
   if(gameRef.current.isMultiplayer) socketRef.current.emit('gameOver', {win: !won});
};

//закрити вікно з інформацією про кінець гри
const closeEndGameModal = () => {
   setIsEndModalOpen(false);
   setIsStartModalOpen(true); //відкрити вікно вибору типу нової гри
};

if (loading) return <div>Loading...</div>;
```

```
        if (!user) return <div>Error loading game or user data.</div>;

        return (
          <div className="game-container">
            {isStartModalOpen ? (
              <StartGameModal
                isOpen={isStartModalOpen}
                userDeck={user.deck.length > 0}
                onSelectGameType={handleGameTypeSelection}
              />
            ) : isWaiting ? (
              <div className="waiting-screen">
                <h2>Waiting for another player to connect...</h2>
              </div>
            ) : isEndModalOpen ? (
              <EndGameModal
                  isOpen={isEndModalOpen}
                  onClose={closeEndGameModal}
                  didWin={didWin}
                  opponentDisconnected={opponentDisconnected}
                  userId={user._id}
                  userRating={user.rating}
                />
            ) : (
              <div className="game-container">

                <div className="opponent-section">
                  <div className="deck-area opponent-deck">
                    <img                src="https://tcg.pokemon.com/assets/img/global/tcg-card-back-2x.jpg"
alt="Opponent Deck" className="deck" />
                  </div>
                  <div className="player-info">
                    <img src={opponentAvatar} alt={opponentName} className="avatar" />
                    <h2 className="player-name">{opponentName}</h2>
                    <div className="health-container">

                      <div className="health">

                        <div className="health-inner" style={{ width: `${(gameRef.current.opponentHealth
/ maxHealth) * 100}%` }}></div>
                      </div>
                      <span          className="health-label">{Math.max(gameRef.current.opponentHealth,
0)}</span>
                    </div>
                  </div>
                  <div className="hand opponent-hand">
                    {gameRef.current.opponentHand.map((card, index) => (
                      <div className="card-slot" key={index}>
                        <img          src="https://tcg.pokemon.com/assets/img/global/tcg-card-back-2x.jpg"
alt="Opponent Card"/>
                      </div>
                    ))}
                  </div>
                </div>

                <div className="card-placement-area">
                  <div className="opponent-field">
                    {gameRef.current.opponentPlacedCards.map((card, index) => (
                      <animated.div
                        className={`card-slot ${card ? 'occupied' : ''}`}
                        key={index}
                        style={cardSprings[index] || { transform: 'translateY(0)' }}
                      >
```

```
                                {card ? <PlayingCard pokemonId={card} isInGame={true} currentHP={gameRef.current.opponentPokémonHealth[card]}/> : <div className="empty-slot" />}
                                </animated.div>
                            ))}
                        </div>
                    <div className="turn-indicator">
                        <h2 className={opponentTurn ? 'opponent-turn' : 'your-turn'}>
                            {opponentTurn ? "Opponent's Turn" : "Your Turn"}
                        </h2>
                    </div>
                        <div className="player-field">
                            {gameRef.current.playerPlacedCards.map((card, index) => (
                                <animated.div
                                    className={`card-slot ${card ? 'occupied' : ''}`}
                                    key={index}
                                    style={cardSprings[index] || { transform: 'translateY(0)' }}
                                    onClick={() => moveCardToPlayZone(index)}
                                >
                                    {card ? <PlayingCard pokemonId={card} isInGame={true} currentHP={gameRef.current.playerPokémonHealth[card]}/> : <div className="empty-slot" />}
                                </animated.div>
                            ))}
                        </div>
                    </div>

                    <div className="user-section">
                        <div className="hand user-hand">
                            {gameRef.current.playerHand.map((card, index) => (
                                <div className={`card-slot ${selectedCard === card ? 'selected' : ''}`} key={index} onClick={() => handleCardClick(card)}>
                                    <PlayingCard pokemonId={card} isInGame={true}/>
                                </div>
                            ))}
                        </div>
                        <div className="player-info">
                            <img src={user.avatar} alt={user.name} className="avatar" />
                            <h2 className="player-name">{user.name}</h2>
                            <div className="health-container">
                                <div className="health">
                                    <div className="health-inner" style={{ width: `${(gameRef.current.playerHealth / maxHealth) * 100}%` }}></div>
                                </div>
                                <span                 className="health-label">{Math.max(gameRef.current.playerHealth, 0)}</span>
                            </div>
                        </div>
                        <div className="deck-area user-deck">
                            <img  src="https://tcg.pokemon.com/assets/img/global/tcg-card-back-2x.jpg"  alt="Player Deck" className="deck" />
                        </div>
                    </div>
                    <div className="action-buttons">
                        <button onClick={() => handleTurn('draw')} disabled={isAnimating || opponentTurn}>
                            Draw Card
                        </button>
                        <button onClick={() => handleEndTurn()} disabled={isAnimating || opponentTurn}>
                            End Turn
                        </button>
                    </div>
                </div>
            )}
        </div>
    );
```

