

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ СИМУЛЯТОРА ІГРОВИХ МЕХАНІК
SOFTWARE IMPLEMENTATION OF A GAME MECHANICS SIMULATOR

Виконав(ла): студент(ка) 4 курсу, групи КНТ-132

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ТЮПА А.К.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ТЮПИ Андрія Костянтиновича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація симулятора ігрових механік.
Software Implementation of a Game Mechanics Simulator

керівник проєкту (роботи) д-р філософ., доцент, ЛЕОЩЕНКО Сергій
Дмитрович,

(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація,
тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень,
 кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтроль	КАЛІНІНА М.В., зав. лабораторії		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Андрій ТЮПА
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій ЛЕОЩЕНКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 84 с., 6 табл., 42 рис., 3 дод., 18 джерел.

СИМУЛЯТОР, ВЕБЗАСТОСУНОК, ІГРОВІ МЕХАНІКИ, ТЕОРІЯ ЙМОВІРНОСТЕЙ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, WEBSOCKET, REACT, NODE.JS.

Об'єкт дослідження – процес програмної симуляції імовірнісних ігрових механік у вебсередовищі.

Предмет дослідження – програмне забезпечення для симуляції ігрових механік.

Мета роботи – підвищення доступності та безпеки аналізу ігрових математичних моделей шляхом створення веборієнтованого програмного комплексу для симуляції ігор «Блекджек» та «Рулетка» з підтримкою синхронізації даних у реальному часі.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, клієнт-серверна архітектура, протокол WebSocket, мова програмування JavaScript, бібліотека React, середовище виконання Node.js, система управління базами даних PostgreSQL, персональний комп'ютер під управлінням операційної системи Microsoft Windows.

Результати. Створено веборієнтований клієнт-серверний додаток, який забезпечує точну математичну симуляцію ігрових процесів на базі генератора псевдовипадкових чисел, реалізує безпечне збереження прогресу через JWT-автентифікацію та підтримує двосторонній обмін даними без затримок.

Висновки. Розроблено програмну систему для симуляції ігрових механік, яка успішно пройшла функціональне тестування, довела свою математичну достовірність та високу стабільність під час обробки віртуальних фінансових транзакцій.

Галузь використання – освітній процес у закладах вищої освіти для емпіричної демонстрації законів теорії ймовірностей та комп'ютерного моделювання, а також індивідуальне використання для безпечного тестування математичних та ігрових стратегій.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 84 pages, 6 tables, 42 figures, 3 appendixes, 18 sources.

SIMULATOR, WEB APPLICATION, GAME MECHANICS, PROBABILITY THEORY, CLIENT-SERVER ARCHITECTURE, WEBSOCKET, REACT, NODE.JS.

The object of research is the process of software simulation of probabilistic game mechanics in a web environment.

The subject of research is software for simulating game mechanics.

The purpose of the work is to increase the accessibility and security of the analysis of game mathematical models by creating a web-based software package for simulating the games "Blackjack" and "Roulette" with support for real-time data synchronization.

Materials, methods and technical means: object-oriented programming, client-server architecture, WebSocket protocol, JavaScript programming language, React library, Node.js runtime environment, PostgreSQL database management system, personal computer running the Microsoft Windows operating system.

Results. A web-based client-server application was created that provides accurate mathematical simulation of game processes based on a pseudo-random number generator, implements secure progress saving through JWT authentication, and supports two-way data exchange without delays.

Conclusions. A software system for simulating game mechanics has been developed, which has successfully passed functional testing, proven its mathematical reliability and high stability when processing virtual financial transactions.

Field of use – educational process in higher education institutions for empirical demonstration of the laws of probability theory and computer modeling, as well as individual use for safe testing of mathematical and gaming strategies.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	9
Вступ.....	10
1 Аналіз проблеми та постановка завдань дослідження	12
1.1 Аналіз предметної області.....	12
1.2 Аналіз математичних моделей ігрових механік	14
1.3 Огляд існуючих систем, що вирішують аналогічні задачі	18
1.3.1 Vojoko Blackjack Trainer	18
1.3.2 CalcBE Probability Simulator.....	19
1.3.3 Vojoko Roulette Trainer.....	20
1.3.4 Порівняння описаних систем.....	21
1.4 Постановка завдань роботи.....	22
1.5 Висновки за розділом 1	24
2 Розробка архітектури програми.....	25
2.1 Концептуальне проєктування та вибір методології розробки.....	25
2.2 Обґрунтування вибору інструментальних засобів	25
2.2.1 Вибір Frontend-технології	25
2.2.2 Вибір Backend-платформи	27
2.2.3 Вибір системи керування базами даних	28
2.2.4 Вибір середовища розробки.....	29
2.3 Висновки за розділом 2	30
3 Розробка програми	31
3.1 Загальні відомості та функціональне призначення програми.....	31
3.2 Вимоги до складу технічних та програмних засобів.....	31
3.3 Моделювання динамічних та статичних аспектів системи	32
3.3.1 Діаграма прецедентів.....	32
3.3.2 Діаграма компонентів	32
3.3.3 Діаграма послідовності.....	33

3.4	Проектування структури даних	34
3.5	Опис логічної структури програми	35
3.6	Початкові та вихідні дані	36
3.7	Програмна реалізація клієнтського інтерфейсу на базі React	38
3.8	Система верифікації, безпеки та допоміжні сервіси	40
3.9	Використані технічні засоби, виклик і завантаження програми	41
3.10	Висновки за розділом 3	41
4	Експлуатація, тестування та експериментальне дослідження програми	43
4.1	Керівництво програміста	43
4.2	Керівництво системного програміста	43
4.3	Керівництво оператора	44
4.3.1	Виконання програми	44
4.3.2	Повідомлення оператору	53
4.4	Методика та результати тестування	56
4.5	Експериментальні дослідження та аналіз результатів	57
4.6	Висновки до розділу 4	58
	Висновки	59
	Перелік джерел посилання	61
	Додаток А Технічне завдання	63
	Додаток Б Текст програми	67
	Додаток В Слайди презентації	78

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- CSS – Cascading Style Sheets;
- DOM – Document Object Model;
- HTML – HyperText Markup Language;
- JSON – JavaScript Object Notation;
- JWT – JSON Web Token;
- SMTP – Simple Mail Transfer Protocol;
- URL – Uniform Resource Locator;
- БД – база даних.

ВСТУП

У сучасному світі, де інформаційні технології швидко міняються, створення інтерактивних систем для моделювання складних імовірнісних процесів стає все більш важливим. Симулятори ігрових механік відіграють ключову роль у аналізі математичних моделей і розвитку логічного мислення. Хоча деякі проекти працюють добре технічно, навчальний потенціал залишається недовикористаним. Часто статистика користувача не фіксується взагалі, або опиняється закритою для аналізу. Ситуація вимагає переходу від простих скриптових моделей до повноцінних клієнт-серверних вебзастосунків у реальному часі, які можуть забезпечити високу точність симуляції та надійність збереження даних.

Актуальність цієї теми зумовлена тим, що сучасні вебтехнології досягли такого рівня, що дають змогу запускати складні логічні й графічні системи просто в браузері. У зв'язку з глобальною цифровізацією освіти зростає попит на доступні інтерактивні тренажери, але існуючі безкоштовні аналоги часто мають обмежений функціонал або застарілу технічну базу. Це підкреслює необхідність розробки сучасного симулятора на основі React та Node.js.

Об'єктом дослідження в цій роботі є процес програмної симуляції імовірнісних ігрових механік у вебсередовищі. А предметом дослідження є методи та засоби розробки клієнт-серверних вебзастосунків, які використовують технології React, Node.js та WebSocket для створення інтерактивних симуляторів. Визначені об'єкт і предмет дозволяють сформулювати основну мету роботи: підвищити доступність та безпеку аналізу ігрових моделей шляхом розробки веборієнтованого програмного комплексу для симуляції механік ігор «Блекджек» та «Рулетка», з підтримкою синхронізації даних користувачів у реальному часі.

Для досягнення поставленої мети в цій роботі було вирішено ряд завдань. Серед них – аналіз теоретичних основ та математичних принципів

ігрових механік, проведення порівняльного огляду вже існуючих систем для виявлення їхніх недоліків, а також розробка архітектури системи і проєктування реляційної бази даних для обліку результатів. Не менш важливим був етап програмної реалізації серверного компонента та модульного інтерфейсу користувача, що включало подальше дослідження працездатності системи та перевірку точності ігрових алгоритмів.

Під час дослідження використовували об'єктно-орієнтований підхід – це допомогло чітко спроектувати програмне забезпечення. Технологічний стек обирали шляхом системного аналізу, щоб уникнути довільних рішень. Результати симуляції перевіряли через функціональне тестування, яке показало їхню достовірність.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області

Ігрова індустрія та моделювання ймовірнісних процесів мають ґрунтовну історичну та математичну основу. Коли люди почали шукати закономірності в невизначеності, виникла теорія ймовірностей. Думки про хаос і порядок привели до перших наукових робіт з цієї теми. Серед тих, хто заклав фундамент – Паскаль, Ферма, Гюйгенс. Аналізуючи ставки в грі, вони несподівано сформували новий напрям у науці. У сучасний період математичне моделювання ігрових механік вийшло далеко за межі простих розваг. У фінансах, економіці, шифруванні чи технологіях штучного інтелекту вони допомагають передбачати можливі сценарії. Невизначеність більше не просто хаос – вона стала предметом аналізу. Минаючи розважальні корені, ці методи все частіше служать стратегічному плануванню.

Сьогодні замість тривалих фізичних експериментів люди покладаються на потужні цифрові симуляції, щоб вивчати непередбачувані події. Ще кілька десятиліть тому дослідники не мали іншого вибору – кожна перевірка означала тисячі однакових операцій – такі як пускати кульку по колесу рулетки або підкидати монету. Достатньо лише лічених секунд для того, щоб сучасне програмне забезпечення виклало складну серію подій одна за одною. Замість старих методів тепер цифрові інструменти формують новий погляд на аналіз теоретичних гіпотез. Безперервні чисельні експерименти стають можливими завдяки обчислювальним ресурсам, які дозволяють слідкувати за поведінкою систем у реальному часі. Цифрова симуляція замінює класичні форми практичного дослідження. Так стало можливим створення програм для моделювання процесів генерації випадкових подій, які повторюють появу випадкових результатів за допомогою алгоритмів псевдовипадкових чисел. Попри це, більшість існуючих рішень працюють переважно на користь азартним іграм, залишаючи потребу у надійних освітніх середовищах.

Незалежно від того, хто перед екраном: учень, що опановує основи теорії ймовірностей, чи спеціаліст, який перевіряє складні фінансові стратегії, потребує надійного інструменту, здатного не лише імітувати ігровий процес, але й забезпечувати детальну статистику, яка підтверджує або спростовує математичне очікування гри. Наприклад, у разі американської рулетки наявність двох нульових секторів забезпечує закладу математичну перевагу у 5,26%. Хоча формули пояснюють механізм цієї переваги, багато хто не відчуває її реального ефекту доти, доки не побачить результат симуляції великої кількості ігор. Тисячі симуляцій показують: гравець поступово втрачає кошти, навіть якщо час від часу перемагає.

Психологія прийняття рішень має не менше значення, ніж математика, у цьому контексті. Через симулятори ігрових механік люди потрапляють у середовище, подібне до захищеної зони для експериментів. У такий спосіб можна випробувати стратегії, які гравець вважає надійними, замість того щоб ставити справжні кошти. Ризиковане поводження отримує простір для прояву без шкоди для гаманця. Такі інструменти знімають тиск можливого програшу, перетворюючи дослідження на цікавий та безпечний процес. З сучасних психологічних робіт з теми лудоманії видно: коли люди самі бачать, що популярні ігрові стратегії не працюють у захищеному середовищі, шанс, що вони спробують таку ж саму стратегію, але вже у реальні грі – замалий. Таким чином, подібні інструменти набувають профілактичного та соціального значення.

Із погляду технологій, предметна область поступово переходить із окремих десктопних рішень до розподілених вебзастосунків, що обумовлено потребою у кросплатформеності та доступності. Доступ до симуляції має бути можливий скрізь, де є з'єднання з мережею. Це ставить перед розробниками нові виклики: забезпечення безперервного зв'язку між клієнтом та сервером. Запобігання підробці даних на боці клієнта також має значення. Не менше важливою є реалізація механізмів надійної авторизації та реєстрації. Відповідно, проектування таких систем вимагає застосування сучасних

архітектурних підходів, спрямованих на синхронізацію стану гри в режимі реального часу.

Підсумовуючи, предметна область розробки веборієнтованого симулятора ігрових механік є багатогранною. Вона охоплює глибокий математичний аналіз імовірнісних моделей та проектування надійних клієнт-серверних архітектур для роботи в реальному часі. Реалізація такого програмного комплексу дозволить подолати розрив між академічною теорією та практичним моделюванням ризиків, забезпечивши користувачів ефективним інструментом для навчання.

1.2 Аналіз математичних моделей ігрових механік

База кожного імовірнісного симулятора – це математика. Точне відображення законів розподілу ймовірностей необхідне, інакше модель не матиме освітньої або дослідницької цінності. У цьому проекті взято дві відомі ігри, протилежні за своєю природою: американську рулетку обрано тому, що вона є грою з незалежними подіями та блекджек було обрано через залежні події.

Колесо рулетки працює за принципом повної незалежності подій – типовий випадок схеми Бернуллі. Кожен наступний запуск колеса не враховує те, що було раніше, минулі результати просто не мають значення. У американській версії – тридцять вісім можливих положень кульки після оберту. Тут зустрічаються цифри від одного до тридцяти шести, по чергово червоні й чорні. Два додаткові сегменти виділені зеленим кольором: один позначений як «0», інший – «00». На цьому тлі система виглядає трохи інакше, аніж у європейському форматі. Саме ця пара нульових полів створює іншу картину математичного очікування.

Ймовірність випадіння будь-якого конкретного числа розраховується за класичною формулою ймовірності (1.1):

$$P = \frac{m}{n} = \frac{1}{38} \approx 2.63\%, \quad (1.1)$$

де P – загальна кількість всіх рівноможливих результатів експерименту;

m – кількість сприятливих результатів;

n – загальна кількість можливих результатів.

Для ставок на рівні шанси, а саме: червоне/чорне (Red/Black), парне/непарне (Even/Odd) та більше/менше («19 to 36» / «1 to 18»), ймовірність виграшу становить (1.2):

$$P = \frac{18}{38} \approx 47.37\%, \quad (1.2)$$

Звідси очевидно, що ймовірність програшу завжди вища (52.63%) за рахунок наявності двох зелених секторів.

Для ставок на третини – дюжини («1st 12», «2nd 12», «3rd 12») та колонки («2:1») – гравець перекриває 12 номерів. Ймовірність виграшу в такому випадку становить (1.3):

$$P = \frac{12}{38} \approx 31.58\%, \quad (1.3)$$

У разі виграшу виплата складає 2 до 1 (гравець отримує 2 фішки на кожну поставлену).

Математичне очікування (Expected Value, $M(x)$) дискретної випадкової величини є ключовим показником рентабельності гри і розраховується за загальною формулою (1.4):

$$M(x) = \sum xP(x), \quad (1.4)$$

де x – можливий результат (виграш або програш);

$P(x)$ – ймовірність цього результату.

Застосовуючи цю формулу для ставок на рівні шанси при ставці в 1 умовну одиницю, ми знаходимо суму добутків чистого виграшу на його ймовірність та втрати на ймовірність програшу (1.5):

$$EV = \left(2 * \frac{12}{38}\right) + \left(-1 * \frac{26}{38}\right) = \frac{24-26}{38} = -\frac{2}{38} \approx -0.0526, \quad (1.5)$$

Унікальною математичною властивістю рулетки є те, що математичне очікування залишається константою незалежно від типу ставки. Наприклад, розрахуємо математичне очікування для ставки на дюжину (1.6):

$$EV = \left(1 * \frac{18}{38}\right) + \left(-1 * \frac{20}{38}\right) = \frac{18-20}{38} = -\frac{2}{38} \approx -0.0526, \quad (1.6)$$

Цей показник, що становить -5.26%, називається перевагою закладу. Він математично гарантує, що на нескінченно довгій дистанції гравець неминуче втратить 5.26% від сумарного обсягу всіх своїх ставок. Симуляція цього процесу дозволяє наочно продемонструвати користувачам хибність стратегій на кшталт «Мартінгейла», які не можуть подолати від'ємне математичне очікування.

