МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп'ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

КЛІЄНТСЬКА ЧАСТИНА ВЕБЗАСТОСУНКУ ДЛЯ КАРТКОВОЇ ГРИ

Виконав: студент групи 2П-21
Спеціальності
121 Інженерія програмного забезпечення
Дмитро ТКАЧЕНКО
Керівник:
Антон МАЛИЙ

Черкаси 2025

АНОТАЦІЯ

Кваліфікаційна робота «Клієнтська частина вебзастосунку для карткової гри» присвячена розробці інтерфейсу користувача для інтерактивного вебзастосунку, що реалізує функціональність карткової гри. У роботі досліджено сучасні підходи до фронтенд-розробки, проведено аналіз ринку подібних рішень, обґрунтовано вибір технологій та інструментів для реалізації клієнтської частини. Основну увагу приділено створенню функціонального, адаптивного та зручного інтерфейсу із застосуванням новітніх фреймворків і бібліотек.

Метою цієї роботи є проєктування та реалізація клієнтської частини вебзастосунку для карткової гри з використанням сучасних технологій веброзробки.

У процесі роботи було реалізовано структуру інтерфейсу, логіку гри, інтеграцію з API, а також проведено тестування застосунку для перевірки його стабільності та відповідності вимогам. Робота демонструє практичне застосування сучасних підходів до побудови SPA-застосунків і є прикладом ефективного використання UI/UX принципів у вебіграх.

Ключові слова: ВЕБЗАСТОСУНОК, ФРОНТЕНД, КАРТКОВА ГРА, UI/UX, КЛІЄНТСЬКА ЧАСТИНА, ВЕБІНТЕРФЕЙС, API, ТЕСТУВАННЯ, REACT, TAILWIND, TYPESCRIPT, SOCKET.IO.

ABSTRACT

The qualification work “Client-side Web Application for a Card Game” focuses on the development of a user interface for an interactive web application that implements the functionality of a card game. The work explores modern approaches to frontend development, analyzes the market of similar solutions, and justifies the choice of technologies and tools used to implement the client side. Special attention is given to creating a functional, responsive, and user-friendly interface using the latest frameworks and libraries.

The aim of this work is to design and implement the client-side of a web application for a card game using modern web development technologies.

The project includes the implementation of the interface structure, game logic, API integration, and testing of the application to ensure its stability and compliance with the requirements. The work demonstrates the practical application of modern approaches to building SPA applications and serves as an example of effective use of UI/UX principles in web-based games.

Keywords: WEB APPLICATION, FRONTEND, CARD GAME, UI/UX, CLIENT-SIDE, WEB INTERFACE, API, TESTING, REACT, TAILWIND, TYPESCRIPT, SOCKET.IO.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

UI/UX – інтерфейс користувача (UI) і досвід користувача (UX), що визначають зовнішній вигляд та зручність використання застосунку.

API – інтерфейс програмування застосунків, що дозволяє різним програмам взаємодіяти між собою.

HTTP-запит – запит, який клієнт (наприклад, браузер) надсилає до сервера для отримання чи надсилання даних за протоколом HTTP.

CI/CD – безперервна інтеграція та безперервне доставлення, автоматизовані процеси для тестування, збірки та деплою коду.

SSH-канал – захищений протокол для віддаленого доступу до серверів та передавання даних через зашифроване з'єднання.

JWT-токен – компактний токен у форматі JSON, що використовується для безпечної передачі інформації про автентифікацію.

JSON – текстовий формат обміну даними, який легко читається людиною та обробляється машинами.

XSS-атака – тип атаки, за якої зловмисник впроваджує шкідливий скрипт у вебсторінку для викрадення даних користувача.

SQL-ін'єкція – вразливість, що дозволяє зловмиснику вставити шкідливий SQL-код у запит до бази даних.

ВСТУП

Нині ринок вебзастосунків є надзвичайно перспективним, але складним для конкуренції. Розробка унікального продукту в таких умовах потребує не лише креативного підходу, а й глибоких знань сучасних технологій. Проте розвиток технологій надає розробникам інструменти, що значно спрощують реалізацію складних завдань. Тому тестування нових технологій у процесі створення реального проєкту стає актуальним й надає не лише теоретичні відомості, а й перевірку на практиці ефективності обраних підходів. Саме тому тема розробки клієнтської частини вебзастосунку для карткової гри є своєчасною та важливою.

Об'єктом дослідження виступають існуючі вебсайти та застосунки, які демонструють різноманітні підходи до реалізації клієнтської частини. У процесі дослідження буде проаналізовано як проєкти, що мають схожість із темою карткових ігор, так і інші ресурси, які можуть надати корисну інформацію щодо реалізації сучасного UI/UX, логіки та архітектури фронтенд-застосунків.

Предметом дослідження є сучасні фреймворки, бібліотеки та технології, які використовуються для створення клієнтської частини вебзастосунків. Особлива увага приділятиметься їхньому практичному застосуванню, а також демонстрації переваг та недоліків у межах реального проєкту.

Метою даного проєкту є створення клієнтської частини інтерактивного вебзастосунку для карткової гри з використанням сучасних технологій. При цьому основна увага зосереджена не лише на розробці функціонального продукту.

Завдання, що вирішуватимуться у процесі роботи:

1. Проведення дослідження та порівняння актуальних технологій на ринку веброзробки.
2. Створення дизайну вебзастосунку та його реалізація відповідно до принципів сучасного UI/UX.
3. Розробка логічної структури клієнтської частини гри, імплементація

функціонального коду.

4. Проведення тестування готового застосунку з метою перевірки стабільності та ефективності реалізованих рішень.

Практичне значення розробки полягає в тому, що результатом роботи стане повноцінна клієнтська частина вебзастосунку, яка може бути використана як основа для комерційного або освітнього продукту. Крім того, застосування сучасних інструментів дозволить створити шаблон ефективної архітектури фронтенд-застосунку, який може бути адаптований для інших інтерактивних ігор або проєктів. Результати дослідження також можуть бути корисними для розробників, які прагнуть впроваджувати новітні технології у свої проєкти.

РОЗДІЛ 1
АНАЛІЗ РИНКУ ВЕБЗАСТОСУНКІВ ТА ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ДЛЯ ВЕБРОЗРОБКИ

1.1 Аналіз ринку вебзастосунків

Нині ринок вебзастосунків розвивається надзвичайно динамічно, щодня зростає кількість інноваційних сервісів, інструментів і інтерактивних платформ. Це відображає тенденцію глобалізації та цифровізації багатьох сфер діяльності, включаючи ігри, де вебзастосунки займають важливе місце. Вебігри, зокрема, стали популярним способом проведення часу для мільйонів користувачів по всьому світу, адже вони дозволяють грати без необхідності завантаження великих файлів чи встановлення додаткового програмного забезпечення.

Важливою частиною ринку стали багатокористувацькі ігри в браузері, які дають змогу користувачам взаємодіяти з іншими гравцями в реальному часі, створюючи соціальну та конкурентну атмосферу. Ці ігри охоплюють безліч жанрів - від рольових до стратегій і симуляторів, і є доступними з будь-якого пристрою, що має підключення до інтернету.

У цьому контексті карткові ігри надають наближення до реального світу, надаючи ті самі відчуття, при цьому залишаючись зрозумілими для більшості. Тому такі вебзастосунки за останні кілька років набули надзвичайної популярності. Найвідоміші сервіси пропонують гравцям можливість змагатися між собою з будь-яких пристроїв, включаючи комп'ютери, планшети та смартфони. Ці сервіси мають розвинену екосистему, що включає реєстрацію користувачів, рейтингові системи, внутрішні валюти для ставок і транзакцій, багатокористувацьку логіку та складну графіку. Однак, незважаючи на численні функції та привабливий контент, деякі платформи можуть бути складними для нових гравців через необхідність розуміння різних правил і механік, а також через наявність певних обмежень щодо часу гри чи доступність режимів.

У зв'язку з цим на ринку з'являється попит на легкі, швидкі та доступні

вебзастосунки з можливістю грати одному, які дозволяють користувачам одразу почати гру без складних процесів реєстрації, складної навігації та великих системних вимог або без фінансових вкладів. Особливо це актуально для користувачів, які шукають швидкі і прості розважальні методи, не маючи наміру проводити годинні сесії чи інвестувати в повноцінну онлайн-гру. У цьому контексті демо-версії таких ігор виступають прекрасною альтернативою, оскільки вони пропонують мінімалістичний, але захопливий досвід гри, орієнтуючись на простоту та доступність. Такі застосунки можуть бути використані як навчальні інструменти для нових гравців, для тестування стратегії або просто для веселих коротких сесій з друзями.

Загалом, аналіз ринку [13] свідчить про те, що вебігри мають стабільну аудиторію, що розширюється з кожним роком завдяки росту популярності онлайн-ігор та інтерактивних платформ. Гравці цінують реалістичність, динамічність та інтуїтивний інтерфейс, що дозволяє швидко зануритися в гру без складних налаштувань. Крім того, зростає попит на мінімалістичні та адаптивні рішення, що дозволяють зберігати високу якість гри, не перевантажуючи користувача надмірними функціями. Звідси, створення сучасної, спрощеної клієнтської частини для карткової гри є актуальним кроком у напрямку задоволення запиту на легкий ігровий досвід у вебсередовищі.

У майбутньому можна очікувати більшу популярність інтерактивних ігор, які будуть використовувати новітні технології, включаючи реальний час, штучний інтелект, адаптивні інтерфейси та інноваційні механізми геймплею, що значно покращить досвід користувача. Сучасні тренди вказують на необхідність інтеграції соціальних функцій, мобільних версій та кросплатформності для більшої залученості користувачів і створення більш інтуїтивно зрозумілих, швидких і зручних рішень.

1.2 Порівняння наявних рішень і виділення ключових особливостей

Існує безліч рішень для реалізації інтерактивних платформ і ігор, які

відрізняються за складністю, функціональними можливостями та вимогами до користувача. Однак усі вони мають свої сильні і слабкі сторони, що варто враховувати при виборі оптимальної платформи або технології. Ключові характеристики, які відрізняють різні рішення на ринку, можна розглянути в контексті простоти використання, доступності та соціальних аспектів.

Одним з важливих аспектів, який вирізняє деякі рішення, є простота у використанні. Для більшості користувачів критично важливою складовою є можливість швидко стартувати без необхідності тривалого навчання чи занурення в складний інтерфейс. Такі рішення часто пропонують інтуїтивно зрозумілий інтерфейс, що дозволяє початківцям без зусиль освоїтися в процесі гри.

Платформи, що орієнтуються на простоту, зазвичай пропонують зручний і швидкий доступ до основних функцій гри, з мінімумом додаткових опцій. Інші рішення можуть надавати більше можливостей для налаштувань, що дозволяє користувачам адаптувати платформу під свої особисті уподобання. Це можуть бути різноманітні варіанти налаштування ігрових параметрів, управління інтерфейсом, а також гнучкі налаштування для різних типів пристроїв і екранів. Такі платформи, як правило, підходять для досвідчених користувачів, які хочуть налаштувати гру під свої особисті уподобання або пристрої.