```
};

export default Game;

//import React from "react";

export default function Home(){
   return (
      <div>Home</div>
   )
}

/* eslint-disable no-unused-vars */
import React, { useState, useEffect } from 'react';
import PlayingCard from '../components/PlayingCard';
import SearchBar from '../components/SearchBar';
import '../css/index.css';
import axios from 'axios';

const Library = () => {
  const [pokemonList, setPokemonList] = useState([]);
  const [filteredList, setFilteredList] = useState([]);
  const [currentPage, setCurrentPage] = useState(1);
  const [searchTerm, setSearchTerm] = useState('');
  const cardsPerPage = 20;
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchPokemonList = async () => {
      try {
        //завантажити дані про покемонів
        const response = await axios.get('https://pokeapi.co/api/v2/pokemon?limit=151', {
          baseURL: '',
          withCredentials: false,
        });
        const pokemonData = response.data.results.map((pokemon, index) => ({
          id: index + 1,
          name: pokemon.name,
        }));
        setPokemonList(pokemonData);
        setFilteredList(pokemonData);
        setLoading(false);
      } catch (error) {
        console.error('Помилка при завантаженні даних', error);
        setLoading(false);
      }
    };

    fetchPokemonList();
  }, []);

  useEffect(() => {
    //фільтрувати покемонів за іменем
    const filtered = pokemonList.filter((pokemon) =>
      pokemon.name.toLowerCase().includes(searchTerm.toLowerCase())
    );
    setFilteredList(filtered);
    setCurrentPage(1);
  }, [searchTerm, pokemonList]);

  //розподілення карток по сторінкам
  const indexOfLastCard = currentPage * cardsPerPage;
  const indexOfFirstCard = indexOfLastCard - cardsPerPage;
```

```
  const currentCards = filteredList.slice(indexOfFirstCard, indexOfLastCard);

  const nextPage = () => {
    if (currentPage < Math.ceil(filteredList.length / cardsPerPage)) {
      setCurrentPage(currentPage + 1);
    }
  };

  const prevPage = () => {
    if (currentPage > 1) {
      setCurrentPage(currentPage - 1);
    }
  };

  if (loading) {
    return <div className="loading">Loading Pokémon Library...</div>;
  }

  return (
    <div className="pokemon-library">
      <h1>Pokémon Library (First Generation)</h1>
      <SearchBar searchTerm={searchTerm} setSearchTerm={setSearchTerm} />
      <div className="cards-grid">
        {currentCards.map((pokemon) => (
          <PlayingCard key={pokemon.id} pokemonId={pokemon.id} />
        ))}
      </div>
      <div className="pagination">
        <button onClick={prevPage} disabled={currentPage === 1}>Previous</button>
        <span> Page {currentPage} of {Math.ceil(filteredList.length / cardsPerPage)} </span>
        <button onClick={nextPage} disabled={currentPage === Math.ceil(filteredList.length / cardsPerPage)}>Next</button>
      </div>
    </div>
  );
};

export default Library;

import {useState} from "react";
import axios from "axios";
import {toast} from "react-hot-toast";
import { useNavigate } from "react-router-dom";
import '../css/ProfileEditForm.css';

export default function Login(){
    const navigation = useNavigate();
    const [userData, setData] = useState({
        email: '',
        password: '',
    })

    const login = async (e) => {
        e.preventDefault();
        const {email, password} = userData;
        try{
            const {data} = await axios.post('/login', {email, password});
            if(data.error){
                toast.error(data.error);
            } else {
                setData({});
                toast.success("Welcome, " + data.name + "!");
                navigation('/dashboard');
```

```
            window.location.reload();
          }
        }
        catch(error){
          console.log(error);
        }
    }