На відміну від рулетки, у блекджека своя логіка – він базується на вибірці без повернення, що робить його системою із залежними подіями. Колода стає коротшою з кожним розіграним раундом, тож шанси оновлюються постійно. Саме цей механізм дає простір для аналізу кожної роздачі.

У стандартній грі використовується колода з 52 карт. Очки нараховуються за значенням кожної карти. Карти з цифрами – від двох до десяти дають стільки ж балів, скільки показано. Карти з фігурами – Валет, Дама, Король коштують по десять за кожен. Туз інакше входить у рахунок: це один чи одинадцять, залежно від ситуації. Складність обчислень тримається саме на такому подвоєному значенні туза.

Оскільки події у блекджеку є залежними (вибірка без повернення), для розрахунку ймовірностей використовується теорема множення ймовірностей залежних подій (1.7):

$$P(A * B) = P(A) * P(B/A), \quad (1.7)$$

де $P(A)$ – ймовірність першої події;

$P(B/A)$ – ймовірність настання другої події за умови, що перша вже відбулася.

Наприклад, початкова ймовірність отримати суму у 21 очко з перших двох карт розраховується як сума ймовірностей двох сценаріїв: витягнути спочатку туза, а потім карту номіналом 10 очок, або навпаки. Застосовуючи формулу (1.7), отримуємо (1.8):

$$P(blackjack) = 2 * \left(\frac{4}{52} * \frac{16}{51} \right) = \frac{128}{2652} \approx 4.82\%, \quad (1.8)$$

Перевага закладу у блекджеку виникає з одного правила: коли гравець набирає більше 21, йому зараховують програш одразу, незалежно від того, що скільки далі набере дилер. З іншого боку, учасник сам обирає, чи брати ще карту, чи залишитися при своїй сумі. Тоді як круп'є не має жодного вибору, він просто слідує фіксованим правилам, де обов'язково добирає, якщо в сумі менше 17.

Щоб симуляція блекджека працювала правильно, потрібен чесний спосіб тасування колоди. Замість простого рандому – основа на надійних джерелах випадковості. Один із шляхів – це комбінувати криптографічні генератори з методом Фішера-Єтса. Цей підхід забезпечує однакову ймовірність для всіх $52!$ варіантів розташування карт. Така система ефективно блокує будь-які спроби передбачити наступну карту. Кожне тасування залишається непередбачуваним, бо математика стоїть на заваді прогнозам. Результатом є стан істинного випадкового розподілу, який неможливо обійти.

Отже, ці дві моделі створюють повноцінну основу для навчання – рулетка показує як працює теорія ймовірностей у незалежних подіях, тоді як блекджек знайомить із ситуаціями, коли шанси постійно міняються, а рішення доводиться узгоджувати з новими обставинами.

1.3 Огляд існуючих систем, що вирішують аналогічні задачі

1.3.1 Wojoko Blackjack Trainer

Це спеціалізований вебзастосунок та мобільний тренажер, головною метою якого є навчання гравців ідеальній стратегії. Додаток пропонує інтерактивний ігровий процес, де користувач отримує карти і повинен прийняти рішення. На будь-який математично не вигідний хід система дає одразу реакцію: гра виводить на екран підказку, пояснюючи, чому інше рішення було б статистично правильнішим. Переваги цього додатку:

- інтуїтивно зрозумілий, сучасний, націлений на мобільні пристрої дизайн;
- наявність режиму «Drill» (тренування), який дозволяє відпрацьовувати лише найскладніші комбінації карт, ігноруючи звичайні роздачі;
- візуальне пояснення щодо прийнятих рішень з наочною демонстрацією оптимального ходу.

Недоліки:

- відсутність клієнт-серверної архітектури для зберігання прогресу. Сесія користувача записується у локальне сховище браузера, тому при оновленні сторінки весь прогрес безповоротно втрачається;
- відсутність системи авторизації та персональних акаунтів користувачів;
- додаток вузьконаправлений і дозволяє тренуватися лише у блекджеку.

Робота програми продемонстрована на рисунку 1.1.



Рисунок 1.1 – Vojoko Blackjack Trainer [3]

1.3.2 CalcBE Probability Simulator

Це інструмент для розрахунку та візуалізації ймовірнісних моделей. Цей сервіс є більш універсальним і зосереджений на чистій математиці, статистиці та генерації випадкових подій. Замість класичного ігрового інтерфейсу, програма пропонує користувачу набір форм для введення початкових параметрів (ймовірність події, кількість ітерацій, розмір ставки). Також дозволяє будувати графіки результатів на великій дистанції. Переваги цього інструменту:

- дуже висока швидкість обчислень;
- формування детальних звітів з побудовою графіків;
- можливість налаштовувати власні математичні змінні та правила симуляції.

Недоліки:

- застарілий, перевантажений та неінтуїтивно зрозумілий інтерфейс;
- високий поріг входження – користувач повинен мати глибокі знання в теорії ймовірностей, щоб правильно зрозуміти отримані результати;

- відсутність візуальної ігрової складової.

Робота програми продемонстрована на рисунку 1.2.

Choose coin flips with custom $p(\text{Heads})$, weighted six-sided dice, or European/American roulette bets. The simulator compares theory and empirical ratios, shows Wilson 95% intervals for binary outcomes, tracks convergence, and records each calculation step.

For handouts, export raw counts as CSV or copy the LaTeX summary. Use the shareable URL to reopen the same scenario later.

Controls

General settings

Experiment: Coin Trials (N): 1000 Seed: 42

Batch / frame: 2000

Coin settings

$p(\text{Heads})$: 0,5

Run Pause Reset Export CSV Copy URL

Embed this calculator App Copy LaTeX

Progress: 100% (1,000/1,000)

Result

Trials processed	1,000
Seed	42
Theoretical probability	50.00% (0.5)
Empirical probability	48.70% (0.487)
Difference (empirical – theory)	-0.013
Wilson 95% CI	[45.613%, 51.797%]

How it's calculated

- 1 Initialise the linear congruential generator ($a = 1664525$, $c = 1013904223$, $m = 2^{32}$) with the chosen seed.
- 2 Record the theoretical probabilities $P(H) = p_H$ and $P(T) = 1 - p_H$ before drawing samples.
- 3 Wilson 95% interval uses $z = 1.96$ and the observed number of heads vs trials.

Рисунок 1.2 – CalcBE Probability Simulator [4]

1.3.3 Wojoko Roulette Trainer

Аналогічний за своєю освітньою філософією продукт від того ж розробника, але повністю сфокусований на симуляції гри в рулетку, як європейську, так і американську. Основна мета застосунку – надати користувачам можливість вивчати розподіл ймовірностей на ігровому полі та тестувати різні математичні системи управління балансом без фінансових ризиків. Переваги цього застосунку:

- вбудований довідник з детальним поясненням математичного очікування для різних варіацій рулетки;

- реалістична візуалізація ігрового столу та плавна анімація обертання колеса;
- відсутність нав'язливих рекламних інтеграцій та закликів до гри на реальні гроші.

Недоліки:

- повністю відсутня база даних для збереження користувацької історії;
- додаток вузьконаправлений і дозволяє тренуватися лише у рулетці.

Робота програми продемонстрована на рисунку 1.3.



Рисунок 1.3 – Bojoko Roulette Trainer [5]

1.3.4 Порівняння описаних систем

Більш детальне порівняння наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння програмних аналогів симулятора ігрових механік

Характеристика	Bojoko Blackjack Trainer	CalcBE Probability Simulator	Bojoko Roulette Trainer
Основна спрямованість	Вивчення стратегій карткової гри	Математичний розрахунок і моделювання ймовірностей	Тестування систем розподілу ставок
Гнучкість налаштувань	Низька (фіксовані правила однієї гри)	Висока (можливість задавати власні змінні)	Низька (фіксовані правила однієї гри)
Навчальний аспект	Підказки щодо математично правильних ходів	Аналіз статистичних відхилень	Пропонуються різні математичні стратегії для максимізації отримання виграшу
Візуалізація даних	Графічний інтерфейс	Текстові результати та математичні графіки	Графічний інтерфейс
Збереження прогресу та сесій	До оновлення вкладки	До оновлення вкладки	До оновлення вкладки

1.4 Постановка завдань роботи

Для досягнення мети розробки вебсимулятора ігрових механік необхідно вирішити наступний комплекс завдань:

- аналітичне дослідження предметної області – вивчення математичних правил блекджеку та рулетки, визначення основних параметрів ймовірності та алгоритмів генерації випадкових подій;
- архітектурне проєктування системи – розробка схеми взаємодії компонентів за моделлю клієнт-сервер, проєктування реляційної структури БД для надійного обліку віртуальних балів та статистики користувачів;
- реалізація серверної логіки – створення Node.js сервера для обробки ігрових правил, керування станами сесій та реалізація SMTP-сервісу для верифікації користувачів [6]-[7];
- розробка модульного та респонсивного клієнтського інтерфейсу – реалізація адаптивних React-компонентів для візуалізації обертання колеса рулетки та карткових розкладів, а також створення кабінету користувача зі статистикою [8];
- система підтримки ігрової активності – розробка механізму бонусного поповнення балансу через розв'язання Captcha, що дозволяє продовжувати вивчення механік при вичерпанні віртуального балансу [9];
- верифікація та оцінка працездатності – проведення функціонального тестування всіх ігрових сценаріїв та перевірка коректності синхронізації даних між декількома активними користувачами.

Окрім функціональних завдань, до розроблюваної системи висувається ряд нефункціональних вимог без яких система працюватиме нестабільно:

- продуктивність – швидкість реакції сервера на дію клієнта, яка має тривати менше 200 мілісекунд. Такий поріг допомагає зберегти ефект безперервного процесу гри;
- масштабованість – щоб система мала змогу розширюватися, архітектура бекенду повинна бути спроектована таким чином, щоб дозволяти горизонтальне масштабування у майбутньому;
- безпека – система повинна гарантувати захист від найпоширеніших вебвразливостей. Усі паролі користувачів мають зберігатися виключно у вигляді хешів, а сесії мають бути захищені за допомогою JWT;

- адаптивність – інтерфейс симулятора повинен коректно відображатися на пристроях із різним розміром екрану (від мобільних телефонів з роздільною здатністю 320px до широкоформатних моніторів);
- відмовостійкість – навіть якщо зв'язок раптом обірветься, то сервер має запам'ятати останній стан сесії гравця, щоб після повторного підключення користувача процес гри просто продовжувався ніби нічого й не сталося.

1.5 Висновки за розділом 1

У результаті виконання першого розділу було проведено всебічний аналіз предметної області та обґрунтовано необхідність розробки вебсимулятора.

Через аналіз математичних моделей ігрових механік блекджеку та рулетки виявили ключові закономірності ймовірностей, які мають бути закладені в основу програмної реалізації для забезпечення її достовірності

Аналіз існуючих систем описаний показав, що переважна більшість подібних ресурсів спеціалізується на чомусь одному, наприклад, окремій грі або формальній статистиці. Цей аналіз підтвердив доцільність розробки інтегрованої платформи, яка включає різні ігрові механіки разом із персоналізованим збором даних. На основі отриманих висновків визначено набір конкретних технічних параметрів, які потім буде використано при побудові архітектури системи.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Концептуальне проєктування та вибір методології розробки

Для розробки архітектури симулятора обрано об'єктно-орієнтований підхід (ООП) з використанням мови моделювання UML (Unified Modeling Language). Цей вибір обґрунтований можливістю декомпозиції системи на незалежні компоненти та класи, що значно полегшує подальшу програмну реалізацію на базі React [10]-[11].

2.2 Обґрунтування вибору інструментальних засобів

Для забезпечення високої продуктивності та надійності симулятора було проведено аналіз сучасних технологічних інструментальних засобів. Вибір інструментів базувався на критеріях швидкості оновлення, підтримки реального часу та стабільності роботи з даними.

2.2.1 Вибір Frontend-технології

У таблиці 2.1 наведено порівняння популярних засобів для розробки інтерфейсу.

Таблиця 2.1 – Порівняння популярних засобів для розробки інтерфейсу

Характеристика	React	Vue.js	Angular
1	2	3	4
Оновлення інтерфейсу	Швидко (Virtual DOM).	Швидко (Virtual DOM)	Повільніше (Real DOM)
Архітектура	Компонентна	Компонентна	MVC
Поріг входження	Середній	Середній	Високий

Продовження таблиці 2.1

1	2	3	4
Екосистема та бібліотеки	Велика	Середня	Велика, але переважно орієнтована на корпоративні рішення
Інтеграція сторонніх скриптів	Легка	Середня	Складна
Управління станом	Гнучке	Двостороннє зв'язування	Складне

З порівняльного аналізу помітно, що фреймворк Angular є занадто громіздким для розробки вебсимулятора, оскільки його архітектура орієнтована на великі корпоративні системи і його застосування збільшило б час розробки базового функціоналу. Тим часом Vue.js показує чудову швидкодію та легкість, проте має меншу екосистему готових рішень для візуалізації даних.

Найкращим кандидатом для реалізації клієнтської частини було обрано бібліотеку React. Її головна перевага для даного проєкту полягає у використанні механізму Virtual DOM, оскільки симулятор ігрових механік передбачає постійне отримання результатів обчислень від сервера та необхідність миттєвого оновлення статистики і графіки на екрані клієнта. Virtual DOM дозволяє замість повного оновлення сторінки перебудовувати лише фрагменти, що змінились. Це гарантує високу продуктивність та відсутність затримок в інтерфейсі користувача.

2.2.2 Вибір Backend-платформи

У таблиці 2.2 наведено порівняння середовищ, які здатні обробляти велику кількість одночасних з'єднань.

Таблиця 2.2 – Порівняльна характеристика Backend-платформ

Характеристика	Node.js	Python (Django)	PHP
Модель виконання	Асинхронна (Event Loop)	Синхронна	Синхронна
Робота з WebSocket	Нативна	Підтримується, але вимагає додаткових архітектурних надбудов	Потребує розгортання окремих зовнішніх сервісів
Мова розробки	JavaScript	Python	PHP
Швидкість обробки I/O (Input/Output) запитів	Дуже висока	Середня	Низька
Споживання ресурсів сервера	Низьке	Середнє	Високе
Серіалізація даних (JSON)	Нативна	Потребує додаткових кроків	Потребує виклику спеціальних функцій

Після проведення аналізу популярних бекенд-рішень – серед яких Python і PHP стало зрозуміло, що незважаючи на те, що ці технології мають широке застосування, синхронна обробка запитів у PHP не підходить для сервісів, які вимагають постійного з'єднання з користувачами. Python

пропонує гнучкі можливості для реалізації логіки на стороні сервера, тим не менше, розгортання WebSocket потребує надбудов, які ускладнюють систему [12].

Справа у тому, що його несинхронна модель на основі циклу подій дозволяє ефективно керувати численними операціями введення/виведення. Через те що основа системи – це математичний симулятор ігрових механік, потрібно швидко доставляти результати обчислень користувачу, уникаючи поновлення інтерфейсу. У такому контексті сполучення Node.js із технологією WebSocket через Socket.io дає можливість досягти мінімальної затримки та високу пропускну здатність мережі [13].

2.2.3 Вибір системи керування базами даних

У таблиці 2.3 наведено порівняння систем керування базами даних.

Таблиця 2.3 – Порівняльна характеристика систем керування базами даних

Характеристика	PostgreSQL	MySQL	MongoDB
Тип даних	Реляційна	Реляційна	NoSQL
Потенціал масштабування	Високий	Середній	Обмежений
Цілісність даних	Максимально сувора	Менш сувора	Відсутня
Транзакційність	Повна	Часткова	Обмежена
Надійність	Висока	Середня	Висока
Складні запити	Підтримуються	Обмежені	Обмежені

На етапі планування шару для зберігання інформації було проаналізовано класичні бази типу MySQL разом із гнучкими NoSQL варіантами наприклад, MongoDB. Хоча друга добре себе зарекомендувала в

середовищах на основі Node.js, повна відсутність обмежень на структуру даних робить її непридатною для цього випадку. Система покладається на точну передачу стану між операціями: баланс, сукупний розмір ставок і кількість перемог мають залишатися постійними завжди. Через це ключова увага приділяється суворій типізації та цілісності даних.