Зі зростанням використання мобільних пристроїв особливо важливим є забезпечення мобільної доступності і кросплатформенності. Рішення, які дозволяють гравцям продовжувати гру на різних пристроях - комп'ютерах, смартфонах, планшетах, - створюють значну перевагу, оскільки користувач може безперешкодно переключатися між платформами. Багато сучасних рішень орієнтовані саме на це, з адаптивними інтерфейсами, що дозволяють комфортно грати в будь-яких умовах.

Ключовою особливістю багатьох сучасних рішень є соціальні функції, що дозволяють користувачам взаємодіяти один з одним. Це можуть бути чати, рейтингові системи, додавання друзів, створення команд або клубів, участь у змаганнях. Такий підхід сприяє залученню користувачів і створює насичений та

глибокий досвід, оскільки ігри стають не лише способом розваги, але й засобом соціалізації та спільного проведення часу.

Не менш важливою характеристикою є рівень безпеки, що надається платформою. Для деяких рішень особливе значення має забезпечення конфіденційності особистих даних користувачів, захист від шахрайства і стабільність роботи системи, особливо під час пікових навантажень. Високий рівень захисту та стабільність роботи платформи є важливими факторами для залучення довіри користувачів, особливо в іграх з грошовими ставками чи обробкою особистих даних.

Залежно від характеру гри, певні рішення повинні мати здатність масштабуватися для підтримки великої кількості одночасних користувачів. Це може включати в себе механізми балансування навантаження на сервери, підтримку великої кількості одночасних сесій без втрати якості гри, а також можливість швидко адаптувати платформу до зростаючого попиту.

Загалом, порівняння наявних рішень показує, що кожна платформа або технологія має свої сильні сторони в окремих аспектах, таких як простота використання, мобільність, соціальна взаємодія, безпека та підтримка великої кількості користувачів. Вибір оптимального рішення залежить від вимог конкретного проєкту, цільової аудиторії та необхідних функцій.

1.3 Огляд і вибір сучасних технологій для розробки клієнтської частини

Розробка клієнтської частини вебзастосунків є важливою складовою частиною процесу створення будь-якого інтерактивного сервісу. Це включає в себе не лише дизайн і графічне відображення, а й взаємодію з користувачем через інтерфейс, що має бути зручним, ефективним та адаптивним до різних пристроїв. У цьому контексті вибір сучасних технологій для розробки клієнтської частини має вирішальне значення для успіху проєкту. Далі наведено кілька основних технологій, які активно будуть використовуються в проєкті.

На етапі проєктування важливу роль відіграє використання інструментів для прототипування й дизайну. Програмний засіб Figma є одним із найпопулярніших середовищ для створення макетів, інтерфейсів та інтерактивних прототипів. Він дозволяє візуалізувати майбутній вигляд застосунку, погодити дизайн між усіма учасниками команди та швидко вносити зміни відповідно до зворотного зв'язку. Завдяки хмарній природі сервісу, Figma забезпечує ефективну командну роботу в реальному часі, що особливо важливо в процесі розробки.

На ринку існує кілька альтернативних інструментів, таких як Adobe XD [3] та Sketch [4], проте Figma [1] має низку переваг, які роблять її оптимальним вибором для даного проєкту. Нижче наведено порівняльну таблицю 1.1 основних характеристик найпопулярніших інструментів для UI/UX-дизайну [2].

Порівняння показало, що Figma має низку суттєвих переваг перед конкурентами: вона є кросплатформним, доступною у браузері без потреби встановлення, підтримує повноцінну спільну роботу в реальному часі, а також має багату екосистему плагінів та API. Саме завдяки цим властивостям Figma була обрана як основний інструмент для проєктування інтерфейсу у даному проєкті, оскільки забезпечує гнучкість, швидкість і зручність для командної розробки.

Для ефективного управління версіями коду та спільної роботи над проєктом використовується Git та GitHub [5]. Система контролю версій Git забезпечує збереження історії змін, дозволяє працювати над різними гілками проєкту, об'єднувати зміни та вирішувати конфлікти. Вебсервіс GitHub, як хостинг для репозиторіїв, забезпечує зручне середовище для спільної роботи, огляду коду і документування.

Таблиця 1.1 – Порівняння інструментів для прототипування інтерфейсів 13

Характеристика	Figma	Adobe XD	Sketch
Платформа	Веб, Windows, macOS	Windows, macOS	Лише macOS
Робота в браузері	Так	Обмежена	Ні
Спільна робота онлайн	Так (в реальному часі)	Обмежена (через хмару)	Через сторонні сервіси
Вартість базової версії	Безкоштовна	Безкоштовна	Платна (30 днів безкоштовно)
Імпорт/експорт ресурсів	Зручно, багато форматів	Добре	Добре
Плагіни та розширення	Багато, активно оновлюються	Менше	Дуже багато, але тільки для macOS
Підтримка командної роботи	Вбудована, ефективна	Обмежена	Через Abstract, GitHub тощо
Платформа для розробників	API, інтеграції, REST API	Обмежено	Через сторонні рішення

Стилізація за допомогою CSS дозволяє створювати ефектні та адаптивні стилі для вебсторінок, що дає можливість будувати інтерфейси, які чудово виглядають на різних екранах - від мобільних пристроїв до великих моніторів. Завдяки медіазапитам, анімаціям, переходам і можливостям побудови гнучких макетів, CSS забезпечує високу гнучкість у дизайні, покращує взаємодію користувача з інтерфейсом і дозволяє значно зменшити потребу в сторонніх бібліотеках для оформлення.

CSS Grid та Flexbox [6] – це сучасні методи для створення адаптивних і складних макетів вебсторінок, які автоматично підлаштовуються під розміри екрану користувача. Вони дозволяють створювати складні макети без необхідності використання фреймворків і можуть значно полегшити процес верстки інтерфейсу.

Tailwind CSS [7] – це сучасний CSS-фреймворк, який базується на утилітарних класах, дозволяючи створювати адаптивні та естетично привабливі інтерфейси без необхідності писати власний CSS. Використовуючи прості класи,

зокрема, для кольорів, розмірів або відступів, розробники можуть швидко створювати інтерфейси, що автоматично адаптуються під різні екрани. Tailwind CSS також дозволяє налаштовувати фреймворк під конкретні потреби проєкту, зокрема змінювати кольорові палітри чи шрифти через конфігураційний файл. Така система класів забезпечує високу консистентність дизайну, що дозволяє підтримувати узгодженість в усьому проєкті, а також суттєво зменшує час, необхідний для розробки. Завдяки своїй гнучкості та швидкості, Tailwind CSS став популярним інструментом серед розробників для створення адаптивних, швидких та сучасних інтерфейсів.

Мова JavaScript є основною мовою програмування для створення інтерактивних елементів на вебсторінках. Завдяки численним бібліотекам і фреймворкам, таким як React, JavaScript дозволяє реалізовувати складні ігрові механіки, динамічні зміни інтерфейсу, а також керувати станом застосунку в реальному часі. Крім того, JavaScript активно підтримується спільнотою розробників, має велику кількість готових рішень і модулів, що значно пришвидшує розробку і дозволяє легко масштабувати застосунок відповідно до потреб проєкту.

Мова JavaScript має широку екосистему бібліотек, які значно розширюють його функціональність та спрощують розробку клієнтської частини. Однією з ключових задач є обмін даними з сервером, для чого застосовуються спеціалізовані бібліотеки. Бібліотека Axios [8] - популярна JavaScript-бібліотека, яка використовується для здійснення HTTP-запитів до серверу. Вона забезпечує простий і зрозумілий синтаксис для виконання запитів типу GET, POST, PUT, DELETE, а також дозволяє зручно працювати з асинхронними операціями завдяки підтримці промісів. Використання Axios має низку переваг: автоматичне перетворення відповідей у формат JSON, обробка помилок, підтримка таймаутів, налаштування заголовків і авторизації. У контексті розробки клієнтської частини для гри, ця бібліотека є корисною для отримання або надсилання даних про стан гри, дії гравців тощо, сприяючи ефективній взаємодії між клієнтом і сервером.

Фреймворк ReactJS – один із найпопулярніших JavaScript-фреймворків,

який дозволяє створювати ефективні односторінкові вебзастосунки. Завдяки використанню компонентної архітектури та віртуального DOM, React забезпечує високу швидкість і зручність у розробці. Цей фреймворк відмінно підходить для створення динамічних інтерфейсів, де потрібно часто оновлювати контент або взаємодіяти з користувачем в реальному часі, що особливо важливо для ігор, таких як покер.

Мова TypeScript [9] – надбудова над JavaScript, яка додає статичну типізацію. Вона дозволяє зменшити кількість помилок при розробці, зробити код більш надійним і підтримуваним. Мова TypeScript є чудовим вибором для великих проєктів, де важливі зрозумілість коду та можливість розширення в майбутньому. Це дозволяє значно полегшити процес розробки та тестування.

Протокол комунікації WebSockets [10] дозволяє організувати двостороннє з'єднання між клієнтом і сервером в реальному часі, що є необхідним для багатьох інтерактивних вебзастосунків. Вебсокети забезпечують збереження постійного з'єднання, що дає змогу миттєво передавати дані між клієнтом і сервером, не потребуючи перезавантаження сторінки. Це особливо важливо для ігор, де стан гри повинен оновлюватися без затримок.

Бібліотека Socket.IO [11] використовує WebSockets і забезпечує ще більше можливостей для роботи з реальним часом, підтримуючи при цьому всі браузери та мережеві підключення.

Інструмент Vitest [12] – це сучасний тестовий раннер для JavaScript/TypeScript, який спроєктований для максимально швидкого та зручного запуску тестів у проєктах, побудованих на базі Vite. Він чудово інтегрується з React, підтримує всі необхідні можливості для модульного тестування, зокрема його переваги наведені в табл. 1.2.

Таблиця 1.2 – Порівняння Vitest з іншими популярними інструментами

Особливість	Vitest	Jest	Mocha + Chai	Jasmine
Швидкість	Дуже швидкий (завдяки Vite)	Повільніший через Babel	Помірна	Помірна
Підтримка TypeScript	Вбудована, без додаткових налаштувань	Вимагає додаткових трансформерів	Потрібна конфігурація	Потрібна конфігурація
Мокінг	Вбудований	Вбудований	Потрібні додаткові бібліотеки	Вбудований
Інтеграція з Vite	Ідеальна	Потрібні плагіни/налаштування	Відсутня	Відсутня
Паралельне виконання тестів	Так	Так	Обмежена	Обмежена
Зручність налаштування	Мінімальна	Помірна	Вимагає багато налаштувань	Помірна
API	Схоже на Jest	Широко використовуваний	Гнучкий, але складніший	Простий, але старомодний

З огляду на порівняння, Vitest вирізняється серед інших інструментів модульного тестування завдяки високій швидкості виконання, вбудованій підтримці TypeScript, зручному мокінгу та інтеграції з Vite. Його просте налаштування, сучасний API та активна підтримка спільноти роблять Vitest особливо привабливим вибором для проєктів, побудованих з використанням сучасного стеку, зокрема Vite + React.