    //форма для логіну
    return (
      <form className="profile-edit-form" onSubmit={login}>
      <h2>Login</h2>
      <div className="form-group">
        <label>Email</label>
        <input
          type='email'
          placeholder="Enter Email"
          value={userData.email}
          onChange={(e) => setData({ ...userData, email: e.target.value })}
        />
      </div>
      <div className="form-group">
        <label>Password</label>
        <input
          type='password'
          placeholder="Enter password"
          value={userData.password}
          onChange={(e) => setData({ ...userData, password: e.target.value })}
        />
      </div>
      <button type='submit'>Login</button>
    </form>
    )
}

/* eslint-disable no-unused-vars */
import React, { useState, useEffect, useContext } from 'react';
import axios from 'axios';
import CreateOffer from '../components/CreateOffer';
import '../css/Marketplace.css';
import '../css/Shop.css';
import { UserContext } from "../../context/userContext";
import PlayingCard from '../components/PlayingCard';
import { toast } from 'react-hot-toast';

const Marketplace = () => {
  const { user, loading, setUser } = useContext(UserContext);
  const [cards, setCards] = useState([]);
  const [offers, setOffers] = useState([]);
  const [error, setError] = useState(null);

  //завантажити колекцію карток користувача
  useEffect(() => {
    if (user && user._id) {
      const userCards = user.cardCollection ? user.cardCollection.map(card => card.pokemonId) : [];
      setCards(userCards);
    }
  }, [user]);

  //завантажити всі пропозиції з бд
  useEffect(() => {
    const fetchOffers = async () => {
      try {
```

```
      const response = await axios.get('/offers');
      setOffers(response.data);
    } catch (error) {
      console.error('Помилка при завантаженні пропозицій ', error);
      setError('Не вийшло завантажити пропозиції');
    }
  };

  if (!loading && user) {
    fetchOffers();
  }
}, [loading, user]);

//оновлення списку пропозицій
const refreshOffers = async () => {
  try {
    const response = await axios.get('/offers');
    setOffers(response.data);
  } catch (error) {
    console.error('Помилка при оновленні пропозицій ', error);
  }
};

//обробник покупки чи обміну карток
const handleOfferAction = async (offer, isExchange) => {
  if (user._id === offer.userId._id) {
    toast.error("Ви не можете придбати власну картку");
    return;
  }

  try {
    const endpoint = isExchange
      ? `/offers/${offer._id}/exchange/${user._id}`
      : `/offers/${offer._id}/purchase/${user._id}`;

    const response = await axios.post(endpoint, {
      cardId: offer.pokemonId,
      exchangeCardId: isExchange ? offer.desiredCardId : undefined
    });

    if (response.data && response.data.money !== undefined) {
      setUser(prevUser => ({ ...prevUser, money: response.data.buyerMoney }));
    }

    toast.success(response.data.message || 'Обмін пройшов успішно!');
    refreshOffers();
  } catch (error) {
    toast.error(error.response?.data?.error || 'Помилка при обміні');
    console.error("Помилка транзакції ", error);
  }
};

//обробка видалення
const handleDeleteOffer = async (offerId) => {
  try {
    const response = await axios.delete(`/offers/${offerId}`, {
      data: { userId: user._id }
    });
    toast.success('Пропозиція видалена успішно');
    refreshOffers();  //оновити список пропозицій після видалення
  } catch (error) {
    toast.error(error.response?.data?.error || 'Помилка при видаленні пропозиції');
    console.error("Помилка видалення ", error);
```

```
  }
};


if (loading) {
  return <div className="loading">Loading Marketplace...</div>;
}

if (!user) {
  return <div className="error">User profile not found.</div>;
}

return (
  <div className="marketplace">
    <h1>Marketplace</h1>
    {error && <div className="error">{error}</div>}

    <CreateOffer cards={cards} refreshOffers={refreshOffers} />

    <h2>Current Offers</h2>
    <div className="offers-list cards-grid">
      {offers.length === 0 ? (
        <p>No offers available.</p>
      ) : (
        offers.map((offer) => {
          const isOwnedByUser = offer.userId._id === user._id;
          const userHasDesiredCard = cards.includes(offer.desiredCardId);
          const canExchange = offer.desiredCardId && offer.desiredCardName && userHasDesiredCard;

          return (
            <div key={offer._id} className="offer">
              <PlayingCard pokemonId={offer.pokemonId} />
              <p>Price: ${offer.price}</p>