Як основну СКБД було обрано об'єктно-реляційну систему PostgreSQL. Серед усіх можливих рішень саме ця платформа виділяється надзвичайною строгістю правил для типів інформації та підтримкою принципів ACID (Атомарність, узгодженість, ізолюваність, Довговічність). Це гарантує, що будь-які математичні операції з балансом профілю будуть виконані безпечно [14]-[15].

Хоча поточна версія системи використовує одну таблицю, вибір PostgreSQL закладає потужний архітектурний фундамент для майбутнього масштабування. У разі необхідності розширення функціоналу, PostgreSQL дозволить легко перейти до складної реляційної структури.

2.2.4 Вибір середовища розробки

У таблиці 2.4 наведено порівняння сучасних середовищ для написання та налагоджування коду.

Для реалізації програмної частини симулятора було проведено аналіз сучасних середовищ розробки та редакторів коду. Хоча Sublime Text і демонструє високу швидкість роботи, проте йому бракує інструментарію для налагодження коду складних клієнт-серверних проєктів. Такі платформи, як WebStorm, навпаки, дають майже все одразу – але потребують чимало ресурсів ПК (персонального комп'ютеру). Вартість цих інструментів також виявилася важливою обставиною під час вибору.

Таблиця 2.4 – Порівняльна характеристика середовищ розробки

Характеристика	Visual Studio Code	WebStorm	Sublime Text
Вартість	Безкоштовна	Платна ліцензія	Умовно- безкоштовна
Екосистема та розширюваність	Максимальна	Висока	Обмежена
Інтеграція інструментів	Глибока та нативна	Потужна, але має перевантажений інтерфейс	Базова
Споживання системних ресурсів	Середнє	Високе	Низьке

Для роботи було обрано Visual Studio Code – головним критерієм стало те, що інструмент поєднує простоту текстового редактора з функціоналом середовища для розробки. Хоча спочатку виглядає як базова програма, завдяки додаткам стає потужною платформою. Такий підхід дає можливість точно налаштувати процес, не навантажуючи систему зайвим.

2.3 Висновки за розділом 2

На етапі другого розділу було сформовано основу для архітектури вебсимулятора. Було обґрунтовано використання об'єктно-орієнтованого підходу, що дозволяє чітко розділити ігрову логіку, інтерфейс користувача та роботу з базою даних.

Вибір технологічного стеку – React для клієнтської частини, Node.js для серверної та PostgreSQL для збереження даних – повністю задовольняє функціональні вимоги до продуктивності, стабільності та забезпечення багатокористувацької взаємодії у реальному часі.

3 РОЗРОБКА ПРОГРАМИ

3.1 Загальні відомості та функціональне призначення програми

Симулятор ігрових механік являє собою інтерактивний клієнт-серверний додаток. Головне функціональне призначення програми полягає у симуляції класичних ігрових процесів для освітніх та дослідницьких цілей без використання реальних фінансових активів. Розробка симулятора базується на комплексному поєднанні декількох видів забезпечення:

- математичне забезпечення – включає алгоритми генерації псевдовипадкових чисел для симуляції результатів ігор;
- лінгвістичне забезпечення – основними мовами розробки є JavaScript для фронтенду та бекенду. Передачу інформації організовано за допомогою JSON, оскільки саме цей формат став стандартним у сучасних вебзастосунках;
- інформаційне забезпечення – реалізовано через реляційну модель бази даних PostgreSQL та систему кешування станів ігрових сесій у пам'яті сервера для мінімізації затримок;
- програмне забезпечення – базується на середовищі Node.js. Серверну частину створено з використанням Express. Реалізація інтерфейсу – завдяки React [16].

3.2 Вимоги до складу технічних та програмних засобів

Для коректного функціонування симулятора встановлено наступні вимоги:

- програмні засоби – веббраузер з підтримкою HTML5 та WebSocket (Chrome, Firefox, Opera, Safari останніх версій); серверне середовище Node.js v18 або вище;
- технічні засоби (сервер) – процесор від 2 ГГц, оперативна пам'ять від 4 ГБ, 1 ГБ вільного місця на диску;

– технічні засоби (клієнт) – пристрій з доступом до мережі Інтернет та мінімальною роздільною здатністю екрану 320x480 пікселів.

3.3 Моделювання динамічних та статичних аспектів системи

3.3.1 Діаграма прецедентів

Для формалізації вимог побудовано діаграму прецедентів (Use Case Diagram), яка відображає основні сценарії взаємодії користувача з системою: реєстрація, авторизація, участь у симуляціях ігор, відновлення пароля, видалення та виходу з акаунту та отримання бонусів через проходження Captcha (рис. 3.1).

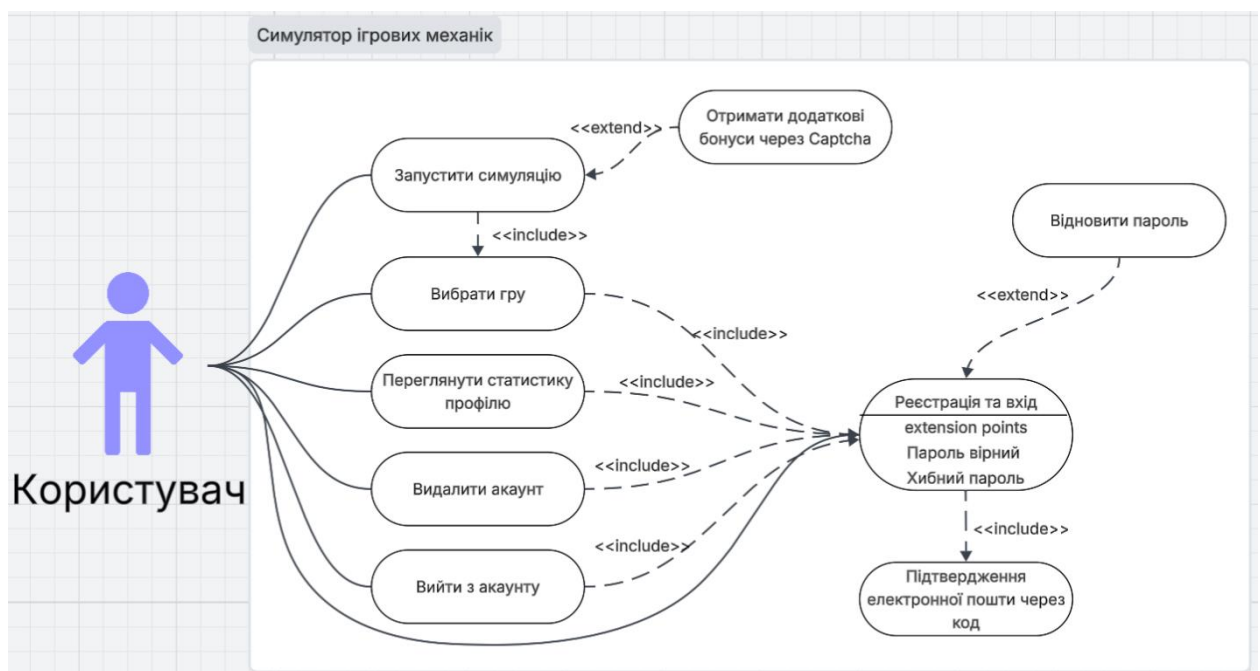


Рисунок 3.1 – Діаграма прецедентів

3.3.2 Діаграма компонентів

Статична структура відображена через діаграму компонентів, що ілюструє поділ системи на Frontend-модуль, Backend та базу даних (рис. 3.2).

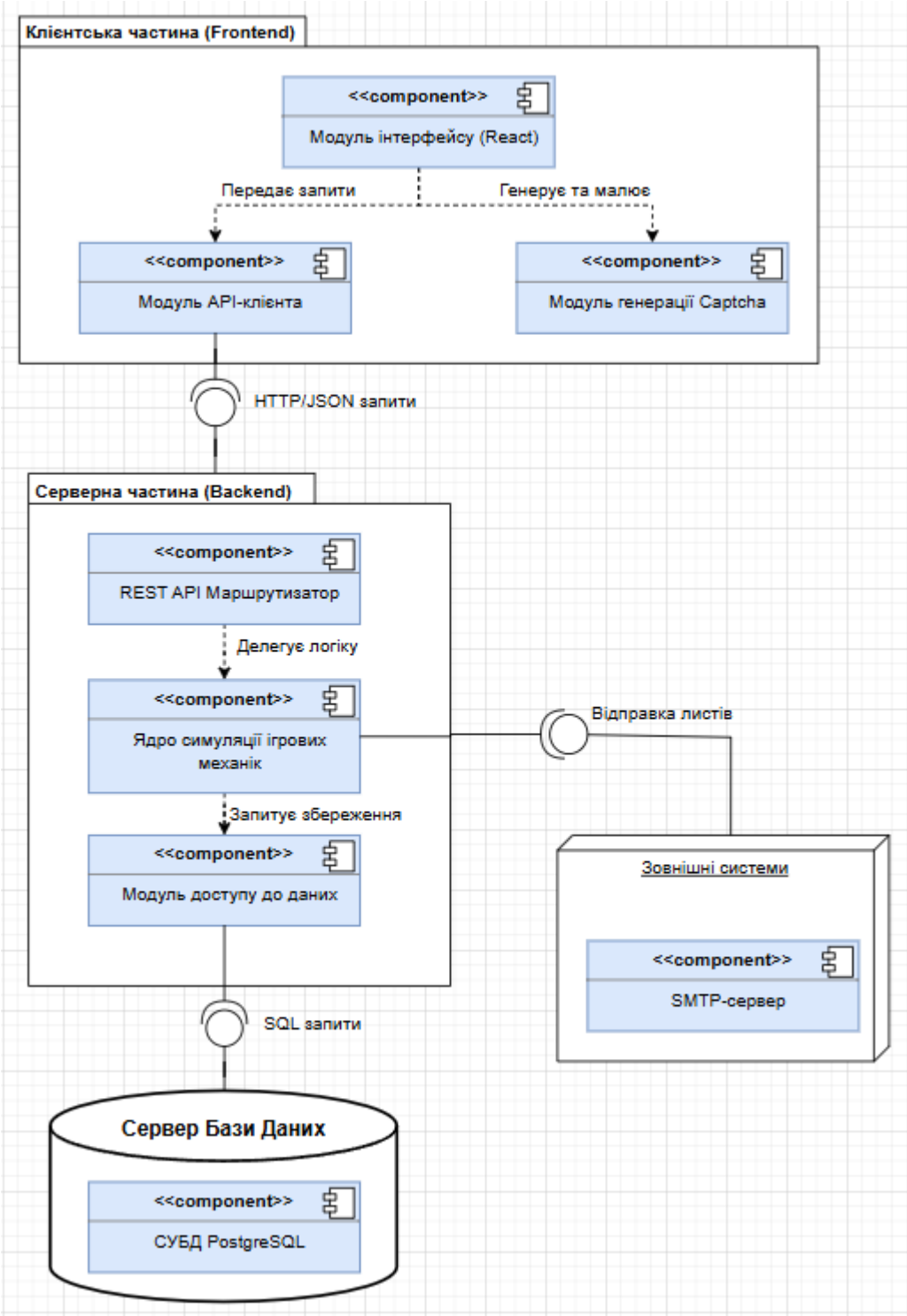


Рисунок 3.2 – Діаграма компонентів

3.3.3 Діаграма послідовності

Динаміка взаємодії у реальному часі представлена діаграмою послідовності, яка демонструє життєвий цикл ігрового раунду: від ставки користувача до отримання результату (рис. 3.3).

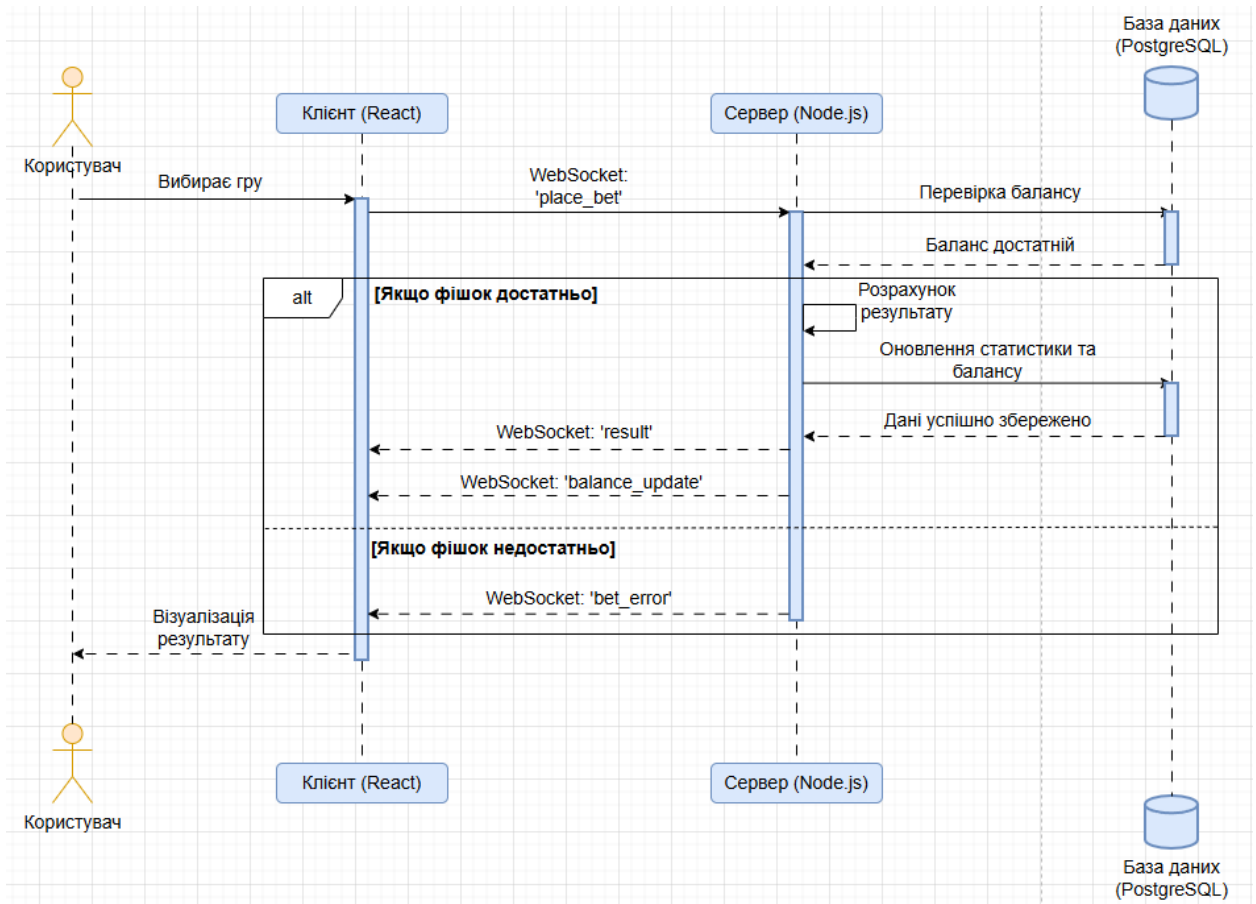


Рисунок 3.3 – Діаграма послідовності

3.4 Проєктування структури даних

Оскільки в архітектуру системи закладено підхід агрегації стану без логування кожної окремої дії, база даних складається з однієї комплексної таблиці. Вона включає тільки сутність «User», яка зберігає облікові дані та статистичні показники. Реляційний підхід обрано через необхідність суворої валідації балансу користувача та запобігання неправильним даним при одночасній обробці запитів (рис. 3.4).