РОЗДІЛ 2
ПРОЄКТУВАННЯ І РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБЗАСТОСУНКУ

2.1 Функціональні та нефункціональні вимоги до застосунку

Одним із ключових завдань на етапі планування є визначення вимог до застосунку. Це охоплює опис основних можливостей системи, а також технічних і експлуатаційних характеристик, що гарантують її стабільну та ефективну роботу. Основні функціональні вимоги:

1. Авторизація та аутентифікація користувачів. Для безпечного доступу реалізовано перевірку логіна і пароля з хешованим зберіганням паролів. Після входу користувач отримує JWT-токен для подальшої ідентифікації. Авторизація контролює доступ до функцій залежно від ролі.

2. Надсилання сповіщень у реальному часі. Застосунок підтримує миттєві сповіщення про важливі події, які відображаються без оновлення сторінки, що підвищує зручність та оперативність взаємодії.

3. Адаптивний інтерфейс. Інтерфейс автоматично підлаштовується під різні розміри екранів за допомогою CSS-медіазапитів і гнучких сіток (Flexbox, Grid), забезпечуючи комфортне користування на всіх пристроях.

4. Інтерфейс гри та логіка відображення карт. Відповідає за зручне та зрозуміле відображення карткових елементів, анімації роздачі карт, ходу гравця, підсумкових результатів тощо. Забезпечує інтуїтивно зрозумілий і привабливий візуальний досвід.

5. Обробка користувацьких дій у грі. Реалізація реагування на дії гравця: вибір карти, ставка, пас, хід тощо. Взаємодія з сервером для відправки ходів і отримання оновлень стану гри.

6. Реалізація мультиплеєра (онлайн-з'єднання). Забезпечення синхронізації ігрового процесу між кількома гравцями в реальному часі, включно з обробкою стану гри, черговості ходів і повідомленнями про дії інших

учасників. 18

7. Інформаційна панель ігрового процесу. Відображення поточного стану гри, кількості очок, імен гравців, історії ходів і іншої допоміжної інформації.

8. Можливість чату між гравцями. Вбудований чат для комунікації гравців під час гри, що підвищує соціальну взаємодію.

9. Налаштування профілю користувача. Можливість зміни аватара, нікнейма, а також управління особистими даними.

10. Історія ігор і статистика. Збереження та відображення статистики гравця: кількість зіграних ігор, перемог, поразок, найкращі результати.

Для наочності взаємозв'язку користувача із системою та основних функцій застосунку наведено на рис.2.1

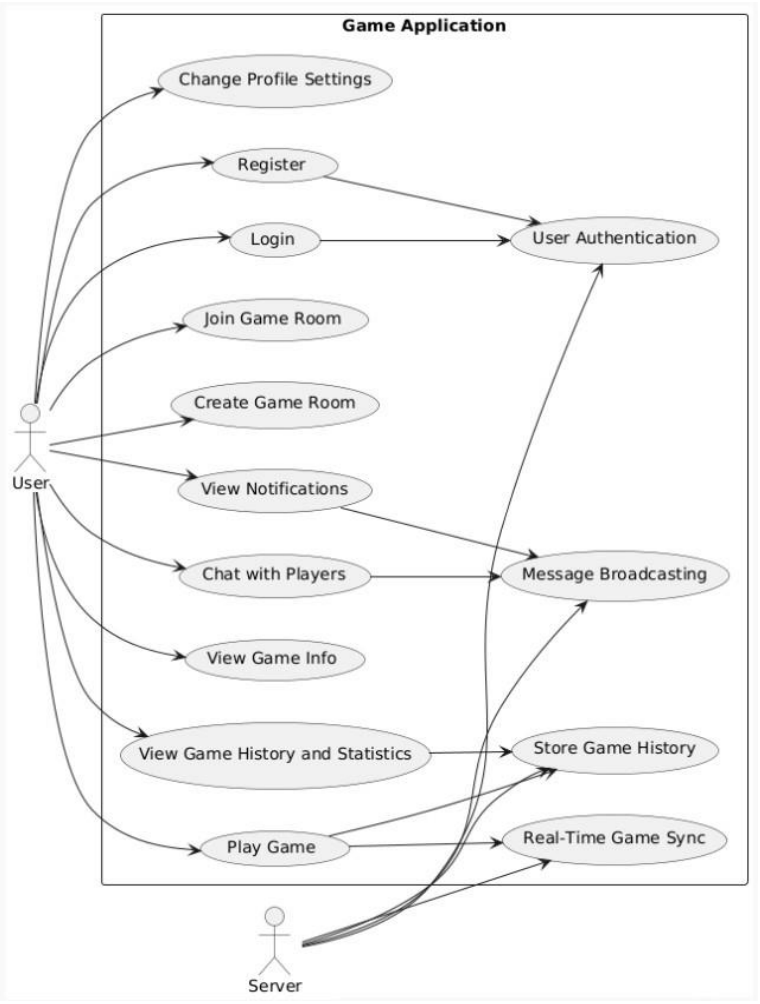

Рисунок 2.1 – Діаграма прецедентів для ключових сценаріїв взаємодії

Нефункціональні вимоги:

1. Продуктивність - швидкий відгук інтерфейсу (не більше 200 мс на основні дії).
2. Безпека - захист особистих даних користувачів, шифрування переданих даних (HTTPS, зберігання хешованих паролів).
3. Масштабованість - можливість розширення функціональності без потреби повного переписування архітектури.
4. Зручність користування - інтуїтивно зрозумілий інтерфейс, відповідність принципам UI/UX.
5. Сумісність - підтримка актуальних версій браузерів (Google Chrome, Mozilla Firefox, Microsoft Edge).
6. Надійність - відсутність критичних помилок при стандартних сценаріях використання.

Ці вимоги формують основу для реалізації застосунку та забезпечують відповідність як очікуванням користувачів, так і стандартам розробки програмного забезпечення.

2.2 Проєктування структури сайту

Проєктування структури сайту є одним з ключових етапів розробки вебзастосунку, що визначає логіку переходів між сторінками, зручність навігації для користувачів, а також дозволяє забезпечити ефективну організацію коду та UI-компонентів. У межах цього проєкту структура сайту сформована з урахуванням основних функціональних вимог, а також типових сценаріїв взаємодії користувача з платформою. Схематично все зображено на рис. 2.1.

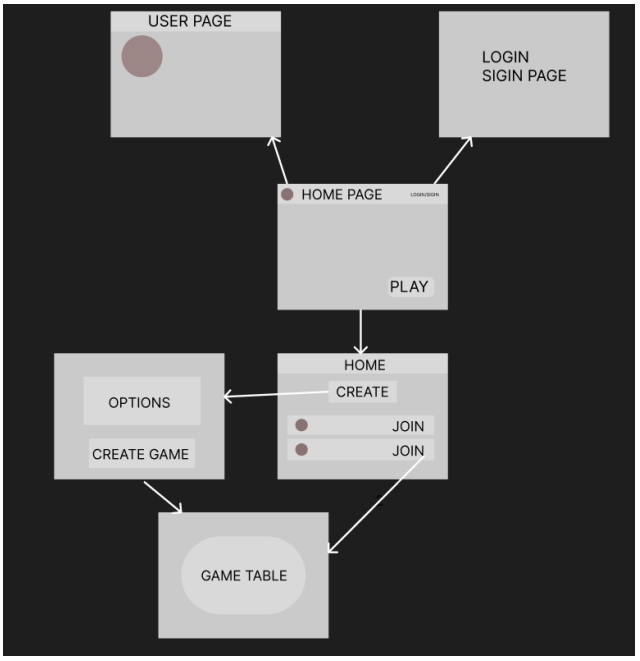

Рисунок 2.2 – Схема сторінок вебзастосунку

Сайт складається з таких основних сторінок:

1. Головна сторінка/ Сторінка навігації. Головна сторінка є стартовою точкою взаємодії користувача з платформою. Вона містить: заклики до дії (Call to Action): "Увійти", "Зареєструватися"; навігаційне меню; елементи брендування (логотип). Головна сторінка має інформативну функцію, привертає увагу нових користувачів і дозволяє швидко перейти до ключових функцій.

Вигляд головної сторінки наведено на рис. 2.3.

Рисунок 2.3 – Сторінка навігації

2. Сторінка користувача доступна лише після входу в систему. Вона відображає: профіль користувача (ім'я, аватар, статистика ігор); список активних та завершених ігор; можливість редагування профілю; вихід з акаунту.

Ця сторінка є центром особистої інформації та швидкого доступу до дій користувача. Також ця сторінка зображена на рис. 2.4.

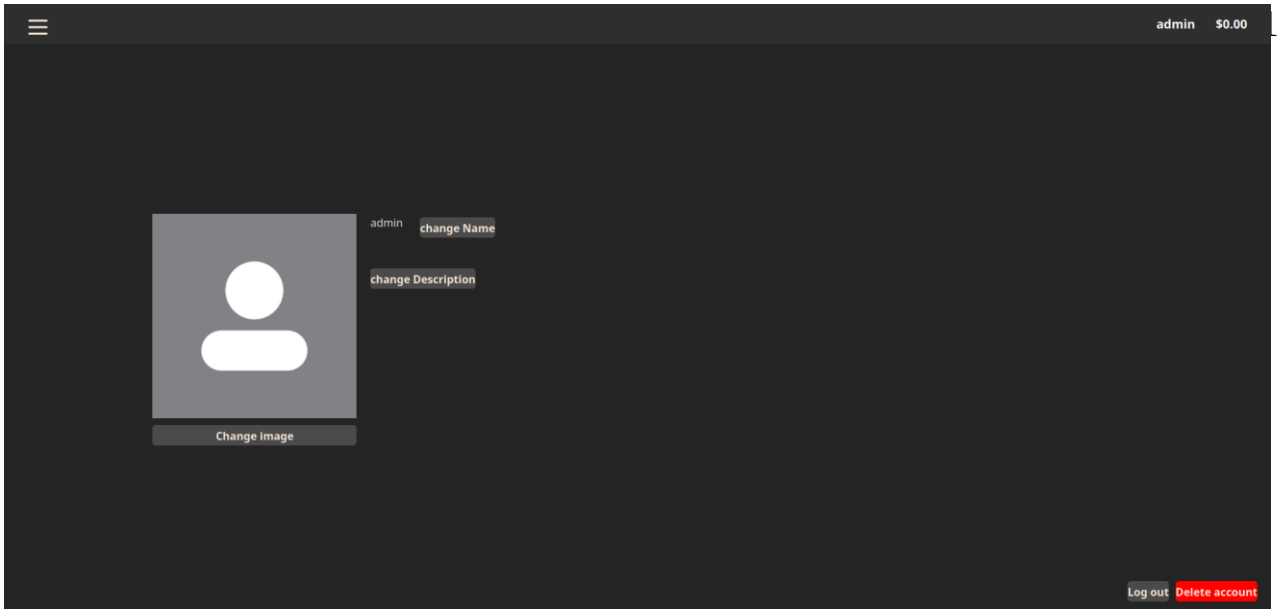

Рисунок 2.4 – Сторінка користувача

3. Сторінка вибору гри надає користувачу можливість: переглядати список доступних ігор (публічних або за запрошенням); приєднуватись до вже створених ігор; переглядати статус гри (очікує на гравців / активна / завершена). Передбачено фільтрацію за жанром, кількістю гравців, типом гри тощо.