              <button
                onClick={() => handleOfferAction(offer, false)}
                disabled={isOwnedByUser}
              >
                {isOwnedByUser ? 'Owned' : 'Buy for Money'}
              </button>

              {offer.desiredCardId !== 0 && offer.desiredCardName && (
                <>
                  <p>Exchange for: {offer.desiredCardName}</p>
                  <button
                    onClick={() => handleOfferAction(offer, true)}
                    disabled={isOwnedByUser || !canExchange}
                  >
                    {isOwnedByUser ? 'Owned' : (canExchange ? 'Exchange' : 'You dont have required card')}
                  </button>
                </>
              )}
              {isOwnedByUser && (
                <button
                  onClick={() => handleDeleteOffer(offer._id)}
                  className="delete-offer-btn"
                >
                  Delete Offer
                </button>
              )}
            </div>
          );
        })
```

```
        )}
      </div>
    </div>
  );
};

export default Marketplace;

import { useContext } from "react";
import { UserContext } from "../../context/userContext";
import { useNavigate } from "react-router-dom";
import '../css/Profile.css';
import '../css/index.css';
import UserInfo from "../components/UserInfo";

const Profile = () => {
    const { user, loading } = useContext(UserContext);
    const navigate = useNavigate();

    if (loading) {
        return <div className="loading">Loading...</div>;
    }

    if (!user) {
        return <div className="error">User profile not found.</div>;
    }

    const handleNavigate = (path) => {
        navigate(path);
    };

    return (
        <div className="profile-page">
            <UserInfo user={user}></UserInfo>
            <button onClick={() => handleNavigate("/profile-edit")}>Edit Profile</button>
            <button onClick={() => handleNavigate("/profile-cards")}>Card Collection</button>
        </div>
    );
};

export default Profile;

/* eslint-disable no-unused-vars */
import axios from "axios";
import {useState} from "react";
import {toast} from "react-hot-toast";
import { useNavigate } from "react-router-dom";
import '../css/ProfileEditForm.css';

export default function Register(){
    //навігатор - для перенаправлення на іншу сторінку
    const navigation = useNavigate();
    //дані користувача
    const [userData, setData] = useState({
        name: '',
        email: '',
        password: '',
    })

    //функція реєстрації викликається при підтвердження форми нижче
    const register = async (e) => {
        e.preventDefault();
        const {name, email, password} = userData;
```

```
        try{
            //запит до сервера post надсилає введені користувачем дані
            const {data} = await axios.post('/register', {name, email, password});
            //якщо відповідь від сервера містить помилку
            if(data.error){
                //виводимо спливаюче повідомлення про помилку
                toast.error(data.error);
            } else {
                //якщо користувач зареєстрований видаляємо його дані з форми
                setData({});
                //повідомлення про успішну реєстрацію
                toast.success('Реєстрація успішна!');
                //перенаправлення на іншу сторінку сайту
                navigation('/login');
            }
        }
        catch(error){
            console.log(error);
        }

    }
    //форма реєстрації
    //при підтвердженні викликається функція реєстрації
    //всі поля привязуються до відповідних змінних
    //при зміні значень полей вони записуються в змінні
    return (
        <form onSubmit={register} className="profile-edit-form">
            <h2>Register</h2>

            <div className="form-group">
                <label>Name</label>
                <input
                    type='text'
                    placeholder="Enter name"
                    value={userData.name}
                    onChange={(e) => setData({ ...userData, name: e.target.value })}
                />
            </div>
            <div className="form-group">
                <label>Email</label>
                <input
                    type='email'
                    placeholder="Enter Email"
                    value={userData.email}
                    onChange={(e) => setData({ ...userData, email: e.target.value })}
                />
            </div>
            <div className="form-group">
                <label>Password</label>
                <input
                    type='password'
                    placeholder="Enter password"
                    value={userData.password}
                    onChange={(e) => setData({ ...userData, password: e.target.value })}
                />
            </div>
            <button type="submit">Register</button>
        </form>
    )
}