User	
[PK] int	<u>id</u>
varchar(50)	username
varchar(100)	email
varchar(255)	password_hash
int	balance
int	total_bets
int	bets_won
int	total_wagered

Рисунок 3.4 – Сутність «User»

3.5 Опис логічної структури програми

Серверна частина симулятора (модуль бекенду) виконує роль головного контролера. Відповідно до логічної структури програми, вона обробляє ігрову логіку та гарантує неможливість маніпуляції результатами з боку клієнта.

Ключові методи та алгоритмічні рішення серверної реалізації:

- «startRouletteLoop()» – функція, що відповідає за безперервний життєвий цикл раунду рулетки. Вона керує таймерами для фази ставок (15 секунд) та фази обертання колеса (10 секунд). Після завершення обертання алгоритм ітерує всі зроблені ставки, розраховуючи виграші за фіксованими коефіцієнтами;

- «createDeck()» – метод генерації початкового масиву карт для механіки гри «Блекджек». Після створення 52 карт алгоритм застосовує метод перемішування, аналогічний алгоритму Фішера-Єтса, для досягнення криптографічної випадковості роздачі [17];

- «calculateHandValue(hand)» – математична функція для розрахунку суми очок гравця або дилера з динамічною зміною вартості туза (1 або 11). Якщо початкова сума перевищує 21 очко, а в руці є туз, його вартість автоматично зменшується до 1;

- «resolveBlackjackGame(socket, game, user)» – асинхронна процедура, що визначає переможця після завершення набору карт. Вона реалізує правило для дилера – брати карти, доки сума очок менша за 17. Відповідно до результату, процедура оновлює статистичні поля «balance», «bets_won» та «total_wagered» у базі даних.

Також сервер постійно відслідковує події Socket.io, такі як «place_bet», «bj_hit», «bj_stand», гарантуючи миттєву реакцію на дії гравця та блокуючи некоректні або повторні запити.

3.6 Початкові та вихідні дані

Внутрішня логіка програми базується на чітко визначених структурах даних. Основні змінні та колекції, що використовуються в системі, наведено у таблиці 3.1.

Для зв'язку з реляційною базою даних PostgreSQL налаштовано пул з'єднань (об'єкт «pool»). Усі транзакції, які пов'язані з балансом користувача виконуються виключно на стороні сервера. Це запобігає стану гонки та гарантує цілісність даних.

Таблиця 3.1 – Опис основних внутрішніх структур даних

Назва змінної	Тип даних	Призначення
pendingRegistrations	Map	Тимчасове сховище у пам'яті сервера для даних користувачів до верифікації їхнього email кодом
pendingResets	Map	Тимчасове сховище для валідації безпечних токенів під час процесу скидання пароля
rouletteState	Object	Об'єкт стану рулетки (зберігає поточний статус, масив виграшних чисел, історію результатів та таймер)
blackjackGames	Map	Хеш-таблиця, що зберігає ізольовані активні сесії гри у Blackjack для кожного підключеного клієнта
betAmount	Integer	Розмір внеску гравця у поточному раунді, який валідується проти його реального балансу
deck	Array	Масив об'єктів, що представляє гральну колоду
socket.user	Object	Розширений об'єкт WebSocket з'єднання, що містить ідентифікатор активної сесії
transporter	Object	Об'єкт конфігурації Nodemailer для асинхронної розсилки email-повідомлень

У вигляді вхідних даних система отримує від клієнта JSON-об'єкти через виклики «socket.emit». Перед обробкою будь-якої транзакції на сервері проводиться валідація типів даних та перевірка поточного балансу гравця, що унеможливорює відправку мінусових сум або підробку балансу. Сервер у свою

чергу, мінімізує обсяг вихідних даних, надсилаючи лише змінені стани системи за допомогою подій сокета («roulette_state», «bj_get_state» або «bet_result»). Формат серіалізації всіх даних – UTF-8.

3.7 Програмна реалізація клієнтського інтерфейсу на базі React

Інтерфейс користувача побудований за принципом компонентної декомпозиції. Кожен ігровий модуль живе самостійно, бо зроблена як незалежний React-елемент, де стиль і код не втручаються в інші частини. Завдяки цьому все працює швидко: коли щось міняється, оновлюється тільки потрібне, а решта лишається на своїх місцях. Це можливо завдяки тому, що React спочатку перевіряє розбіжності через Virtual DOM. Сторінка не перезавантажується кожен раз, адже браузер отримує чітку інструкцію що має зникнути, а що з'явитися.

Особливості реалізації ключових компонентів:

- компонент «RouletteWheel» – забезпечує плавну візуалізацію обертання колеса. Використовує динамічні CSS-трансформації для точної зупинки на секторі. Розрахунок кута виконується на основі пропса «resultNumber». Для запобігання зайвим перерендерам компонента застосовуються оптимізації React;

- компонент «Blackjack» – логіка столу відокремлена від логіки мережі. Компонент використовує хук «useEffect» для прослуховування WebSocket-подій («blackjack_state») та хук «useState» для зберігання локального стейту карт. При монтуванні компонента обов'язково викликається подія «bj_get_state», який дозволяє відновити гру у разі оновлення сторінки браузера;

- візуалізація карт «PlayingCard» – відображення окремих карт винесено в окремий дочірній компонент, що підтримує анімацію плавної появи карт за допомогою CSS-переходів;

- керування навігацією та профілем – компоненти «Header», «Footer» та «Lobby» відповідають за загальну навігацію додатком. Компонент «Profile» відправляє подію «socket.emit('get_profile')» для отримання з сервера детальної персональної статистики користувача (загальна кількість зроблених ставок, відсоток успіху, баланс тощо);

- керування авторизацією – модуль «Auth» та його дочірні компоненти «Login» і «ResetPassword» керують формами реєстрації, входу та відновлення доступу, здійснюючи попередню валідацію введених даних на стороні клієнта за допомогою регулярних виразів.

Для забезпечення стабільної роботи без витоків пам'яті, кожен ігровий компонент суворо керує своїм життєвим циклом через хук «useEffect». Цей процес складається з трьох ключових фаз:

- монтування – при відкритті сторінки гри, компонент ініціалізує слухачів подій WebSocket та відправляє запит на сервер для отримання початкового стану;

- оновлення – при отриманні нових даних від сервера, оновлюється локальний стан («useState»). Механізм Virtual DOM обчислює різницю між старим та новим станом, і точково перемальовує лише новий елемент інтерфейсу;

- розмонтування – при переході користувача на іншу сторінку симулятора, ігровий компонент знищується з DOM-дерева. У цей момент React обов'язково викликає функцію очищення всередині «useEffect», яка видаляє слухачів сокета. Це дуже важливий етап, який запобігає витокам пам'яті, дублюванню подій та виконанню коду у фоновому режимі.

Під час створення клієнтської частини особливу увагу було приділено тому, щоб симулятор гарно виглядав навіть на малих екранах. Для досягнення повної респонсивності застосовано сучасні технології CSS Flexbox та CSS Grid Layout, які дають можливість автоматично підлаштовуватися під будь-який формат дисплея. Коли ширина вікна зменшується, інтерфейс не ламається – елементи просто переупорядковуються. Головні компоненти, такі як стіл для

блекджеку чи колесо рулетки, масштабуються за допомогою відносних одиниць виміру та каскадних медіа-запитів. Зокрема, при зменшенні ширини екрану, елементи управління автоматично перегруповуються у вертикальну колонку, розміри гральних карт пропорційно зменшуються, а панель історії результатів рулетки трансформується у зручний горизонтальний скролінг.

3.8 Система верифікації, безпеки та допоміжні сервіси

Реалізація безпеки в проєкті є багаторівневою і включає декілька інструментів захисту:

- авторизація та Bcrypt – функція «generateToken(user)» створює безпечний JWT-токен після успішної перевірки пароля. JWT – це JSON Web Token, який замість сесій у пам'яті сервер аналізує його цифровий підпис, через що навантаження на БД спадає. Зберігається такий токен не на сервері, а в локальному сховищі браузера клієнта. Паролі користувачів ніколи не зберігаються у відкритому вигляді, бо алгоритм «bcrypt.hash» перетворює їх на хеші з використанням унікальної криптографічної «солі». Ця «сіль» робить кожен хеш унікальним, що є надійним захистом від зломів на основі райдужних таблиць. Під час авторизації паролі порівнюються безпечним методом «bcrypt.compare»;

- SMTP-сервіс (Nodemailer) – у серверному модулі налаштовано об'єкт «transporter» для інтеграції з поштовим провайдером. Він використовується для двофакторної аутентифікації на етапі реєстрації (відправка 6-значних кодів) та для генерації безпечних токенів («crypto.randomBytes») для створення одноразових унікальних посилань відновлення пароля;

- Captcha-сервіс – спеціальний інтерактивний компонент «CaptchaPopUp» відображається автоматично, коли баланс користувача досягає дуже малих значень (менше або дорівнює 100 фішок). Клієнт запитує нове графічне завдання через виклик «socket.emit('request_captcha')». Після

успішного розв'язання задачі користувачем викликається подія «solve_captcha», яка валідується сервером для захисту від ботів, після чого відбувається нарахування 100 бонусних фішок за одну пройдену капчу. Це дозволяє користувачам продовжувати тестування симулятора без постійного створення нових акаунтів.

3.9 Використані технічні засоби, виклик і завантаження програми

Розроблена програма призначена для роботи на сучасних ПК або серверах. Для коректного функціонування клієнтської частини необхідний пристрій з мінімальною роздільною здатністю екрану 320x480 пікселів та встановленим сучасним браузером. Серверна частина вимагає наявності процесора архітектури x86-64 або ARM та щонайменше 4 ГБ оперативної пам'яті.

Програма завантажується в оперативну пам'ять сервера за допомогою команди запуску середовища виконання Node.js. Користувач викликає клієнтську частину програми шляхом введення відповідної URL-адреси вебсервера у адресний рядок браузера, після чого відбувається завантаження необхідних HTML/CSS/JS файлів.

3.10 Висновки за розділом 3

У третьому розділі було детально обґрунтовано програмну реалізацію симулятора на всіх рівнях. Описано основні структури даних, формати оптимізованого обміну повідомленнями через WebSocket та функції обробки математичних правил ігор. Впроваджена модульна архітектура на базі React-компонентів, з використанням сучасних хуків керування станом, у поєднанні зі строгим подієво-орієнтованим сервером Node.js, дозволяє забезпечити миттєвий відгук інтерфейсу, надійний облік транзакцій та глибокий рівень безпеки даних користувача.

Застосування UML-діаграм – прецедентів, компонентних та послідовностей дало змогу чітко описати процеси всередині системи, починаючи з реєстрації й закінчуючи реакцією на ігрові події через WebSocket. Побудована ER-діаграма бази даних PostgreSQL була спроектована так, щоб безпечно фіксувати дані про статистику й стан рахунка гравця.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Керівництво програміста

Розроблений симулятор ігрових механік призначений для демонстрації роботи ймовірнісних алгоритмів, архітектури клієнт-серверних додатків реального часу та принципів безпечного управління станом користувача. Основною областю застосування є освітній процес – використання системи підходить для живої демонстрації основ теорії ймовірностей чи аналізу статистичних моделей. Хто бажає, перевіряє свої стратегії без потреби вкладати кошти – система дає змогу експериментувати.

Для розгортання серверної частини необхідний фізичний або віртуальний сервер з такими характеристиками:

- процесор – 2 ядра, від 2 ГГц;
- оперативна пам'ять – від 4 ГБ;
- СУБД – PostgreSQL версії 18 або вище;
- середовище виконання – Node.js (v18+).

Для клієнтської частини достатньо будь-якого сучасного веббраузера з підтримкою технології WebSockets та стандарту ES6+.

4.2 Керівництво системного програміста

Для ініціалізації системи необхідно виконати наступні кроки:

- клонувати репозиторій та встановити залежності за допомогою команди у терміналі «npm install»;
- налаштувати змінні у файлі «.env»: параметри підключення до БД («DB_USER», «DB_PASSWORD», «DB_NAME»), секретний ключ для JWT («JWT_SECRET») та конфігурацію SMTP-сервера («SMTP_EMAIL», «SMTP_PASSWORD»);

- запустити серверний процес командою «npm run dev» у директорії серверу. Під час першого запуску сервер автоматично згенерує необхідні таблиці у базі даних (ініціалізація «db.js»);
- запустити клієнтський додаток командою «npm run dev» у кореневій папці.

4.3 Керівництво оператора

4.3.1 Виконання програми

Користувач взаємодіє з системою через графічний інтерфейс. Робота розпочинається з вікна вітання, де користувач повинен ввести свої облікові дані. Для нових користувачів передбачена форма реєстрації, яка вимагає обов'язкового підтвердження електронної пошти через 6-значний код для додаткового захисту – без нього реєстрація не завершується. Процес реєстрації та авторизації зображені на рис. 4.1, рис. 4.2, рис. 4.3 та рис. 4.4.