4. Сторінка створення гри дозволяє користувачу створити нову гру, задавши необхідні параметри: назва гри; пароль (за потреби).

Після створення гри користувач автоматично перенаправляється до очікуваного лобі гри. Форма створення гри зображена на рис. 2.5.

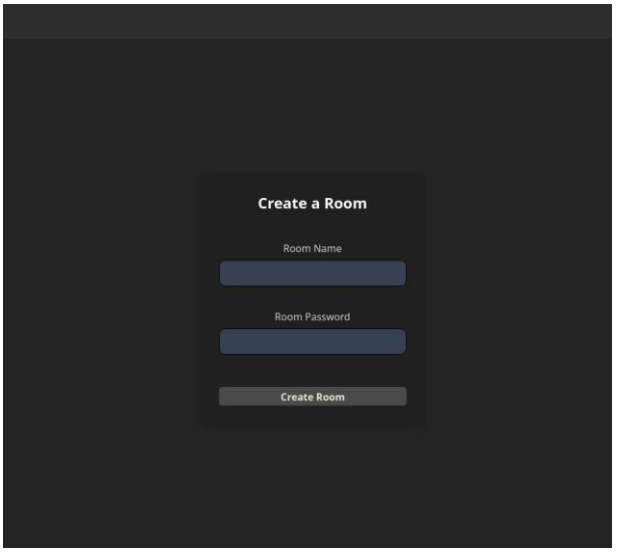

Рисунок 2.5 – Форма створення гри

5. Сторінка реєстрації у вигляді форми для нового користувача передбачає введення: логіна; електронної пошти; пароля; підтвердження пароля. Передбачено валідацію введених даних та повідомлення про помилки (наприклад, зайнятий логін або неправильна електронна пошта). Форма реєстрації зображена на рис. 2.6.

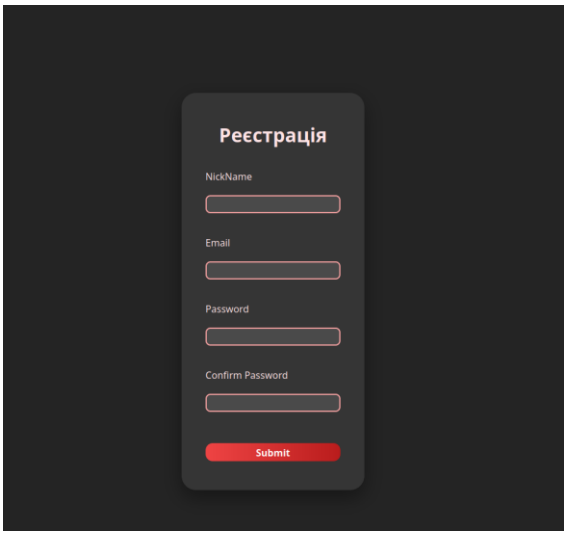

Рисунок 2.6 – Форма реєстрації

6. **Сторінка автентифікації містить** форму входу до системи містить: поля логіна/електронної пошти та пароля; кнопку "Увійти"; посилання для відновлення пароля або реєстрації. У випадку успішної автентифікації користувач перенаправляється на сторінку користувача. Форма автентифікації зображена на рис. 2.7.

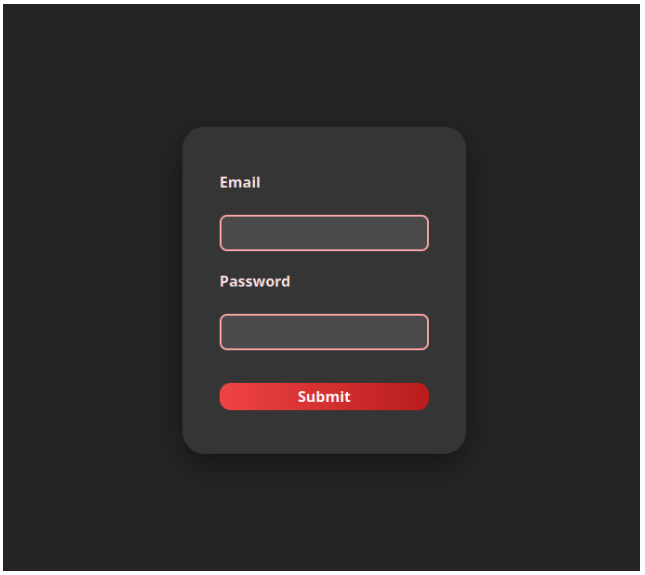

Рисунок 2.7 – Форма автентифікації

7. Сторінка гри - це основна інтерактивна частина сайту, де відбувається сама гра. А саме: відображається поле гри, карти/елементи/столи тощо; таймер ходів; динамічне оновлення стану гри в реальному часі; логіка переможця, очки тощо. Сторінка гри побудована з урахуванням сучасного UI/UX, оптимізована для всіх типів пристроїв. Дизайн сторінки наведений в рис. 2.8.

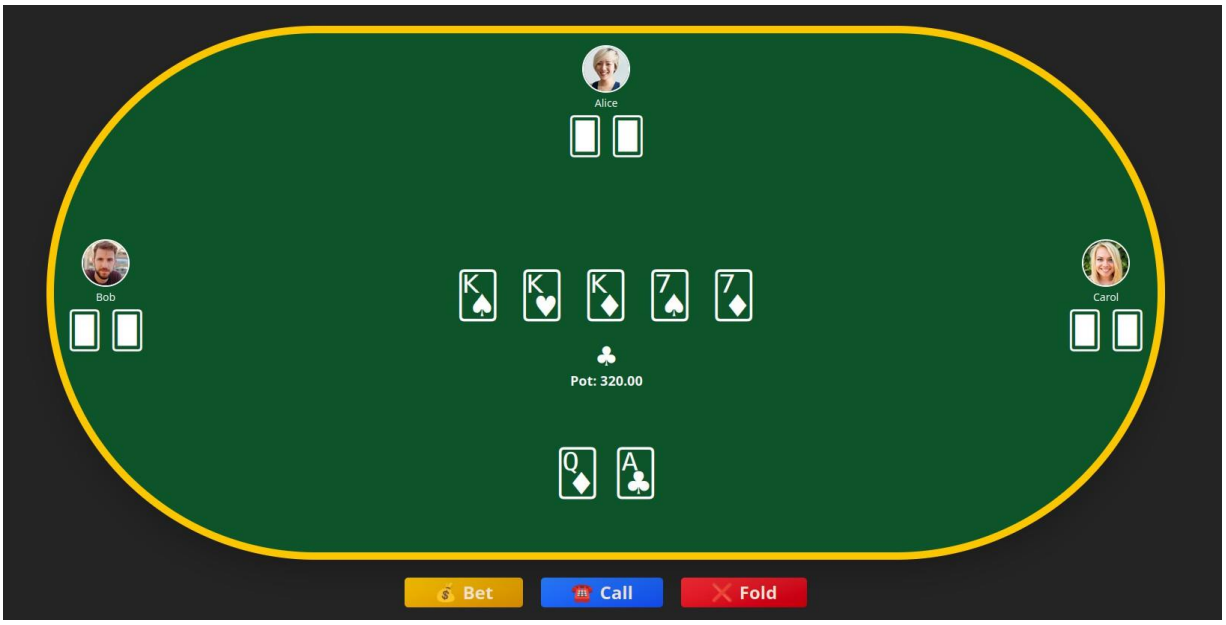

Рисунок 2.8 – Дизайн сторінки гри

Для ліпшого розуміння бізнес-логіки переміщення наведено діаграму послідовності на рис. 2.9.

2.3 Налаштування середовища розробки

Для розробки клієнтської частини вебзастосунку з реалізацією карткової гри було налаштовано відповідне середовище розробки, яке включає в себе інструменти, бібліотеки та фреймворки, необхідні для ефективної роботи над проєктом.

1. Вибір редактора коду. Для написання коду було використано середовище розробки Visual Studio Code, яке забезпечує зручну навігацію по проєкту, автодоповнення коду, налагодження та інтеграцію з системою

2. Створення проєкту за допомогою Vite. Для ініціалізації фронтенд-проєкту було використано Vite - сучасний інструмент збірки, що забезпечує надшвидке збирання та гаряче оновлення модулів (HMR). Процес створення та встановлення залежностей продемонстровано в лістингу 2.1.

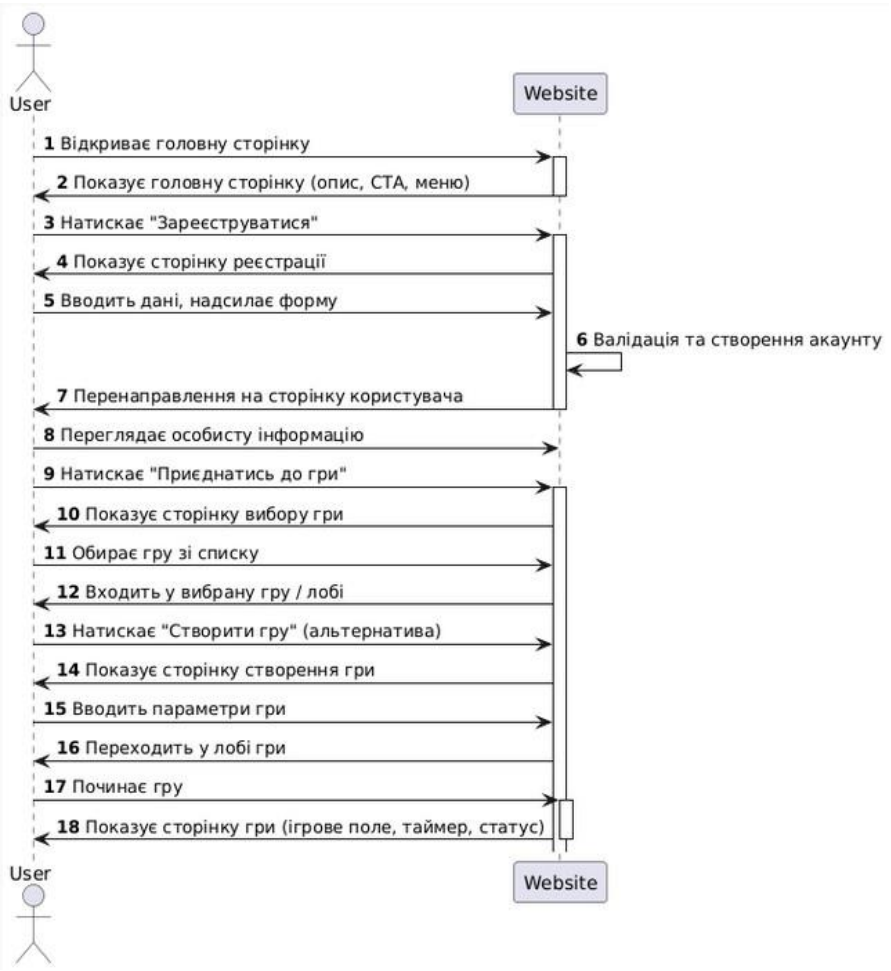

Рисунок 2.9 – Діаграма послідовності переміщення по сайту

Лістинг 2.1 – Створення проєкту та встановлення залежностей

```
npm create vite@latest project-name -- --template react cd project-name
npm install
```

3. Встановлення основних бібліотек. Було додатково встановлено бібліотеку react-router-dom для маршрутизації сторінок; бібліотеку axios для

виконання HTTP-запитів до серверної частини; бібліотеку socket.io-client для підтримки реального часу в грі; фреймворк Tailwind CSS для зручного та швидкого стилізування інтерфейсу.