//import React from "react";
```

```
export default function Rules() {
  return (
    <div className="rules-container">
      <h1>Game Rules</h1>
      <div className="rule">
        <h2>1. Objective of the Game</h2>
        <p>
          The goal of the game is to defeat your opponent by reducing his health points (HP) to zero.
          Players take turns to draw and play cards, with each card representing a different Pokémon.
        </p>
      </div>
      <div className="rule">
        <h2>2. Starting the Game</h2>
        <p>
          Each player starts with a hand of 5 cards, each containing a Pokémon.
          Players will draw cards from their deck to play and attack during their turn.
        </p>
      </div>
      <div className="rule">
        <h2>3. Turn Order</h2>
        <p>
          Players take turns drawing and playing cards. During a players turn,
          they may choose either one of their Pokémon card and place it on game field,
          draw new card from the deck or end turn.
        </p>
      </div>
      <div className="rule">
        <h2>4. Attacks</h2>
        <p>
          Each Pokémon has a set of stats including Attack (ATK), Defense (DEF), and Health Points (HP).
          At the end of each turn players Pokémons placed on field attack. If Pokémon has an ooposing Pokémon on
          opponents side - Pokémon will attack it. If there is no opposing Pokémon, then opponent will be attacked.
          Damage is calculated like this: DEF points substacted from ATK, then resulting damage is substracted from HP.
        </p>
      </div>
      <div className="rule">
        <h2>5. Winning the Game</h2>
        <p>
          A player wins when their opponents  HP is reduced to zero. If the opponent disconnects or leaves the game,
          the player automatically wins.
        </p>
      </div>
      <div className="rule">
        <h2>6. Rewards</h2>
        <p>
          After a match, winners receive in-game money and rating points.
          The reward amount is based on the match result:
          <br></br>- Winning: +200 money and +10 rating
          <br></br>- Losing: +50 money
          <br></br>- Opponent disconnected: +100 money
        </p>
      </div>
      <div className="rule">
        <h2>7. Multiplayer Mode</h2>
        <p>
          The game is designed for two players. The game will only start once both players are connected.
          During the game, players will face off in real-time, making strategic choices to outsmart their opponent.
```

```
                    </p>
                  </div>
                  <div className="rule">
                    <h2>8. Singleplayer Mode</h2>
                    <p>
                      Player can choose to play against computer opponent, then algorithm-based opponent will play again
s player.
                    </p>
                  </div>
                </div>
              );
            }

            /* eslint-disable no-unused-vars */
            import { useContext, useEffect, useState } from 'react';
            import axios from 'axios';
            import { UserContext } from '../../context/userContext';
            import '../css/Shop.css';
            import PlayingCard from '../components/PlayingCard';
            import toast from 'react-hot-toast';

            const Shop = () => {
              const { user, setUser } = useContext(UserContext);
              const [shopData, setShopData] = useState(null);
              const [loading, setLoading] = useState(true);
              const [countdown, setCountdown] = useState('');
              const [money, setMoney] = useState(user?.money || 0); //баланс користувача

              useEffect(() => {
                const fetchShopData = async () => {
                  //отримати дані магазину
                  if (user) {
                    try {
                      const response = await axios.get(`/shop/${user._id}`);
                      setShopData(response.data);
                      calculateCountdown(response.data.lastUpdated);
                    } catch (error) {
                      console.error('Помилка завантаження магазину ', error);
                    } finally {
                      setLoading(false);
                      setMoney(user.money);
                    }
                  }
                };
                fetchShopData();
              }, [user]);

              //розрахувати час до наступного оновлення
              const calculateCountdown = (lastUpdated) => {
                const now = new Date();
                const nextUpdate = new Date(lastUpdated);
                nextUpdate.setDate(nextUpdate.getDate() + 1);

                const timeRemaining = nextUpdate - now;
                if (timeRemaining > 0) {
                  const hours = Math.floor(timeRemaining / (1000 * 60 * 60));
                  const minutes = Math.floor((timeRemaining % (1000 * 60 * 60)) / (1000 * 60));
                  const seconds = Math.floor((timeRemaining % (1000 * 60)) / 1000);
                  setCountdown(`${hours}h ${minutes}m ${seconds}s`);
                } else {
                  setCountdown('Shop is updating soon...');
                }
              };
```

```
//оновлення таймеру щосекунди
useEffect(() => {
  const timer = setInterval(() => {
    if (shopData?.lastUpdated) {
      calculateCountdown(shopData.lastUpdated);
    }
  }, 1000);
  return () => clearInterval(timer);
}, [shopData]);

//покупка картки
const handlePurchase = async (pokemonId) => {
  try {
    const response = await axios.post(`/shop/${user._id}/purchase`, { pokemonId });

    if (response.data && response.data.money !== undefined) {
      setMoney(response.data.money);
    }