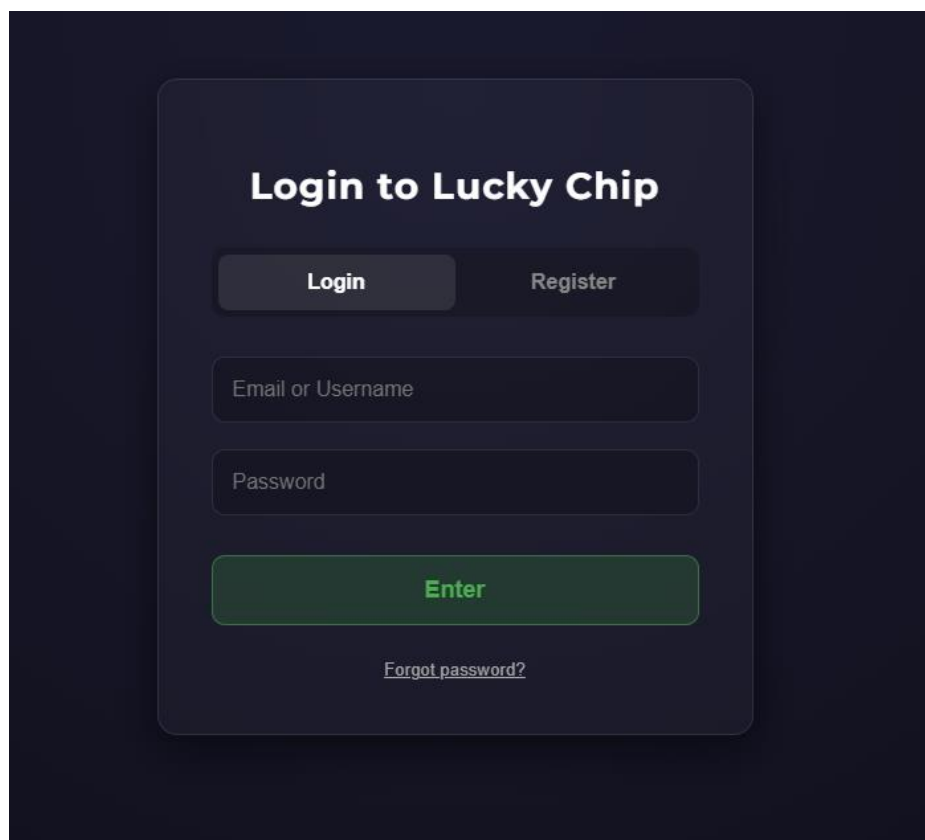
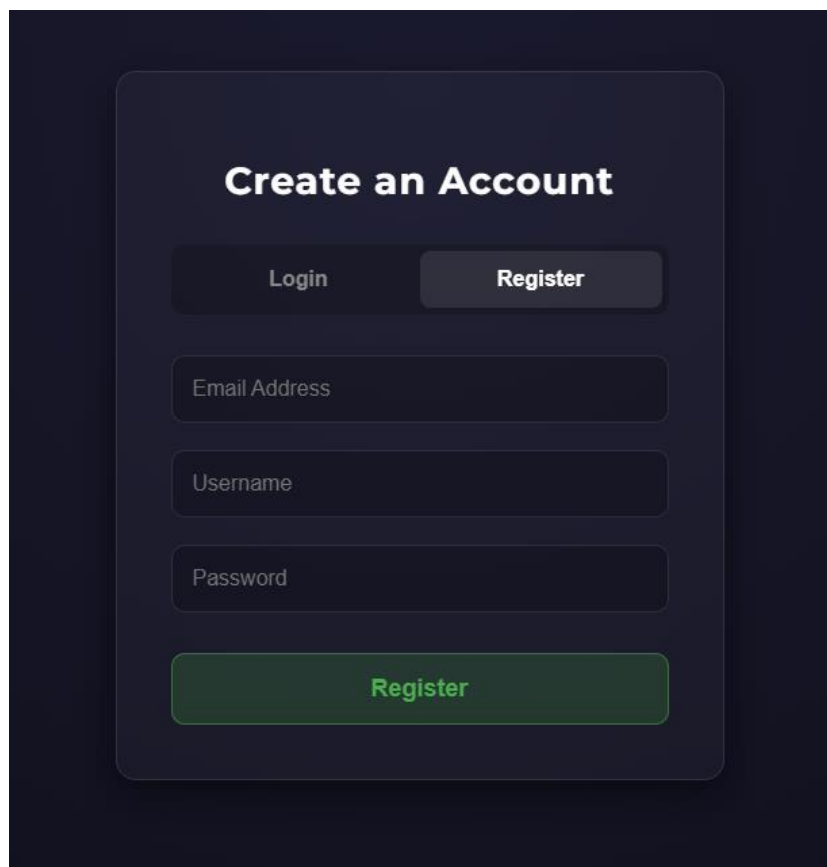
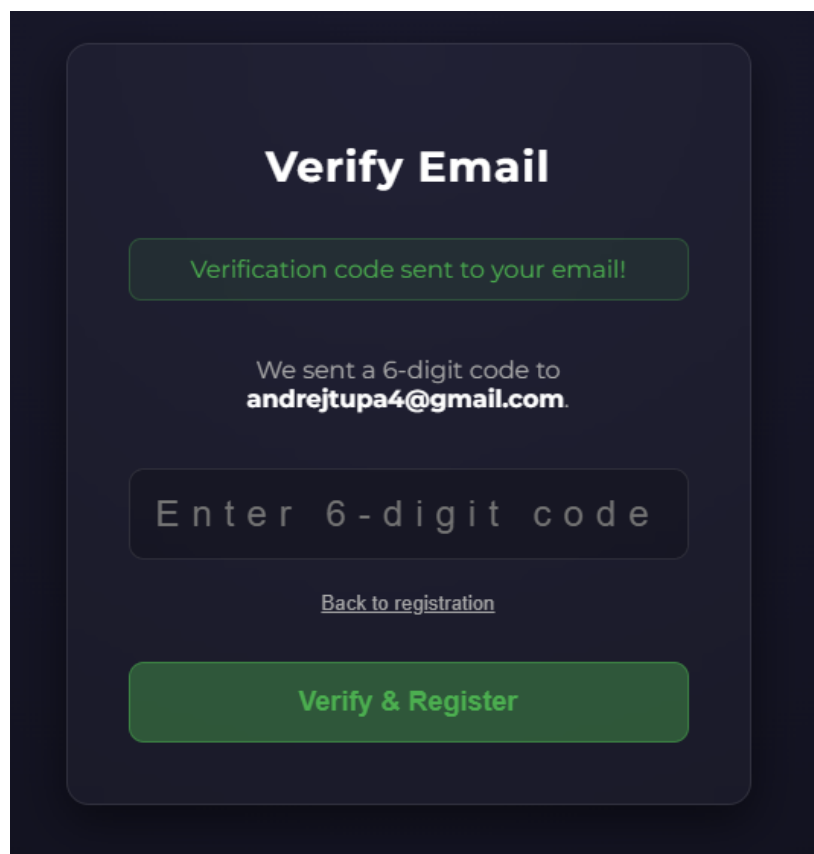


Рисунок 4.1 – Вікно авторизації



The image shows a registration form titled "Create an Account" on a dark background. At the top, there are two buttons: "Login" and "Register". Below these are three input fields labeled "Email Address", "Username", and "Password". At the bottom of the form is a large green button labeled "Register".

Рисунок 4.2 – Вікно реєстрації



The image shows a verification screen titled "Verify Email". At the top, a green message box says "Verification code sent to your email!". Below this, it says "We sent a 6-digit code to **andrejtupa4@gmail.com**." There is a large input field with the placeholder text "Enter 6-digit code". Below the input field is a link that says "Back to registration". At the bottom is a large green button labeled "Verify & Register".

Рисунок 4.3 – Вікно підтвердження пошти через код

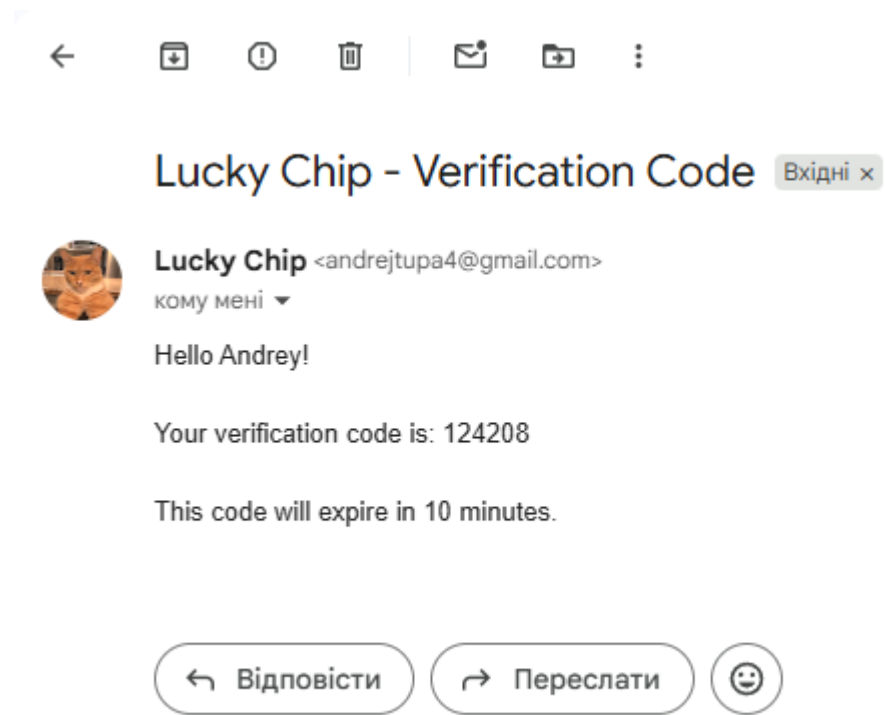


Рисунок 4.4 – Код підтвердження на електроній пошті

Після успішної авторизації оператор потрапляє у головне меню, де відображається його поточний віртуальний баланс та доступні для запуску ігри (рис. 4.5).

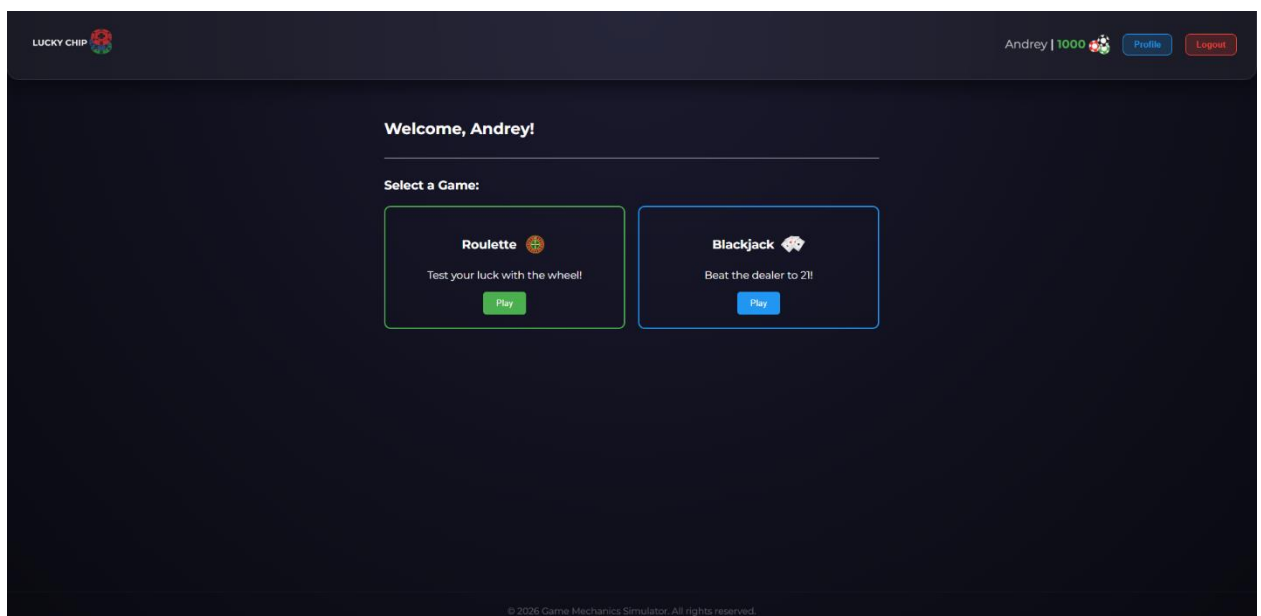


Рисунок 4.5 – Головне лоббі симулятора

При виборі рулетки відкривається ігровий стіл. Оператор взаємодіє з системою шляхом вибору фішок певного номіналу (але не менше 10) та натискання на відповідні сектори ігрового поля. У правій частині екрану (або знизу на мобільних пристроях) відображається панель історії результатів. Процес взаємодії з рулеткою зображений на рис. 4.6 та рис. 4.7.



Рисунок 4.6 – Інтерфейс рулетки



Рисунок 4.7 – Інтерфейс рулетки змінюється в залежності від дій оператора

У модулі «Блекджек» оператор керує процесом роздачі за допомогою кнопок дій («Hit» – взяти карту, «Stand» – зупинити набір карт). Інтерфейс динамічно підраховує та відображає поточну суму очок гравця та дилера. Процес взаємодії з модулем «Блекджек» зображений на рис. 4.8, рис. 4.9, рис. 4.10, рис. 4.11, рис. 4.12, рис. 4.13 та рис. 4.14.