4. Структуру проєкту організовано в вигляді такої ієрархії директорій:

```
├── eslint.config.js      # Конфігурація ESLint для перевірки якості коду
├── index.html  # Головний HTML-файл, у який вбудовується React-застосунок
├── package.json        #       Містить список залежностей, скриптів та загальну конфігурацію проєкту
├── package-lock.json # Фіксує точні версії встановлених npm-пакетів
├── public/
│   └── vite.svg        # Публічні файли, доступні напряму (без обробки Vite)
├── README.md       # Документація проєкту
├── src/    # Основна директорія з вихідним кодом застосунку
│   ├── App.css         # Загальні стилі застосунку
│   ├── App.tsx         # Кореневий компонент застосунку
│   ├── index.css       # Базові глобальні стилі
│   ├── main.tsx        # Точка входу у застосунок (рендерінг у DOM)
│   ├── Pages/ # Сторінки застосунку (розділені за функціональністю)
│   ├── Slices/ # Redux-slices для управління станом застосунку
│   │   ├── serverSlice.ts# Стан, пов'язаний із сервером або WebSocket
│   │   └── userSlice.ts    # Стан користувача (реєстрація, авторизація тощо)
│   ├── Store.ts        # Конфігурація Redux Store
│   └── vite-env.d.ts  # Декларації типів для Vite
├── tsconfig.app.json   # Налаштування TypeScript для застосунку
├── tsconfig.json       # Загальні налаштування TypeScript
├── tsconfig.node.json # Налаштування TypeScript для Node.js середовища
└── vite.config.ts      # Конфігурація Vite (шлях до alias, плагіни тощо)
```

5. Інтеграція з Git. Проєкт збережено в системі контролю версій Git.

Для віддаленого збереження змін та спільної роботи з командою створено репозиторій на GitHub.

6. CI/CD: Автоматичне розгортання на сервер DigitalOcean. Для автоматизації процесу деплою застосунку використовується GitHub Actions та SSH-скрипт на сервері. Файл конфігурації якого зображений на рис. 2.10.

```yaml
name: Deploy to DigitalOcean

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Executing remote SSH commands
        uses: appleboy/ssh-action@v1.0.0
        with:
          host: 142.93.175.150
          username: root
          key: ${{ secrets.DO_SSH_KEY }}
          script: |
            /var/www/Pocker/deploy.sh
```

Рисунок 2.10 – Файл конфігурації для Github Actions

Цей workflow запускається при пуші в гілку main, виконує SSH-з'єднання з сервером DigitalOcean (IP 142.93.175.150), виконує на сервері скрипт deploy.sh. Відповідний скрипт відображено на рис. 2.11. Після відправлення (push) змін у гілку main GitHub Actions запускає завдання, що здійснює підключення до сервера по SSH-каналу, а далі на сервері виконується скрипт deploy.sh. Він скидає локальні зміни та оновлює код з репозиторію; збирає фронтенд (React/Vite); копіює згенеровані файли в кореневу папку вебсервера /var/www/html; перезапускає nginx, щоб застосувати оновлення. У разі будь-якої помилки скрипт припиняється завдяки set-e, що забезпечує відмовостійкість.

```
cd /var/www/Pocker
set -e

git pull origin main

# Повернутися на рівень вище (до кореня проекту)
cd ..
rm -rf dist
# Клієнтська частина (frontend)
npm run build

if [ -d "dist" ] && [ "$(ls -A dist)" ]; then
    echo "Папка dist створена та не порожня, копіюємо файли..."
    rm -rf /var/www/html/*
    cp -r dist/* /var/www/html/
else
    echo "Помилка: папка dist порожня або не існує!"
    exit 1
fi

sudo systemctl restart nginx
echo "  Деплой завершено успішно."
```

Рисунок 2.11 – Список команд написані в deploy.sh

2.4 Реалізація логіки гри та інтерактивних функцій застосунку

Основна логіка застосунку реалізована на стороні клієнта з використанням бібліотеки React у поєднанні зі станом, який управляється через Redux Toolkit. Це дозволяє ефективно відображати зміну стану гри в режимі реального часу та забезпечує реактивність інтерфейсу. Процес можна описати кількома етапами.

1. Реєстрацію та авторизацію користувача реалізовано за допомогою REST API. Використовується JWT-токен для автентифікації запитів. Після входу токен зберігається в localStorage, а у Redux зберігається користувач і статус сесії. Логіка аутентифікації зображена на рис. 2.12.

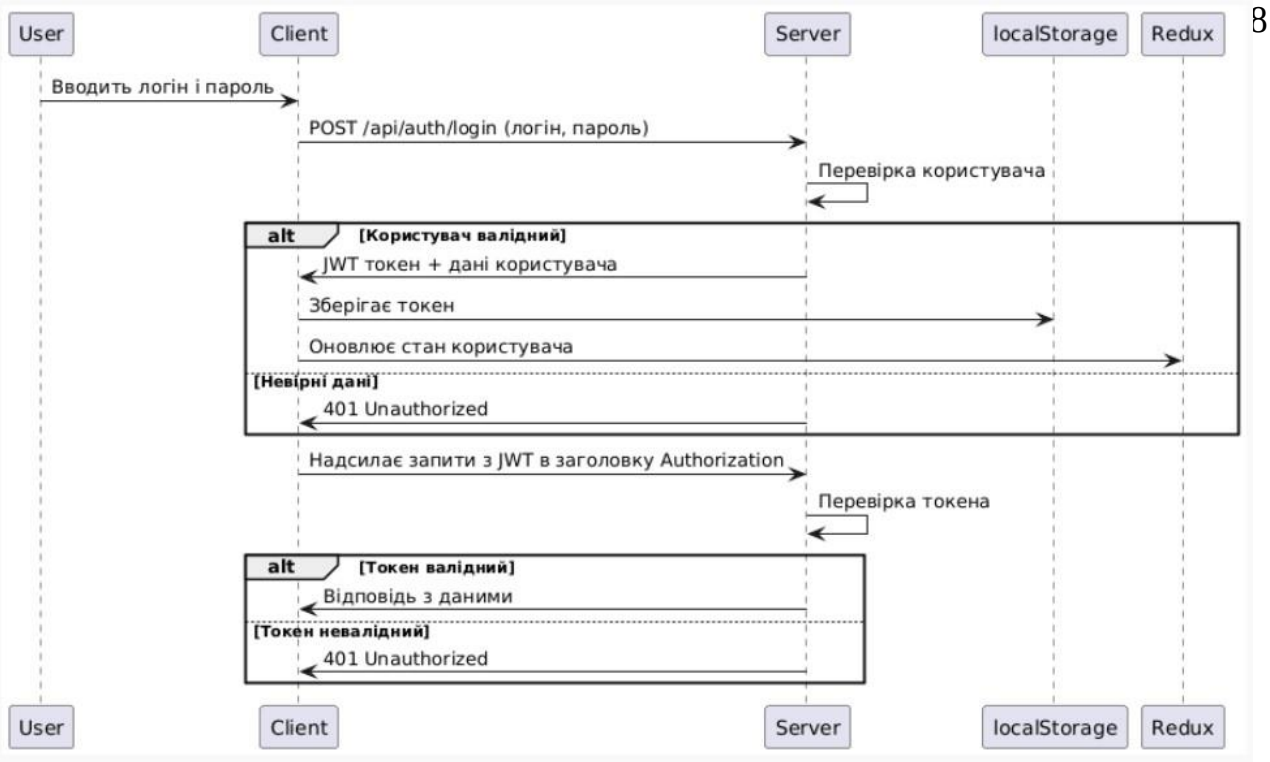

Рисунок 2.12 – Діаграма послідовності: Логіка авторизації з JWT токеном

Послідовність дій передбачає такі кроки:

1. User вводить логін і пароль у форму на клієнті.
2. Client надсилає HTTP POST-запит на /api/auth/login з логіном і паролем.
3. Server отримує запит і перевіряє дані користувача. Якщо дані невірні - повертає помилку (401 Unauthorized), інакше - генерує JWT токен.
4. Serverвідправляє у відповідь JWT токен і базову інформацію про користувача.
5. Client отримує JWT токен та зберігає його у localStorage.
6. Client також оновлює Redux стан, зберігаючи інформацію про користувача та статус сесії (авторизовано).
7. Подальші запити Client відправляє з JWT у заголовку Authorization: Bearer <token>.
8. Server при отриманні кожного запиту перевіряє JWT токен. Якщо він валідний, то запит обробляється, інакше - повертається код 401.

2. Ініціалізація кімнати та підключення гравців починається після

авторизації або реєстрації користувача. На цей момент він має можливість створити кімнату або приєднатись до вже наявної. Логіка створення кімнати реалізована на сторінці CreateRoom, де користувач задає параметри сесії гри. Інформація передається на сервер через WebSocket-з'єднання (налаштування WebSocket відбувається в окремому serverSlice.ts). Після створення кімнати всі гравці, що приєдналися, перенаправляються на сторінку GameLobbyPage, де очікують початку гри.

3. Відображення списку доступних кімнат реалізовано на сторінці Lobby Page, яка динамічно відображає список усіх активних кімнат. Список оновлюється в реальному часі за допомогою WebSocket. Кожна кімната представлена у вигляді компонента Room.tsx, який містить інформацію про її назву, кількість гравців та кнопку для приєднання.

4. Основний інтерфейс гри реалізує ігровий процес на сторінці PockerPage, де відображається головне ігрове поле, карти гравців, кнопки дій (наприклад, Check, Call, Raise, Fold) та поточний банк. Всі інтерактивні елементи пов'язані з діями Redux, які через WebSocket надсилаються на сервер, а потім синхронізуються з іншими гравцями.

5. Інші функції передбачають бокове меню (burger menu) на головній сторінці (Homepage), реалізоване як окремий компонент burgermenu.tsx. Воно дозволяє швидко переходити між сторінками. Сторінка профілю (UserPage) надає базову інформацію про користувача, з можливістю розширення функціоналу в майбутньому (наприклад, історія ігор).

2.5 Підключення до серверної частини через API

У сучасній архітектурі вебзастосунків важливою складовою є взаємодія між клієнтською частиною (frontend) та серверною частиною (backend). Для цього найчастіше використовується інтерфейс прикладного програмування (API), який забезпечує обмін даними у форматі, зручному для обох сторін, найчастіше - в форматі JSON.

У проєкті передбачено повноцінну взаємодію з серверною логікою, яка відповідає за реєстрацію та автентифікацію користувачів; створення та керування ігровими сесіями; синхронізацію стану гри між усіма учасниками; логіку ходу гравців, перевірку правил гри та визначення переможців; збереження історії ігор та статистики.

Для комунікації між клієнтом і сервером використовується протокол HTTP, що забезпечує безпечну передачу даних. API реалізовано на основі REST-архітектури, що дозволяє розділити запити за типами (GET, POST, PUT, DELETE) відповідно до операцій над ресурсами гри. Усі запити й відповіді формуються у форматі JSON, що є стандартом де-факто у веброзробці, зручним для обробки у JavaScript.