    //оновити дані магазин
    setShopData((prevData) => ({
      ...prevData,
      purchasedCards: [...prevData.purchasedCards, response.data.purchasedCard],
    }));

    toast.success(response.data.message || 'Картка придбана!');
  } catch (error) {
    toast.error(error.response?.data?.error || 'Помилка покупки картки');
    console.error("Помилка покупки ", error);
  }
};

if (loading) return <div>Loading...</div>;
if (!shopData) return <div>No shop data available.</div>;

return (
  <div className="shop-page">
    <h1>Daily Pokémon Cards</h1>
    <div className="balance">
      <p>Balance: ${money}</p>
    </div>
    <div className="shop-cards">
      {shopData.dailyCards.map((card) => {
        const isPurchased = shopData.purchasedCards.some((p) => p.pokemonId === card.pokemonId);
        return (
          <div key={card.pokemonId} className="cards-grid">
            <div>
              <PlayingCard pokemonId={card.pokemonId} />
              <p>Price: ${card.price}</p>
              <button
                onClick={() => handlePurchase(card.pokemonId)}
                disabled={isPurchased}
              >
                {isPurchased ? 'Purchased' : 'Buy'}
              </button>
              {isPurchased && <p className="purchased-notice">You have purchased this card!</p>}
            </div>
          </div>
        );
      })}
    </div>
    <div className="countdown-timer">
      <p>Next shop update in: {countdown}</p>
```

```
      </div>
    </div>
  );
};

export default Shop;

import { useState, useEffect } from 'react';
import axios from 'axios';
import { useParams } from 'react-router-dom';
import { toast } from 'react-hot-toast';
import '../css/index.css';
import UserInfo from '../components/UserInfo';

function User() {
    const { userId } = useParams();
    const [userInfo, setUserInfo] = useState(null);

    useEffect(() => {
        const fetchUserInfo = async () => {
            try {
                const { data } = await axios.get(`/user/${userId}`);
                if (data.error) {
                    toast.error(data.error);
                } else {
                    setUserInfo(data);
                }
            } catch (error) {
                toast.error('Помилка отримання даних користувача');
                console.error('Помилка завантаження даних ', error);
            }
        };

        fetchUserInfo();
    }, [userId]);

    if (!userInfo) {
        return <p>Loading...</p>;
    }

    return (
        <UserInfo user={userInfo}></UserInfo>
    );
}

export default User;

import {Routes, Route} from 'react-router-dom'
import Rules from './pages/Rules'
import Register from './pages/Register'
import Login from './pages/Login'
import './css/App.css'
import axios from 'axios'
import {Toaster} from 'react-hot-toast'
import { UserContextProvider } from '../context/userContext'
import Dashboard from './pages/Dashboard'
import Navbar from '../src/components/Navbar'
import Library from './pages/Library'
import Profile from './pages/Profile'
import Shop from './pages/Shop'
import Game from './pages/Game'
import EditProfile from './pages/EditProfile.'
import CardCollection from './pages/CardCollection'
```

```
import Marketplace from './pages/Marketplace'
import User from './pages/User'
//для запитів до сервера
axios.defaults.baseURL = 'http://localhost:3000';
axios.defaults.withCredentials = true;

function App() {
  //використовується контекст користувача для використання даних поточного користувача (див. ./context/userContext)
  //тостер - для спливаючих сповіщень на сайті
  //роути - показують, за яким шляхом знаходяться сторінки (./pages)
  return (
    <UserContextProvider>
    <Toaster position='bottom-right' toastOptions={{duration: 3000}}/>
      <Navbar>   </Navbar>
      <Routes>
        <Route path='/rules' element={<Rules />}/>
        <Route path='/register' element={<Register />}/>
        <Route path='/login' element={<Login />}/>
        <Route path='/dashboard' element={<Dashboard/>}/>
        <Route path='/game' element={<Game/>}/>
        <Route path='/library' element={<Library/>}/>
        <Route path='/shop' element={<Shop/>}/>
        <Route path='/marketplace' element={<Marketplace/>}/>
        <Route path='/profile' element={<Profile/>}/>
        <Route path='/profile-edit' element={<EditProfile/>}/>
        <Route path='/profile-cards' element={<CardCollection/>}/>
        <Route path="/user/:userId" element={<User></User>} />
      </Routes>
    </UserContextProvider>
  )
}

export default App

import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './css/index.css'
import { BrowserRouter as Router } from 'react-router-dom'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <Router>
      <App />
    </Router>
  </React.StrictMode>,
)
```