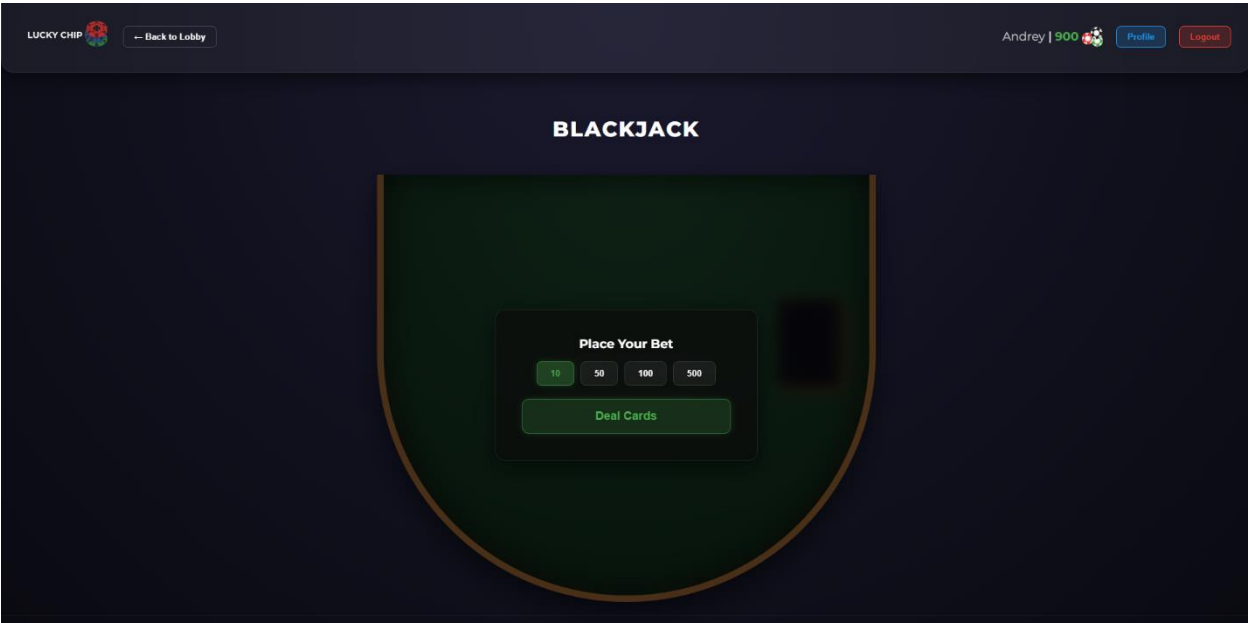


Рисунок 4.8 – Початкове вікно з вибором ставки

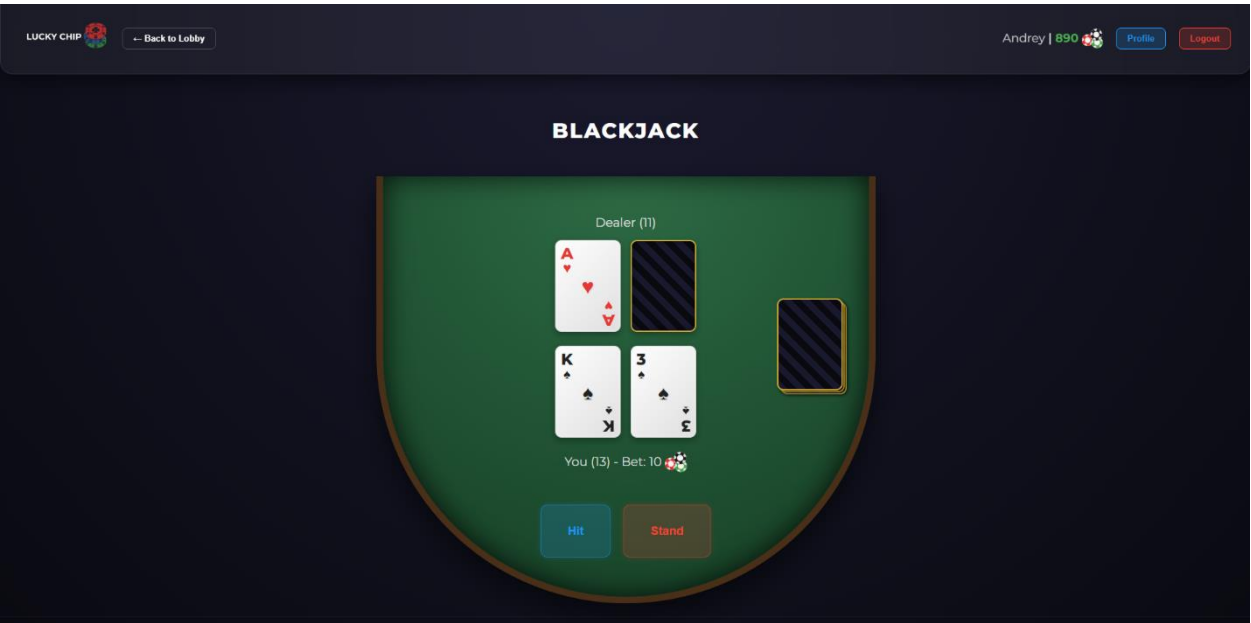


Рисунок 4.9 – Інтерфейс ігрового процесу гри

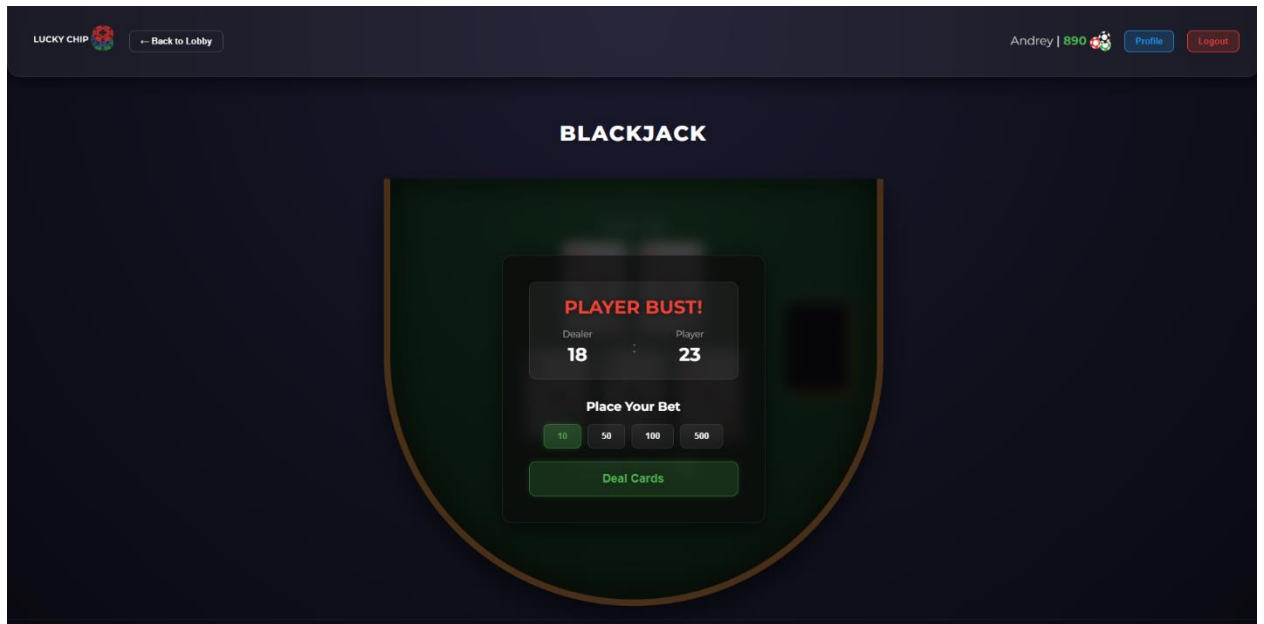


Рисунок 4.10 – Повідомлення про перебор карт гравця

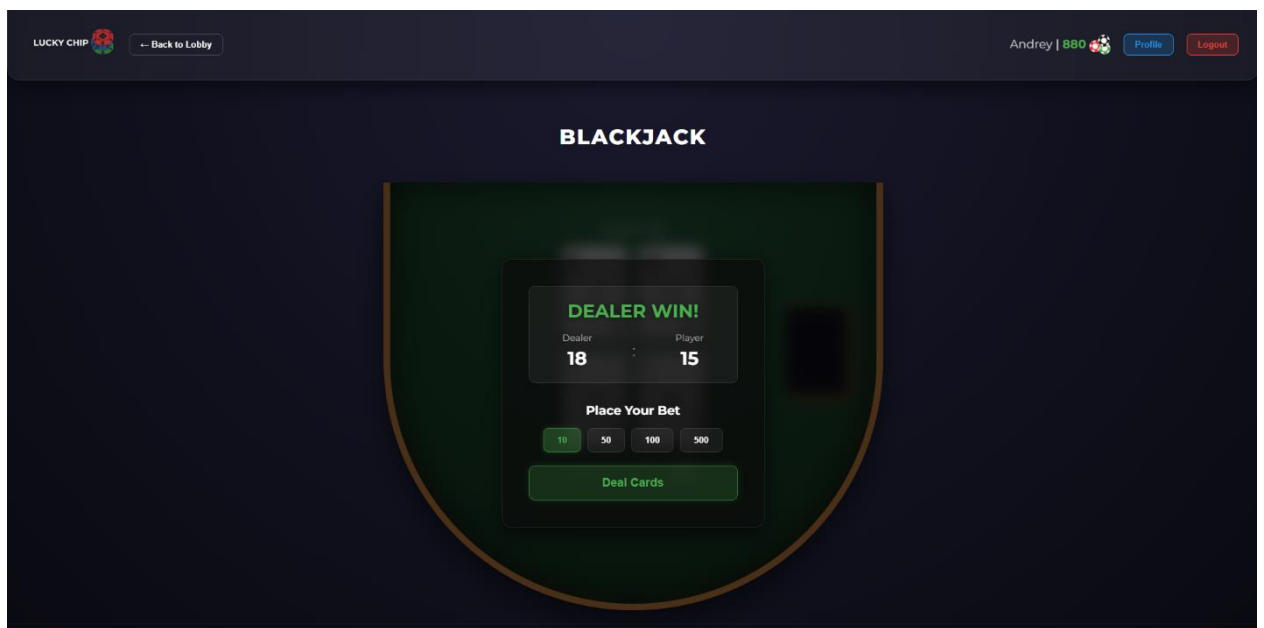


Рисунок 4.11 – Повідомлення про перемогу дилера

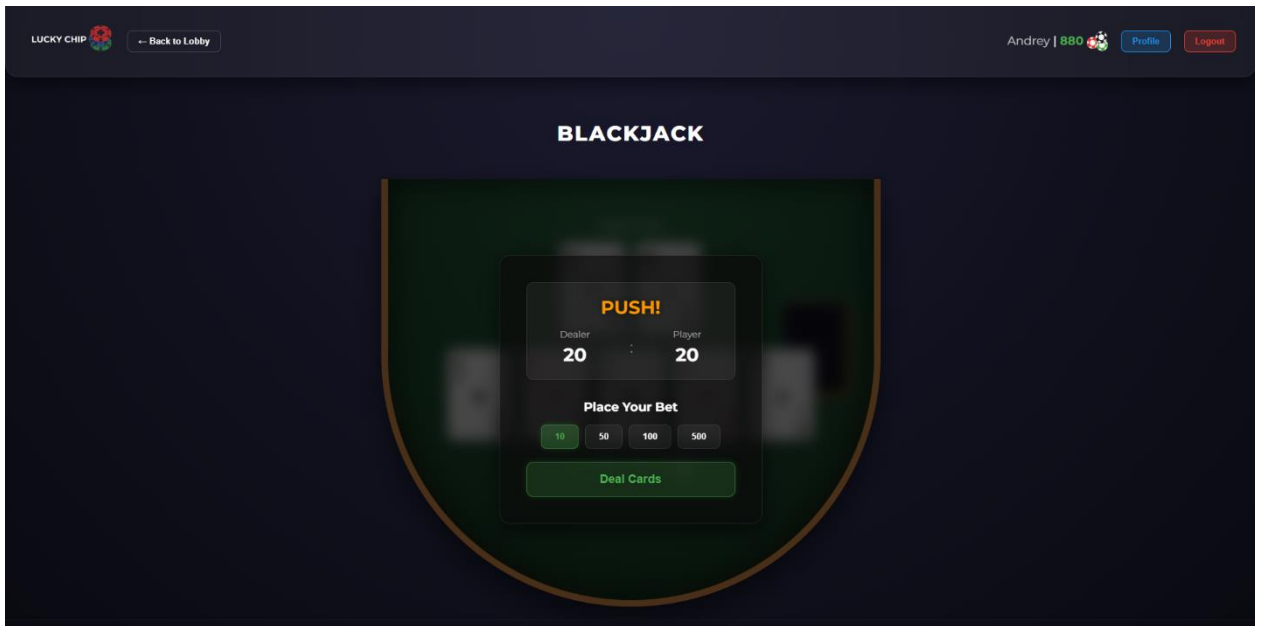


Рисунок 4.12 – Повідомлення про ничію

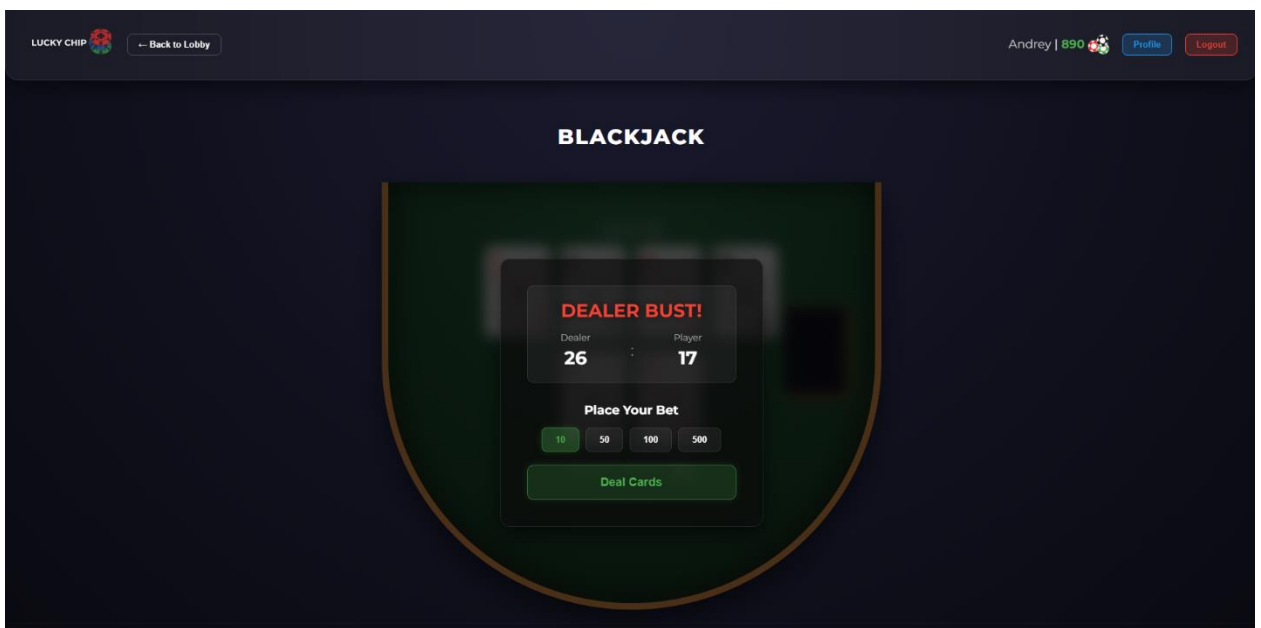


Рисунок 4.13 – Повідомлення про перебор дилера

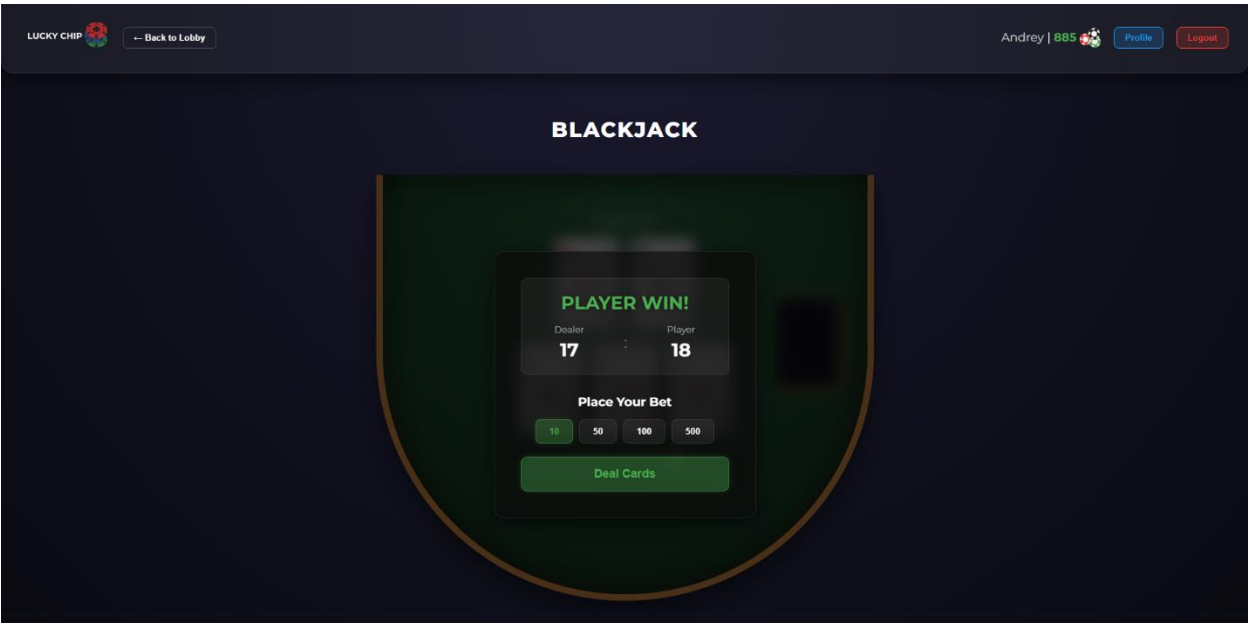


Рисунок 4.14 – Повідомлення про перемогу гравця

У випадку падіння балансу до критичного рівня (менше або дорівнює 100 фішок), система автоматично викликає вікно Captcha. Після правильного її вирішення баланс оператора відновлюється на 100 фішок за кожне правильне вирішення задачі. Цю механіку зображено на рис. 4.15, рис. 4.16 та рис. 4.17.



Рисунок 4.15 – Поп-ап вікно Captcha



Рисунок 4.16 – Спливаюче вікно Captcha



Рисунок 4.17 – Правильне вирішення Captcha

Оператор має можливість переглянути свою розгорнуту статистику, натиснувши на кнопку профілю. У цьому вікні відображаються дані про загальну кількість зроблених ставок, загальний обсяг відіграних фішок та процент виграшу (рис. 4.18).

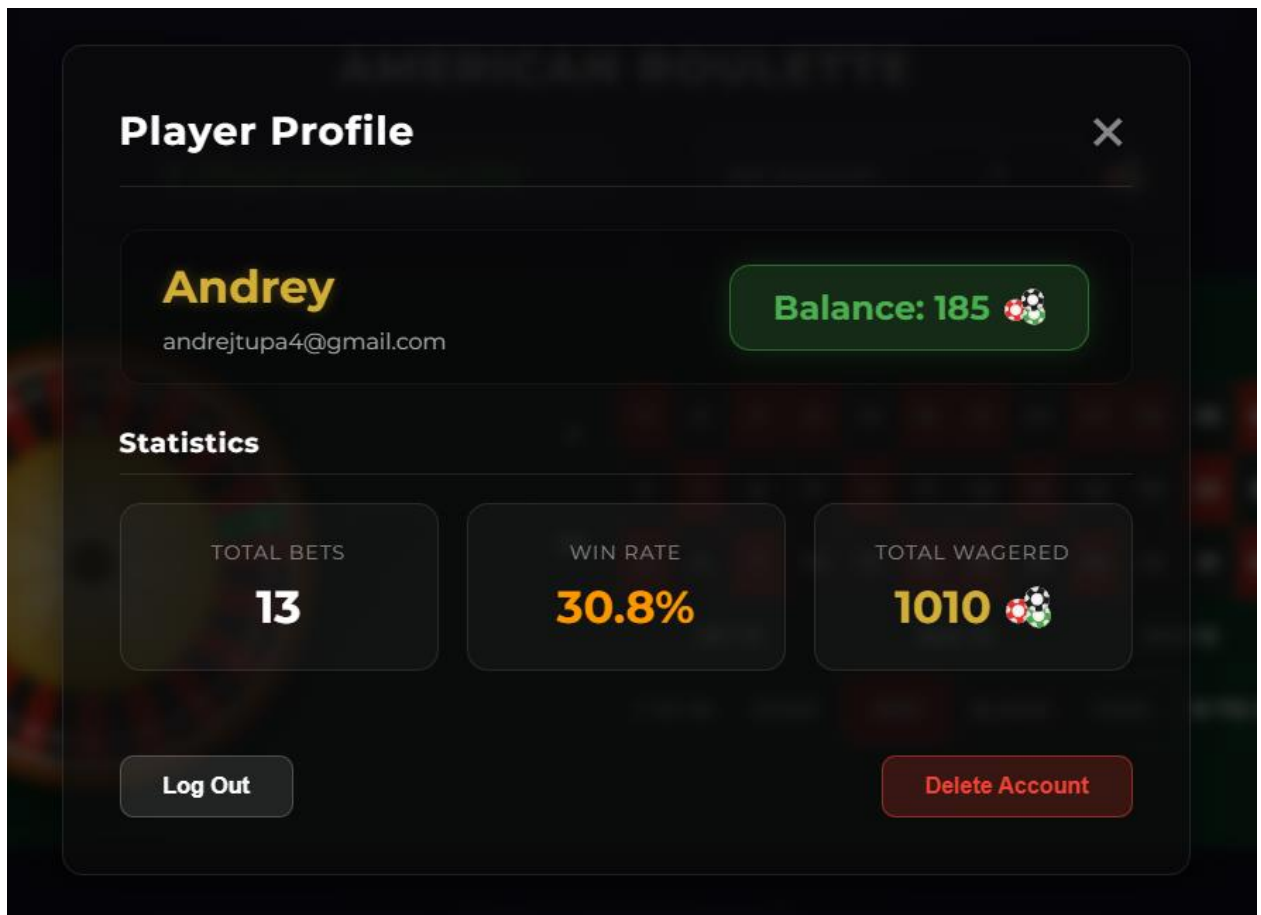
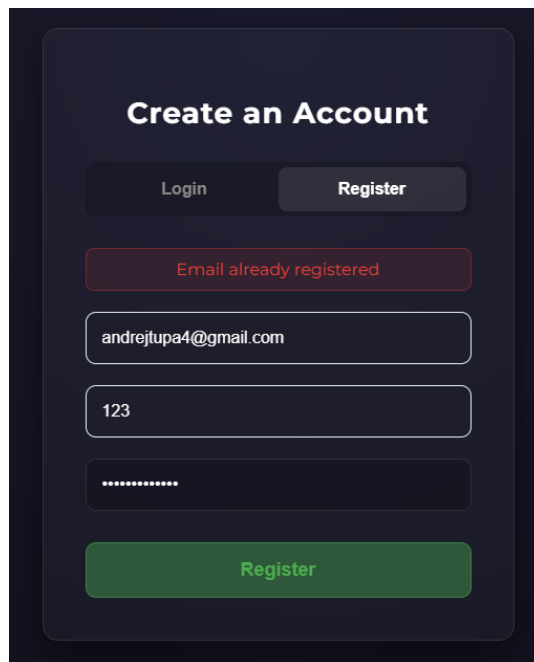