Нижче наведено опис основних маршрутів (endpoint-ів) API для управління профілю користувача, з якими працює клієнтська частина в табл. 2.1.

Після успішної аутентифікації через REST API користувач підключається до ігрового лобі за допомогою WebSocket-з'єднання. Це забезпечує двонаправлену, постійну комунікацію між клієнтом і сервером в реальному часі, що необхідно для синхронізації стану гри, повідомлень і дій гравців.

Логіка підключення та взаємодії через WebSocket передбачає низку операцій:

– встановлення з'єднання (після отримання JWT-токена клієнт відкриває WebSocket-з'єднання з сервером, передаючи токен у заголовках або як параметр запиту для аутентифікації);

– вхід у лобі (після підтвердження автентифікації сервер додає користувача до загального лобі - кімнати, де користувачі очікують на початок гри або вибір столу);

– обмін повідомленнями про стан лобі;

– сервер транслює всім підключеним клієнтам інформацію про список доступних ігор, гравців, що чекають, та інші події лобі (наприклад, новий гравець приєднався, гра почалася тощо);

– запити на створення або приєднання до гри (Користувач може надіслати через WebSocket повідомлення типу create_game або join_game з необхідними параметрами. Сервер обробляє ці запити, створює сесію гри або додає користувача до існуючої);

– синхронізація ігрового процесу (Після початку гри всі дії гравців (ходи, ставки, паси тощо) надсилаються через WebSocket у вигляді структурованих повідомлень. Сервер виконує валідацію ходів, оновлює стан гри і надсилає оновлення всім учасникам);

– обробка завершення гри і статистики(Після завершення раунду або гри сервер надсилає фінальні дані - переможця, розподіл банку, оновлення статистики, які відображаються клієнтам);

– відключення і повторне підключення (сервер підтримує механізм повторної автентифікації через токен при відновленні з'єднання, щоб користувач міг продовжити гру без втрати даних);

Таблиця 2.1 – Основні маршрути API для керування користувачами Підключення користувача до ігрового лобі через WebSocket

Метод	URL	Опис
GET	/api/user/email?email=	Пошук користувача за email
GET	/api/user/name?name=	Пошук користувача за name
GET	/api/user/auth?email=&passwo rd=	Аутентифікація користувача
GET	/api/user	Виведення всіх користувачів
GET	/api/user/profile	Аутентифікація користувача за допомогою JWT токена
POST	/api/user	Створення нового користувача за введеними даними
PATCH	/api/user	Змінення параметрів користувача
DELETE	/api/user?email=&password=	Видалення користувача з перевіркою паролю
POST	/api/auth/login	Видача JWT токена

Формат повідомлень WebSocket всі повідомлення мають формат JSON з полями: type - тип повідомлення (наприклад, join_lobby, game_action, update_state); payload - дані, які передаються (параметри ходу або інформація про користувача); token - JWT-токен для підтвердження автентифікації (при необхідності).

Отже, у другому розділі було виконано повний цикл проєктування та реалізації клієнтської частини вебзастосунку: визначено функціональні та нефункціональні вимоги, розроблено структуру сайту, налаштовано середовище розробки та реалізовано ключові ігрові та інтерактивні функції. Особливу увагу приділено інтеграції з серверною частиною через API, яка забезпечує динамічну взаємодію між користувачем і застосунком. Проведена робота заклала надійну основу для стабільної та зручної роботи інтерфейсу, забезпечивши гнучкість для подальшого розвитку й удосконалення функціоналу.

РОЗДІЛ 3
ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1 Перелік і обґрунтування обраних методів тестування

Для забезпечення стабільності та безпеки роботи клієнтської частини вебзастосунку були обрані наступні методи тестування.

Модульне тестування [14] дає змогу ізолювати окремі частини коду - такі як React-компоненти, Redux-дії, редюсери та допоміжні функції - і перевірити їхню поведінку незалежно від решти застосунку. Це дозволяє не лише оперативно виявляти логічні помилки та регресії, а й значно полегшує підтримку та масштабування проєкту. Завдяки модульним тестам можна з упевненістю вносити зміни в кодову базу,.

Тестування безпеки [15] зосереджується на виявленні потенційних вразливостей у клієнтській частині, які можуть виникати під час обробки введення користувачем даних, збереження конфіденційної інформації (наприклад, токенів автентифікації), а також у разі неправильного поводження з HTML/JavaScript-даними. Особливу увагу приділено захисту від поширених атак, таких як XSS (міжсайтовий скриптинг), що можуть виникати під час небезпечного виведення даних у DOM, або витоків особистої інформації через ненадійне використання localStorage.

Інтеграційне тестування зосереджене на перевірці взаємодії між різними модулями клієнтської частини вебзастосунку – зокрема, між React-компонентами, Redux-станом і зовнішніми API. Таке тестування дозволяє виявити помилки, що можуть виникати на стиках між окремими частинами системи, навіть якщо кожна з них окремо працює коректно. Наприклад, перевіряється правильність обробки відповіді від сервера, оновлення стану застосунку після запиту, а також коректне відображення відповідної інформації в інтерфейсі. Завдяки цьому забезпечується стабільна взаємодія всіх частин системи в умовах, наближених до реального сценарію використання.

Інтеграційні тести проводилися вручну, що дало змогу безпосередньо оцінити поведінку системи під час типових дій користувача, оперативно виявити помилки у логіці взаємодії модулів і забезпечити належну якість роботи застосунку.

Для реалізації тестів використовується сучасна бібліотека Vitest, яка забезпечує швидке й ефективне тестування в середовищі, близькому до реального виконання застосунку. Окрім цього, було проведено ручне тестування для додаткової перевірки функціональності та виявлення можливих недоліків у роботі інтерфейсу.

3.2 Тестовий план проєкту (демонстрація обраного набору тестів)

На основі обраних методів тестування, був сформований тестовий план, що охоплює ключові аспекти функціональності клієнтської частини вебзастосунку. Метою тестового плану є виявлення помилок на етапах розробки, перевірка правильності логіки роботи окремих компонентів інтерфейсу, а також забезпечення базового рівня безпеки клієнтської частини гри. Основні напрямки тестування:

Модульне тестування (Unit Testing):

- Тестування окремих компонентів React, зокрема форм (реєстрації, входу), елементів інтерфейсу гри (ігрове поле, карти, кнопки керування), а також Redux-логіки (actions, reducers, selectors).
- Використання мок-даних та емуляція взаємодії користувача для перевірки реакції компонентів на події (натискання, введення, навігація).
- Верифікація коректної валідації форм та обробки користувацького введення.

Тестування безпеки (Security Testing):

Аналіз захищеності компонентів від міжсайтового скриптингу (XSS) шляхом спроби інжекції шкідливого коду у поля введення.

Перевірка коректного зберігання й обробки автентифікаційного токена у

Емуляція потенційно небезпечних дій, як-от ручна модифікація DOM чи доступ до внутрішніх змінних через консоль браузера.

Приклад реалізації модульного тесту

Одним із прикладів реалізованого модульного тесту є перевірка роботи сторінки реєстрації.

Лістинг 3.1 – Код модульного тесту сторінки реєстрації

```
      test("рендерить форму реєстрації", () => {
        setup();

expect(screen.getByText(/Реєстрація/i)).toBeInTheDocument();
            expect(screen.getByRole("button", { name: /Submit/i
})).toBeInTheDocument();
      });
});
test("показує помилки при пустих полях", async () => { setup();
  const submit = screen.getByRole("button", { name: /Submit/i

  fireEvent.click(submit);
  await waitFor(() => {
     expect(screen.getByText(/Невведено
ім'я/)).toBeInTheDocument();
          expect(screen.getByText(/Невведено
email/)).toBeInTheDocument();
          expect(screen.getByText(/Невведено
пароль/)).toBeInTheDocument();

          expect(screen.getByText(/Невведено пароль для
перевірки/)).toBeInTheDocument();
          expect(screen.getByText(/Введіть коректний
email!/)).toBeInTheDocument();
        });
        expect(localStorage.setItem).not.toHaveBeenCalled();
```

```
});
```


Цей фрагмент коду реалізує модульне тестування сторінки реєстрації користувача з використанням бібліотеки Vitest у поєднанні з React Testing Library. У тесті перевіряється коректність рендерингу форми реєстрації, обробка помилок при заповненні форми, а також логіка збереження токена аутентифікації після успішної реєстрації. Тест симулює дії користувача, зокрема взаємодію з кнопкою та введення даних, після чого перевіряє відповідні зміни у DOM-структурі й виклики методів збереження даних. Такий підхід дозволяє впевнено виявляти помилки на ранньому етапі та забезпечує стабільну роботу інтерфейсу за різних сценаріїв використання. З іншими тестами можна ознайомитись в додатку А.

Окрім перевірки базової логіки форми, тестування також охоплює захист від поширених вебзагроз, зокрема:

- XSS-атак (міжсайтового скриптингу) - перевіряється, що введення користувача належним чином екранується й не виконується як скрипт. Наприклад, при введенні <script>alert("xss")</script> на фронтенді не повинна виконуватись жодна небезпечна дія, і цей вміст має бути відображений як текст.
- SQL-ін'єкцій - хоча основна логіка роботи з базою даних знаходиться на серверній стороні, клієнтська частина також перевіряється на те, щоб не допускати шкідливого введення. Це досягається через валідацію введених даних, обмеження на символи, а також використання параметризованих запитів на бекенді.
- Маніпуляції з localStorage - перевірено, що токени зберігаються без зайвих даних, доступ до них захищений, і вони не містять конфіденційної інформації у відкритому вигляді.

У рамках інтеграційного тестування було також перевірено взаємодію окремих функцій у контексті роботи всього застосунку. Зокрема, вручну тестувалися сценарії, у яких декілька частин системи – функції обробки ігрової логіки, оновлення стану через Redux та відображення даних у React-компонентах – працюють разом у єдиному потоці. Таке тестування дозволило

виявити помилки у передачі даних між шарами застосунку та переконатися в їх узгодженій роботі при різних сценаріях використання. Як приклади таких інтеграційних тестів можна навести наступні ситуації:

Для першого тестування було проведено перевірку сценарію входу користувача в акаунт. На першому кроці користувач взаємодіє з формою входу, зображено на рис 3.1, де вводить свої облікові дані — логін та пароль. Після натискання кнопки "Увійти" система обробляє запит і при успішній автентифікації перенаправляє користувача на захищену сторінку особистого кабінету, дивись на рис 3.2.

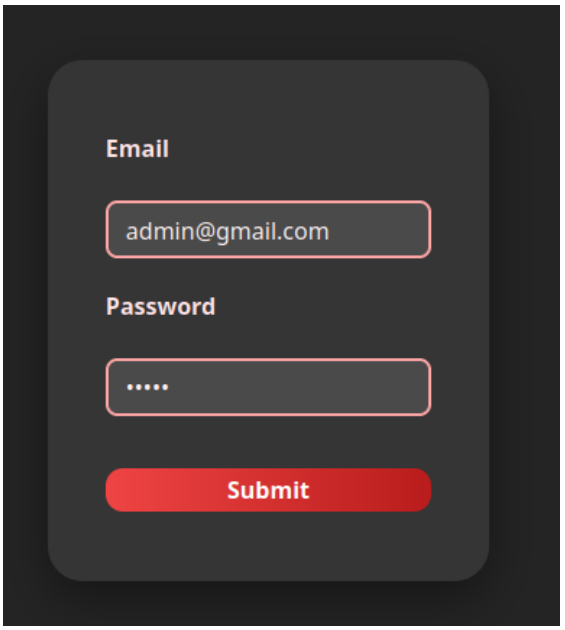

Рисунок 3.1 – Заповнена форма входу

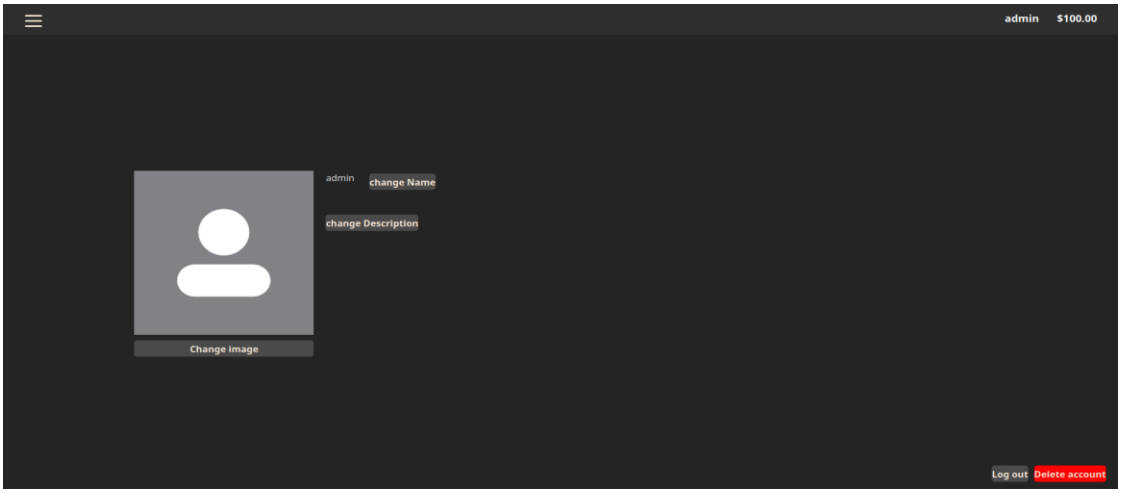

Рисунок 3.2 – Підтвердження входу

Цей тест дозволяє переконатися, що:

- форма коректно приймає та валідує введені дані;
- взаємодія між компонентами React, Redux та API працює узгоджено;
- після успішного входу відбувається правильне оновлення стану застосунку;
- користувач бачить відповідний інтерфейс із підтвердженням входу.

Таким чином, перевіряється коректність комплексної роботи різних частин клієнтської частини вебзастосунку у рамках реального сценарію використання.

Другий тест інтеграційного тестування клієнтської частини також охоплює основні сценарії ігрового процесу, що включають кілька ключових перевірок:

Підключення до гри. Перевіряється, чи коректно відбувається підключення користувача до ігрового столу, а також чи відображається він у списку учасників. Це забезпечує початкову синхронізацію стану гри між клієнтом і сервером. Зображено на рис 3.3. і 3.4.

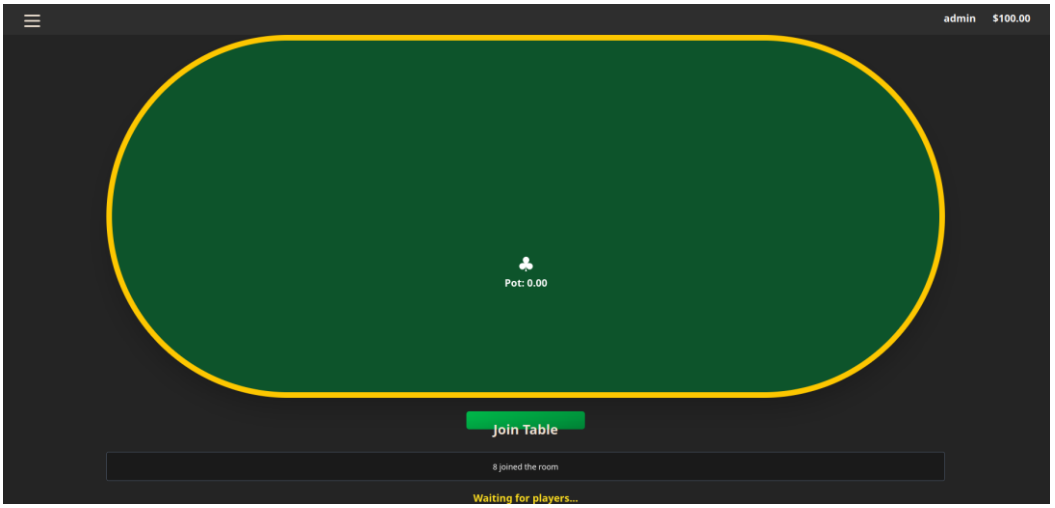

Рисунок 3.3 – Стіл гри до підключення

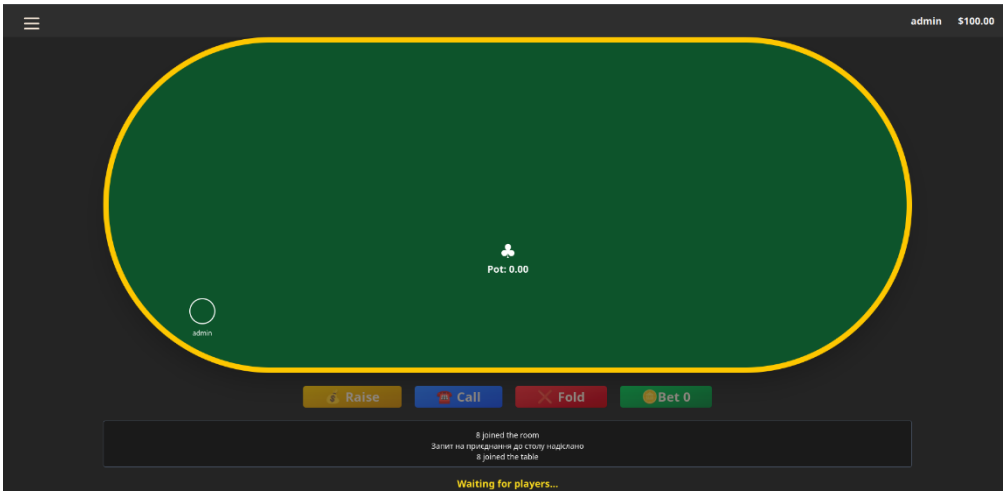

Рисунок 3.4 – Стіл гри після підключення

Підключення інших учасників. Тестування включає перевірку появи інших гравців у кімнаті, що гарантує правильне оновлення інформації про поточний склад учасників для всіх клієнтів. Зображено на рис 3.5.

Робота зі ставками та передача ходу. Перевіряється, чи можна успішно зробити ставку під час свого ходу, а також чи передається хід наступному гравцю відповідно до правил гри. Це підтверджує коректність логіки керування ігровим процесом і взаємодії компонентів. Зображено на рис 3.6.

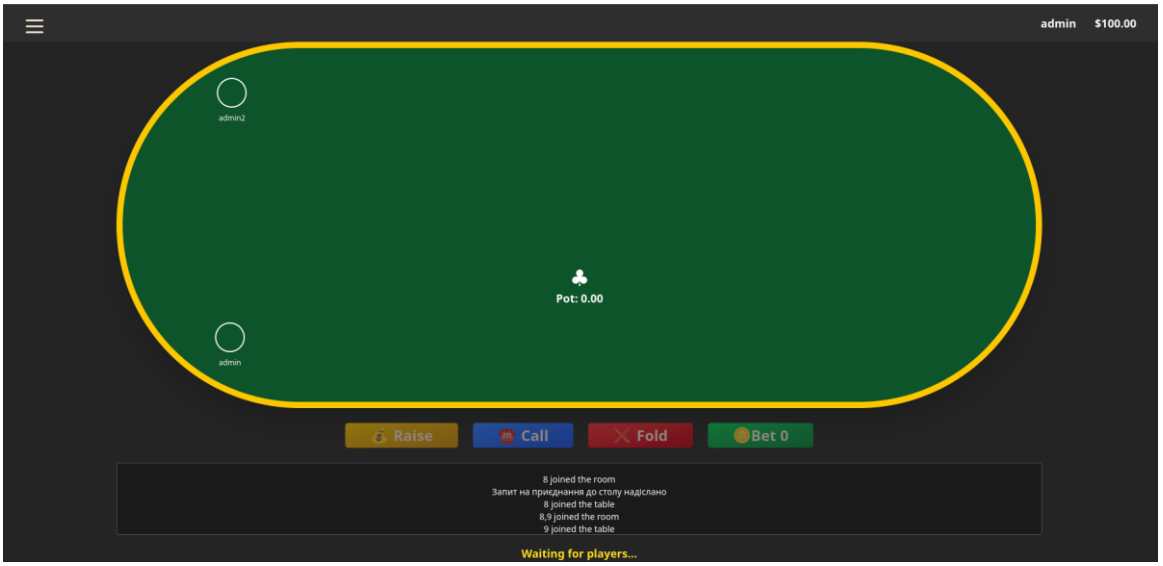

Рисунок 3.5 – Стіл гри з декількома гравцями

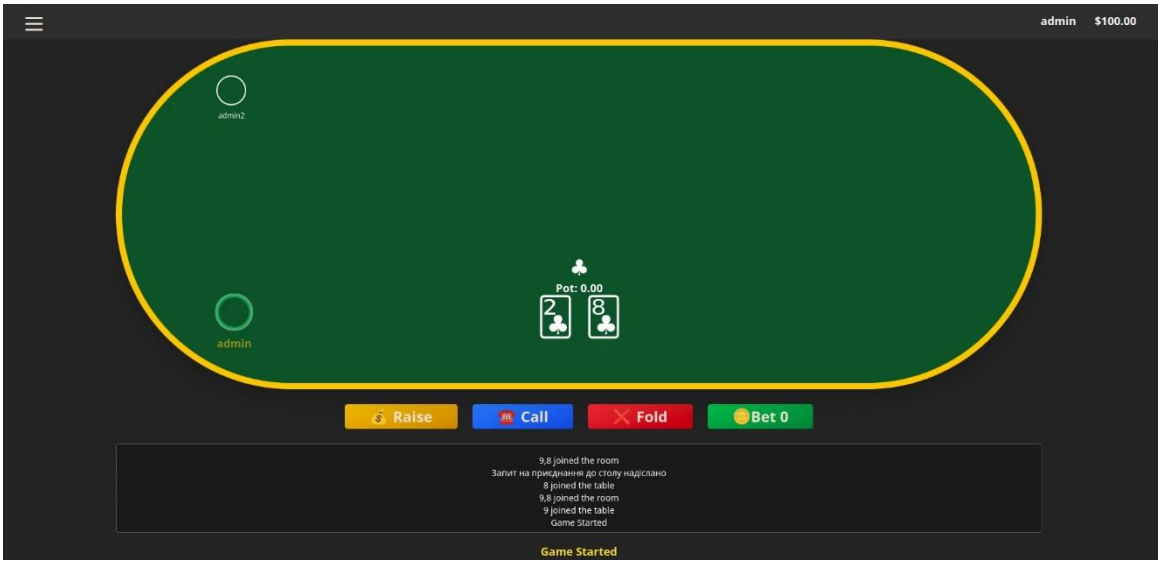

Рисунок 3.6 – Надання можливості керування гравцеві

Виведення карт. Після завершення кожного етапу роздачі (флоп, терн, рівер) перевіряється правильність відображення відповідних карт на ігровому полі. Також перевіряється коректний вивід карт кожного гравця під час відкриття, зокрема переможця – для забезпечення прозорості гри та інформування всіх учасників про результат. Зображено на рис 3.7.