Рисунок 4.18 – Інтерфейс особистого кабінету користувача

4.3.2 Повідомлення оператору

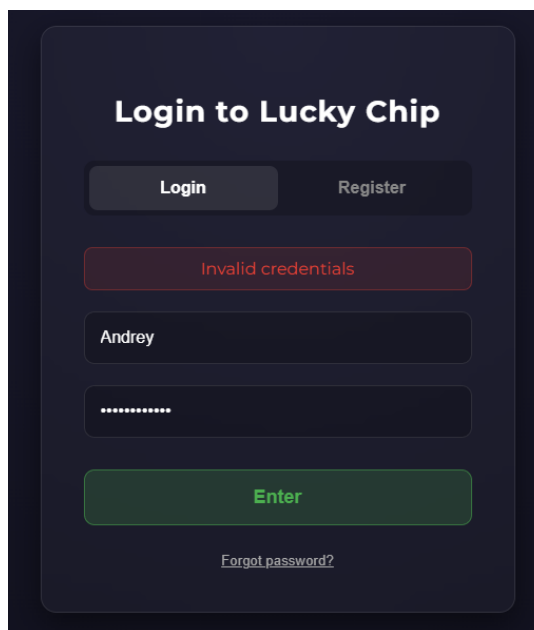
У ході виконання програми система може видавати діалогові повідомлення, що потребують відповідних дій. Під час авторизації та реєстрації можуть виникати помилки валідації: повідомлення «Користувач з таким email вже існує» означає, що краще спробувати увійти або скинути пароль, зображено на рис. 4.19, а повідомлення «Невірні реквізити для входу» свідчить про помилку автентифікації і вимагає перевірки введених даних, зображено на рис. 4.20. Ще один момент: при заданні нового паролю активується контроль його складності, коли пароль замалий за рівнем безпеки, з'являється попередження, яке зображено на рис. 4.21. Безпосередньо під час ігрового процесу оператор може зіткнутися з повідомленням «Недостатньо коштів для ставки», якщо обрана сума більша за поточний баланс, або з попередженням «Мінімальна ставка 10 фішок» у рулетці, що вимагає

збільшення розміру ставки, зображено на рис. 4.22. Крім того, коли баланс наближається до нуля, то автоматично з'являється вікно Captcha. Якщо користувач введе некоректні дані, то система виводить повідомлення «Невірний код», генерує нове зображення та вимагає повторної спроби, повідомлення зображено на рис. 4.23.



The screenshot shows a dark-themed user interface for account creation. At the top, the title 'Create an Account' is displayed in white. Below the title are two buttons: 'Login' and 'Register'. A red error message 'Email already registered' is shown in a rounded rectangle. Below the error message are three input fields: the first contains the email 'andrejtupa4@gmail.com', the second contains the number '123', and the third is a password field with masked characters. At the bottom is a large green button labeled 'Register'.

Рисунок 4.19 – Сповіщення про помилку реєстрації



The screenshot shows a dark-themed user interface for login. At the top, the title 'Login to Lucky Chip' is displayed in white. Below the title are two buttons: 'Login' and 'Register'. A red error message 'Invalid credentials' is shown in a rounded rectangle. Below the error message are two input fields: the first contains the name 'Andrey', and the second is a password field with masked characters. At the bottom is a large green button labeled 'Enter'. Below the button is a link that says 'Forgot password?'.

Рисунок 4.20 – Неправильно введені дані

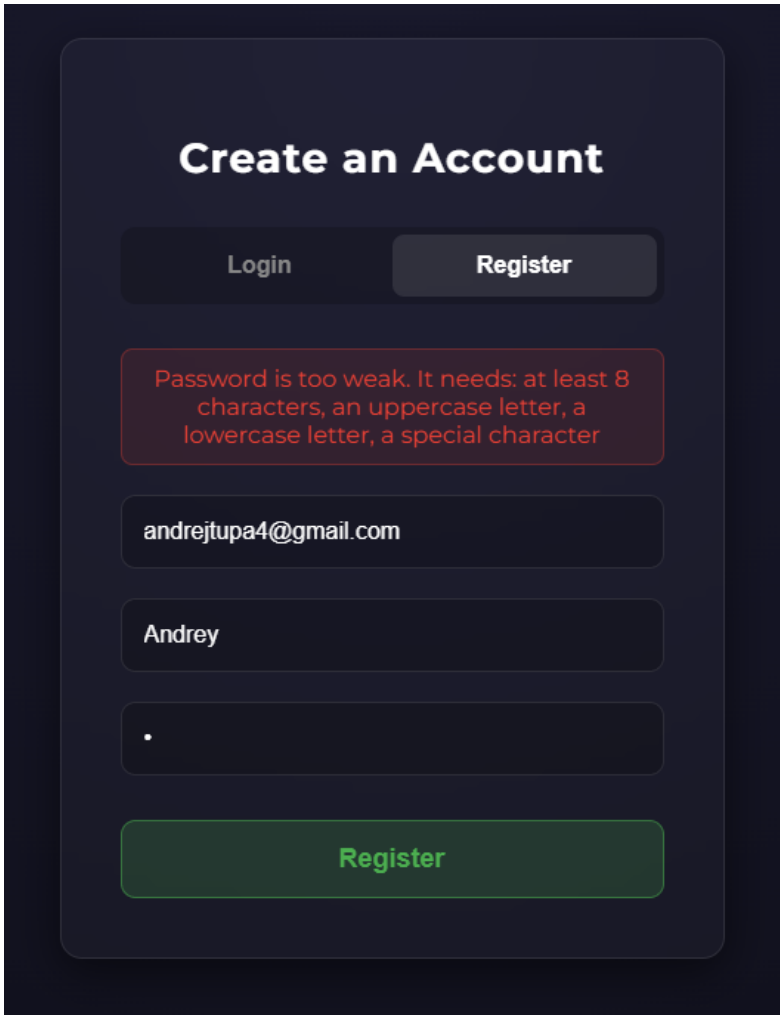


Рисунок 4.21 – Повідомлення про слабкий пароль



Рисунок 4.22 – Сповідання про недостатній баланс



Рисунок 4.23 – Неправильно введена капча

4.4 Методика та результати тестування

Для оцінювання та поліпшення якості програмного забезпечення було застосовано комплексну методику тестування, яка базується на принципах функціонального тестування та тестування інтеграції. На першому плані стояла потреба знайти слабкі місця: не лише алгоритмах розрахунку ймовірностей, але також у реакціях на збої та способах синхронізації даних між користувачем і сервером.

Процес тестування був розділений на три ключові фази. На першій фазі проводилося тестування модулів авторизації та керування активними сеансами. Було доведено, що система блокує нові реєстрації за однаковою поштою та забезпечує надійне хешування паролів. Якщо з'єднання обривається несподівано – стан грального столу не губиться, адже сервер його утримує. Відновлення відбувається повністю через WebSocket, коли клієнт повертається на сторінку.

Друга фаза була зосереджена на тестуванні ігрових лімітів. Захисні механізми продемонстрували стабільну поведінку під навантаженням. Наприклад, сервер надійно відхиляє від'ємні суми ставок. Так само система не

допускає спроб поставити більше, ніж є на балансі користувача. Тестування охопило й правила рулетки – мінімальна ставка має становити хоча б 10 фішок, а модуль Captcha довів свою безперебійну працездатність при падінні балансу гравця до критичного мінімуму. Це показало, що всі елементи взаємодіють прогнозовано. Результати свідчать про цілісність системи під екстремальними умовами. У процесі жоден сценарій не призвів до неконтрольованого стану.

На третій фазі ретельно перевірялася ігрова логіка симуляцій. Для блекджеку було протестовано алгоритм поведінки віртуального дилера: він завжди добирає карту, коли його сума дорівнює шістнадцяти або нижче. Рулетка ж показала математичну коректність розрахунку виграшних коефіцієнтів (1:1 для кольорів та 35:1 для конкретних номерів) із миттєвим оновленням балансу на стороні клієнта.

Загальні результати тестування дозволяють зробити висновок, що програмний комплекс працює стабільно в різних експлуатаційних умовах. Усі заявлені функціональні вимоги виконані в повному обсязі, механізми захисту від некоректних дій оператора працюють надійно. Відсутність критичних збоїв підтверджує готовність системи до практичного застосування.

4.5 Експериментальні дослідження та аналіз результатів

Перевірку точності математичних моделей здійснювали через практичний аналіз генератора псевдовипадкових чисел, що застосовують у рулетці. У ході тестування реалізовували алгоритмічну імітацію десяти тисяч поспіль обертів барабана без візуальної складової.

За результатами експерименту, частота випадіння кожного з 38 секторів склала приблизно 2.63%. Такі дані показують, що внутрішній механізм генерації випадкових значень працює стабільно. Жодних ознак систематичного перекосу на користь окремих цифр чи кольорів не спостерігалось. Результати свідчать про те, що процес дійсно хаотичний.

Відхилення від очікуваного розподілу мінімальні. Таким чином можна говорити про надійну реалізацію логіки випадковості.

Додатково було досліджено динаміку зміни віртуального балансу гравця при використанні стратегії «Мартінгейл» [18]. Експеримент підтвердив математичне очікування на рівні 5.26% на користь закладу: на довгій дистанції баланс гравця неминуче втрачав усі кошти. Оскільки в симуляторі відсутнє штучне обмеження максимальної ставки, головною причиною краху стратегії стало експоненційне зростання суми ставки при серії програшів, що призводило до швидкого вичерпання стартового капіталу гравця.

Достовірність отриманих результатів обґрунтовується їх повним збігом з класичними законами теорії ймовірностей та математичної статистики. Ефективність програмного забезпечення підтверджена шляхом тестування математичних припущень на практиці. Навчальне застосування цієї роботи дає змогу показувати студентам механіку випадкових процесів у доступному форматі. Крім того, реалізована платформа допомагає зрозуміти побудову вебзастосунків та аналізувати слабкі місця популярних ігрових стратегій.

4.6 Висновки до розділу 4

Тестування та експериментальні дослідження підтвердили повну працездатність симулятора та достовірність закладених у нього математичних моделей. Система демонструє стабільну роботу, ефективно захищає дані користувача та може застосовуватися як практичний інструмент для освітніх чи дослідницьких цілей.

ВИСНОВКИ

У роботі вирішено актуальну задачу розробки клієнт-серверного симулятора ігрових механік, який забезпечує наочну демонстрацію ймовірнісних алгоритмів та надійне збереження стану користувачів у реальному часі.

Починаючи з аналізу предметної області та наявних аналогів було сформовано вимоги до математичних моделей – американської рулетки та блекджеку та обґрунтовано вибір сучасного набору інструментів, зокрема React, Node.js та PostgreSQL. На основі цих вимог було спроектовано внутрішню будову системи – як на концептуальному, так і на технічному рівні, було розроблено структуру реляційної бази даних та визначено оптимізовані та ефективні формати обміну даними між клієнтом і сервером.

Під час етапу розробки було програмно реалізовано серверну та клієнтську частини симулятора. З метою захисту даних уведено механізм автентифікації через JWT, паралельно розроблено алгоритми генерації графічної Captcha-задачі. Крім того, налаштовано швидкий двосторонній обмін подіями через протокол WebSocket. На фінальному етапі було проведено функціональне тестування та експериментальні дослідження математичної бази. Отримані результати продемонстрували надійність захисних алгоритмів та емпірично підтвердили коректність розрахунку ймовірностей з показником на рівні 5.26%.

Що стосується наукової цінності – у роботі удосконалено підхід до побудови навчальних ігор-симуляторів завдяки поєднанню подієво-орієнтованої архітектури з механізмами JWT-автентифікації. Такий підхід значно скорочує затримку між клієнтом і сервером, одночасно запобігаючи втратам даних при раптовому обриві мережевого з'єднання. Крім того, глибше пропрацьовано техніку перевірки математичного очікування в цифрових середовищах шляхом автоматизації симуляції послідовних ігрових подій. Усе

це дає можливість за короткий термін переконатися в теоретичній марності стратегії типу «Мартінгейл», не вдаючись до кошовних експериментів.

Практична значущість отриманих результатів полягає у можливості впровадження розробленого симулятора в освітній процес закладів вищої освіти. Крім того, створена платформа може використовуватися кінцевими користувачами як інструмент для емпіричного тестування фінансових стратегій без реальних фінансових ризиків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Учасники проєктів Вікімедіа. Блекджек – вікіпедія. Вікіпедія. URL: [https://uk.wikipedia.org/wiki/Блекджек#:~:text=Блѐкджек%20\(англ.,від%20здібностей%20і%20вміння%20гравця](https://uk.wikipedia.org/wiki/Блекджек#:~:text=Блѐкджек%20(англ.,від%20здібностей%20і%20вміння%20гравця) (дата звернення: 08.05.2026).
2. Учасники проєктів Вікімедіа. Американська рулетка – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Американська_рулетка (дата звернення: 08.05.2026).
3. Saari V. Free online blackjack simulator & trainer. Bojoko - Your Award-Winning Source for UK Online Casinos and Bonuses. URL: <https://bojoko.com/casino/blackjack/simulator-trainer/> (date of access: 08.05.2026).
4. Probability Simulator coin, dice, roulette. Find calculators in your language | Search across locales | CalcBE. URL: <https://calcbe.com/en/calculators/probability-simulator/> (date of access: 08.05.2026).
5. Saari V. Roulette simulator & trainer app - practice free online. Bojoko - Your Award-Winning Source for Canadian Online Casinos and Bonuses. URL: <https://bojoko.ca/casino/roulette/simulator-trainer/> (date of access: 08.05.2026).
6. Учасники проєктів Вікімедіа. Node.js – вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/Node.js> (дата звернення: 08.05.2026).
7. Учасники проєктів Вікімедіа. SMTP – вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/SMTP> (дата звернення: 08.05.2026).
8. Учасники проєктів Вікімедіа. React – вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/React> (дата звернення: 08.05.2026).
9. Учасники проєктів Вікімедіа. CAPTCHA – вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/CAPTCHA> (дата звернення: 08.05.2026).
10. Учасники проєктів Вікімедіа. Об'єктно-орієнтоване програмування – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Об'єктно-орієнтоване_програмування (дата

звернення: 08.05.2026).

11. Учасники проєктів Вікімедіа. Unified modeling language – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Unified_Modeling_Language (дата звернення: 08.05.2026).

12. WebSocket API (WebSockets) - Web APIs | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (date of access: 08.05.2026).