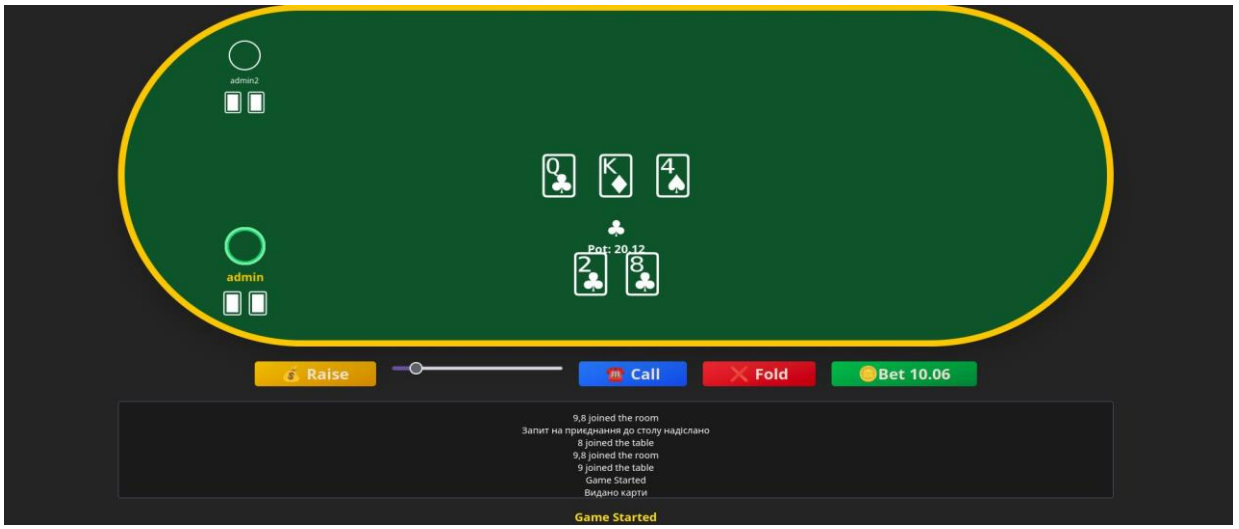

Рисунок 3.7 – Роздача карт

Визначення та відображення переможця. Після завершення роздачі перевіряється правильність визначення переможця та відображення відповідного повідомлення в інтерфейсі, що є важливим для узгодженості результатів гри та користувацького досвіду. Зображено на рис 3.8.

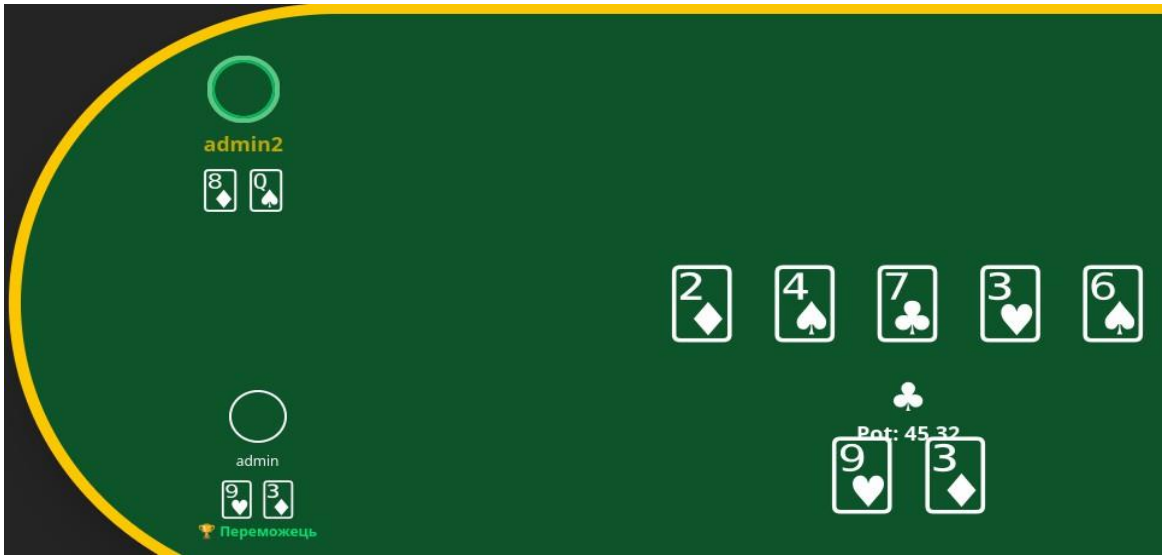

Рисунок 3.8 – Показ переможця

Дані тести забезпечують всебічну перевірку інтеграції ключових функцій ігрової логіки, стану застосунку та відображення даних у реальному часі, що є критичним для стабільної та надійної роботи покерного застосунку.

3.3 Аналіз отриманих результатів

В результаті проведеного тестування клієнтської частини вебзастосунку з реалізацією карткової гри було досягнуто наступних ключових результатів:

Стабільність роботи компонентів. Модульні тести підтвердили коректність логіки окремих React-компонентів, форм та Redux-станів. Завдяки ізольованому тестуванню були виявлені і усунуті початкові помилки у валідації даних і реакції інтерфейсу на дії користувача. Це підвищило загальну стабільність роботи застосунку.

Відповідність вимогам користувача. Тестування підтвердило, що інтерфейс коректно відображає всі необхідні елементи, а також що валідація форм працює відповідно до специфікацій. Користувач отримує зрозумілі повідомлення про помилки, що позитивно впливає на юзабіліті.

Безпека клієнтської частини. Проведені security-тести дозволили виявити потенційні ризики, пов'язані з обробкою введених даних. Застосовані механізми

захисту ефективно запобігають основним типам атак, таким як XSS, а також забезпечують безпечне зберігання токенів. Це гарантує базовий рівень захищеності даних користувачів.

Інтеграційне тестування ігрового процесу. Було реалізовано інтеграційне тестування, яке охоплює повну взаємодію між користувачем, інтерфейсом та серверною частиною через WebSocket. Зокрема, протестовано підключення до ігрового столу, приєднання інших гравців, логіку ставок, передачу ходу, визначення переможця та відображення його карт. Це забезпечує цілісність функціонування гри та узгодженість між усіма компонентами системи.

Швидкість та зручність тестування. Використання бібліотеки Vitest забезпечило швидке виконання тестів та легкість їх написання й підтримки. Це дозволило оперативно впроваджувати нові тести при розширенні функціональності.

Гнучкість та масштабованість тестового плану. Запропонований підхід до тестування можна розширювати і адаптувати під майбутні зміни в застосунку, що є важливим фактором для підтримки якості в довгостроковій перспективі.

Таким чином, проведене тестування не лише забезпечило високу якість клієнтської частини, а й створює фундамент для подальшого розвитку проєкту з урахуванням вимог як до функціональності, так і до безпеки.

ВИСНОВКИ

Отже, було розглянуто та реалізовано клієнтську частину вебзастосунку з картковою грою, а також проведено аналіз, проєктування, розробку та тестування.

У першому розділі було проведено аналіз ринку вебзастосунків, порівняння наявних рішень та виділено ключові особливості сучасних вебзастосунків. Далі було обрано оптимальні технології для розробки клієнтської частини.

Другий розділ присвячений проєктуванню та реалізації застосунку. Було визначено функціональні та нефункціональні вимоги, спроєктовано структуру сайту, налаштовано середовище розробки та реалізовано основну логіку гри разом з інтерактивними елементами. Також забезпечено інтеграцію з серверною частиною через API.

У третьому розділі описано вибрані методи тестування, які включають модульне тестування для перевірки логіки компонентів та security-тестування для забезпечення захисту від типових вразливостей. Розроблено тестовий план, що дозволив систематично перевірити функціональність, стабільність та безпеку клієнтської частини. Аналіз отриманих результатів підтвердив відповідність застосунку вимогам, а також ефективність застосованих технологій та підходів.

Виконана робота демонструє комплексний підхід до розробки вебзастосунку, який поєднує сучасні технології, увагу до безпеки та якісне тестування. Запропонований клієнтський застосунок є надійним, зручним у використанні та має потенціал для подальшого розвитку і масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Figma. Офіційна довідка. URL: https://help.figma.com/ (дата звернення: 05.12.2024)

2. TopTal. Figma vs Sketch vs Adobe XD: Design Tool Comparison. - URL: https://www.toptal.com/designers/ui/figma-vs-sketch-vs-adobe-xd (accessed: 05.12.2024)

3. Adobe XD. Офіційний сайт. URL: https://www.adobe.com/products/xd.html (дата звернення: 05.12.2024)

4. Sketch. Інструмент дизайну для macOS. - URL: https://www.sketch.com/ (дата звернення: 05.12.2024)

5. Вікіпедія Github URL: https://uk.wikipedia.org/wiki/GitHub(дата звернення: 05.12.2024)

6. CSS GRID and FLEXBOX URL: https://mate.academy/blog/front-end-and-js/css-grid-and-flexbox/ (дата звернення: 05.12.2024)

7. Офіційна довідка по TAILWIND CSS URL https://tailwindcss.com/docs/installation/using-vite (дата звернення: 05.12.2024)

8. Офіційна довідка по AXIOS: https://axios-http.com/docs/intro (дата звернення: 05.12.2024)

9. Офіційна документація по TYPESCRIPT URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html (дата звернення: 05.12.2024)

10. Офіційна документація по WEBSOCKETS https://uk.javascript.info/websocket (дата звернення: 06.12.2024)

11. Документація по SOCKET.IO URL: https://socket.io/docs/v4/ (дата звернення: 06.12.2024)

12. Довідка по Vitest URL: https://vitest.dev/guide/why.html (дата звернення: 06.12.2024)

13. The Growth of Web-Based Gaming Platforms. URL: https://cyberogism.com/web-gaming-platforms-growth/?utm_source=chatgpt.c om

(accessed: 06.12.2024)

14. Стаття про модульне тестування. URL: https://qalight.ua/baza-znaniy/modulne-testuvannya/ (дата звернення 20.02.2025)

15. Стаття про тестування безпеки. URL: https://qalight.ua/baza-znaniy/testuvannya-bezpeki/ (дата звернення 20.02.2025)

ДОДАТОК А

Посилання на репозиторій URL: https://github.com/dimon289/Pocker