13. Contributors to Wikimedia projects. Socket.IO - wikipedia. Wikipedia, the free encyclopedia. URL: <https://en.wikipedia.org/wiki/Socket.IO> (date of access: 08.05.2026).

14. Contributors to Wikimedia projects. PostgreSQL - wikipedia. Wikipedia, the free encyclopedia. URL: <https://en.wikipedia.org/wiki/PostgreSQL> (date of access: 08.05.2026).

15. Учасники проєктів Вікімедіа. ACID – вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/ACID> (дата звернення: 08.05.2026).

16. Express - Node.js web application framework. Express - Node.js web application framework. URL: <https://expressjs.com/> (date of access: 08.05.2026).

17. Учасники проєктів Вікімедіа. Тасування фішера – єйтса – вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Тасування_Фішера_–_Єйтса (дата звернення: 08.05.2026).

18. Учасники проєктів Вікімедіа. Мартінгейл – вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/Мартінгейл> (дата звернення: 08.05.2026).

ДОДАТОК А
Технічне завдання

Вступ

Продукт призначений для використання в освітньому процесі у закладах вищої освіти для наочної демонстрації законів теорії ймовірностей та математичної статистики.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація симулятора ігрових механік», затверджене наказом Національного університету «Запорізька політехніка» № 139 від 7 квітня 2026 р.

A.2 Призначення розробки

Програмний продукт призначений для інтерактивної симуляції ігрових механік та навчання користувачів в освітніх цілях без фінансових ризиків.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Система повинна виконувати наступні функції:

- реєстрація та авторизація користувачів із використанням JWT;
- хешування паролів користувачів у базі даних;
- ведення та оновлення персонального віртуального балансу користувача у базі даних;
- програмна реалізація логіки ігор «Американська рулетка» та «Блекджек» з повною валідацією та перевіркою результатів виключно на бекенді;
- відображення статистики в особистому кабінеті користувача;

– збереження поточного стану ігрового столу для захисту від раптової втрати з'єднання клієнтом.

A.3.2 Вимоги до інтерфейсу програми

Графічний інтерфейс користувача повинен бути сучасним, інтуїтивно зрозумілим та побудованим за принципами адаптивного вебдизайну для забезпечення коректного відображення як на десктопних, так і на мобільних пристроях.

A.3.3 Вимоги до надійності

Програма повинна бути стійкою до помилок введення даних та збійних ситуацій. У разі некоректної роботи повинні бути передбачені повідомлення, які допомагають користувачу виправити ситуацію. Для захисту даних користувачів паролі повинні зберігатися у зашифрованому вигляді. У випадку непередбаченої помилки або збою на сервері, система повинна мати можливість швидко перезапуститися та відновити роботу за декілька хвилин.

A.3.4 Умови експлуатації

Для експлуатації програмного симулятора ігрових механік необхідна наявність персонального комп'ютера чи мобільного пристрою.

A.3.5 Вимоги до складу та параметрів технічних засобів

Оскільки розроблена система є клієнт-серверним вебзастосунком, вимоги до технічних засобів та програмного забезпечення поділяються на дві категорії.

Для експлуатації серверної частини необхідно:

- оперативна пам'ять об'ємом 4 Гб і більше;
- не менше 20 Гб вільного простору на запам'ятовуючому пристрою апаратного засобу;

- процесор із тактовою частотою не менше 2.4 ГГц;
- встановлена операційна система Microsoft Windows;
- встановлена система керування базами даних PostgreSQL;
- встановлене середовище виконання Node.js.

Для експлуатації клієнтської частини необхідно:

- персональний комп'ютер, планшет або смартфон з доступом до мережі Інтернет;
- сучасний веббраузер із підтримкою технологій HTML5 та протоколу WebSocket

А.3.6 Вимоги до інформаційної та програмної сумісності

Серверна частина розробляється мовою JavaScript у середовищі виконання Node.js з використанням фреймворку Express.

Клієнтська частина розробляється мовою JavaScript на базі бібліотеки React.

Система управління базами даних: PostgreSQL.

Зв'язок між клієнтом і сервером повинен здійснюватися комбіновано: через протокол HTTP/HTTPS та WebSocket для двостороннього обміну ігровими подіями.

А.3.7 Вимоги до маркування і пакування

Програмний продукт поставляється у вигляді вихідних кодів, розміщених у репозиторії системи контролю версій (Git). Фізичне маркування та пакування не передбачено.

ДОДАТОК Б
Текст програми

Б.1 Текст файла index.js

```

import express from 'express';
import { createServer } from 'http';
import { Server } from 'socket.io';
import cors from 'cors';
import dotenv from 'dotenv';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import nodemailer from 'nodemailer';
import crypto from 'crypto';
import pool, { initDB } from './db.js';

dotenv.config();

const app = express();
const httpServer = createServer(app);

// Setup CORS
app.use(cors({
  origin: '*',
  methods: ['GET', 'POST']
}));

// Setup Socket.io
const io = new Server(httpServer, {
  cors: {
    origin: '*',
    methods: ['GET', 'POST']
  }
});

// Parse JSON bodies
app.use(express.json());

app.get('/', (req, res) => {
  res.send('Game Mechanics Simulator API is running');
});

// --- AUTHENTICATION ROUTES ---

const generateToken = (user) => {
  return jwt.sign(
    { id: user.id, username: user.username, email: user.email },

```

```

    process.env.JWT_SECRET || 'fallback_secret',
    { expiresIn: '24h' }
  );
};

// Email transporter
const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.SMTP_EMAIL,
    pass: process.env.SMTP_PASSWORD
  }
});

// Temporary storage for pending registrations
const pendingRegistrations = new Map();

app.post('/api/auth/register-init', async (req, res) => {
  try {
    const { username, email, password } = req.body;

    if (!username || !email || !password) {
      return res.status(400).json({ error: 'All fields are required' });
    }

    const missing = [];
    if (password.length < 8) missing.push('at least 8 characters');
    if (!/[A-Z]/.test(password)) missing.push('an uppercase letter');
    if (!/[a-z]/.test(password)) missing.push('a lowercase letter');
    if (!/\d/.test(password)) missing.push('a number');
    if (!/[@!%*?&^#]/.test(password)) missing.push('a special character');
  }
});

```

Б.2 Текст файла db.js

```

import pkg from 'pg';
const { Pool } = pkg;
import dotenv from 'dotenv';

dotenv.config();

const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,

```

```

password: process.env.DB_PASSWORD,
port: process.env.DB_PORT,
});

export const initDB = async () => {
  try {
    await pool.query(`
      CREATE TABLE IF NOT EXISTS users (
        id SERIAL PRIMARY KEY,
        username VARCHAR(50) UNIQUE NOT NULL,
        email VARCHAR(100) UNIQUE NOT NULL,
        password_hash VARCHAR(255) NOT NULL,
        balance INTEGER NOT NULL DEFAULT 1000,
        total_bets INTEGER NOT NULL DEFAULT 0,
        bets_won INTEGER NOT NULL DEFAULT 0,
        total_wagered INTEGER NOT NULL DEFAULT 0
      );
    `);

    console.log('[DB] Database initialized successfully.');
```

```

  } catch (err) {
    console.error('[DB] Error initializing database:', err);
  }
};

export default pool;
```

Б.3 Текст файла App.jsx

```

import { useState, useEffect } from 'react'
import { socket } from './socket'
import Auth from './components/Auth'
import Lobby from './components/Lobby'
import Roulette from './components/Roulette'
import Profile from './components/Profile'
import Blackjack from './components/Blackjack'
import ResetPassword from './components/ResetPassword'
import CaptchaPopup from './components/CaptchaPopup'
import Header from './components/Header'
import Footer from './components/Footer'

function App() {
  const [isConnected, setIsConnected] = useState(socket.connected);
  const [userData, setUserData] = useState(null);
```

```

const [currentGame, setCurrentGame] = useState(() => {
  return localStorage.getItem('lucky_chip_current_game') || null;
});
const [showProfile, setShowProfile] = useState(false);
const [isLoading, setIsLoading] = useState(true);
const [resetToken, setResetToken] = useState(null);

useEffect(() => {
  const params = new URLSearchParams(window.location.search);
  const tokenFromUrl = params.get('resetToken');
  if (tokenFromUrl) {
    setResetToken(tokenFromUrl);
  }

  const token = localStorage.getItem('lucky_chip_token');

  if (token) {
    socket.auth = { token };
    socket.connect();
  } else {
    setIsLoading(false);
  }

  function onConnect() {
    setIsConnected(true);
    console.log('Connected to socket server');
  }

  function onDisconnect() {
    setIsConnected(false);
    console.log('Disconnected from socket server');
  }

  function onLoginSuccess(user) {
    setUserData(user);
    setIsLoading(false);
  }

  function onConnectError(err) {
    console.error('Socket connection error:', err.message);
    if (err.message.includes('Authentication error')) {
      localStorage.removeItem('lucky_chip_token');
      localStorage.removeItem('lucky_chip_current_game');
      setUserData(null);
    }
  }

```

```
    setIsLoading(false);
  }
```

Б.4 Текст файла Auth.jsx

```
import { useState } from 'react';

export default function Auth({ onLoginSuccess }) {
  const [isLogin, setIsLogin] = useState(true);
  const [isVerificationStep, setIsVerificationStep] = useState(false);
  const [isForgotPassword, setIsForgotPassword] = useState(false);

  const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [verificationCode, setVerificationCode] = useState("");

  const [error, setError] = useState("");
  const [successMsg, setSuccessMsg] = useState("");
  const [loading, setLoading] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");
    setSuccessMsg("");
    setLoading(true);

    try {
      if (isForgotPassword) {
        // --- FORGOT PASSWORD INIT FLOW ---
        const response = await fetch('http://localhost:3001/api/auth/reset-password-init', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ email, clientId: window.location.origin }),
        });

        const data = await response.json();
        if (!response.ok) throw new Error(data.error || 'Failed to send reset link');

        setSuccessMsg('Password reset link sent to your email!');
        return;
      }
    }
  }
```

```

if (isLogin) {
  // --- LOGIN FLOW ---
  const response = await fetch('http://localhost:3001/api/auth/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ identifier: username, password }),
  });

  const data = await response.json();
  if (!response.ok) throw new Error(data.error || 'Authentication failed');

  localStorage.setItem('lucky_chip_token', data.token);
  onLoginSuccess(data.token, data.user);
}

```

Б.5 Текст файла Login.jsx

```

import { useState } from 'react';
import { socket } from '../socket';

export default function Login({ onLoginSuccess }) {
  const [username, setUsername] = useState("");

  const handleLogin = (e) => {
    e.preventDefault();
    if (username.trim()) {
      socket.emit('login', username.trim());
      socket.once('login_success', (userData) => {
        onLoginSuccess(userData);
      });
    }
  };

  return (
    <div className="login-container" style={{ textAlign: 'center', marginTop: '100px' }}>
      <h2>Welcome to Lucky Chip!</h2>
      <form onSubmit={handleLogin} style={{ display: 'flex', flexDirection: 'column', alignItems: 'center', gap: '10px' }}>
        <input
          type="text"
          placeholder="Enter your nickname"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          required

```

```

        maxLength={15}
        style={{ padding: '10px', fontSize: '16px' }}
      />
      <button type="submit" style={{ padding: '10px 20px', fontSize: '16px', cursor:
'pointer' }}>
        Enter
      </button>
    </form>
  </div>
);
}

```

Б.6 Текст файла ResetPassword.jsx

```

import { useState } from 'react';

export default function ResetPassword({ token, onResetSuccess }) {
  const [password, setPassword] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError("");

    if (password !== confirmPassword) {
      setError('Passwords do not match');
      return;
    }

    const missing = [];
    if (password.length < 8) missing.push('at least 8 characters');
    if (!/[A-Z]/.test(password)) missing.push('an uppercase letter');
    if (!/[a-z]/.test(password)) missing.push('a lowercase letter');
    if (!/\d/.test(password)) missing.push('a number');
    if (!/[@!$%*?&^#]/.test(password)) missing.push('a special character');

    if (missing.length > 0) {
      setError('Password is too weak. It needs: ' + missing.join(', '));
      return;
    }

    setLoading(true);

```

```

try {
  const response = await fetch('http://localhost:3001/api/auth/reset-password-
confirm', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ token, newPassword: password }),
  });

  const data = await response.json();
  if (!response.ok) throw new Error(data.error || 'Password reset failed');

  onResetSuccess();
} catch (err) {
  setError(err.message);
} finally {
  setLoading(false);
}
};

```

Б.7 Текст файла Lobby.jsx

```

export default function Lobby({ userData, onSelectGame }) {
  return (
    <div style={{ padding: '20px', maxWidth: '800px', margin: '0 auto' }}>
      <header style={{ display: 'flex', justifyContent: 'space-between', alignItems:
'center', borderBottom: '1px solid #ccc', paddingBottom: '10px' }}>
        <h2>Welcome, {userData.username}!</h2>
      </header>

      <h3 style={{ marginTop: '30px' }}>Select a Game:</h3>

      <div className="responsive-flex-col" style={{ display: 'flex', gap: '20px',
marginTop: '20px' }}>
        <div
          onClick={() => onSelectGame('roulette')}
          style={{
            border: '2px solid #4CAF50',
            borderRadius: '10px',
            padding: '20px',
            cursor: 'pointer',
            textAlign: 'center',
            flex: 1
          }}
        >

```

```

    >
    <h3 style={{ display: 'flex', alignItems: 'center', justifyContent: 'center' }}>
      Roulette 
    </h3>
    <p>Test your luck with the wheel!</p>
    <button style={{ padding: '10px 20px', background: '#4CAF50', color: 'white',
border: 'none', borderRadius: '5px', cursor: 'pointer' }}>
      Play
    </button>
  </div>

  <div
    onClick={() => onSelectGame('blackjack')}
    style={{
      border: '2px solid #2196F3',
      borderRadius: '10px',
      padding: '20px',
      cursor: 'pointer',
      textAlign: 'center',
      flex: 1
    }}
  >
    <h3 style={{ display: 'flex', alignItems: 'center', justifyContent: 'center' }}>
      Blackjack 
    </h3>
    <p>Beat the dealer to 21!</p>
    <button style={{ padding: '10px 20px', background: '#2196F3', color: 'white',
border: 'none', borderRadius: '5px', cursor: 'pointer' }}>
      Play
    </button>
  </div>
</div>
</div>
);
}

```

Б.8 Текст файла Footer.jsx

```

export default function Footer() {
  return (
    <footer style={{
      textAlign: 'center',

```

```
padding: '20px',
marginTop: 'auto',
color: 'rgba(255,255,255,0.3)',
fontSize: '14px',
borderTop: '1px solid rgba(255,255,255,0.05)'
}}>
  © 2026 Game Mechanics Simulator. All rights reserved.
</footer>
);
}
```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Програмна реалізація симулятора ігрових механік

Виконав: Андрій ТЮПА
ст. групи КНТ-132
Керівник: Сергій ЛЕОЩЕНКО
д-р філософ., доцент НУ «Запорізька політехніка»
2026 р.

Рисунок В.1 – Слайд 1

2

Об'єкт, предмет та мета роботи

Об'єкт дослідження - процес розроблення програмного забезпечення для симуляції ймовірнісних моделей та ігрових механік.

Предмет дослідження - методи та програмні засоби створення клієнт-серверних веб-додатків реального часу.

Мета дослідження - підвищення швидкості та безпеки емпіричного тестування математичних стратегій шляхом розробки веб-симулятора

Рисунок В.2 – Слайд 2

Задачі дослідження

Для досягнення поставленої мети у цій роботі необхідно розв'язати такі задачі:

- виконати аналіз існуючих рішень та систем;
- розробити математичні моделі ігор;
- спроектувати архітектуру та базу даних;
- програмно реалізувати клієнт-серверний додаток;
- виконати тестування і перевірку розробленого симулятора.

Рисунок В.3 – Слайд 3

Порівняння існуючих систем

Характеристика	Bojoko Blackjack Trainer	CalcBE Probability Simulator	Bojoko Roulette Trainer
Основна спрямованість	Вивчення стратегій карткової гри	Математичний розрахунок і моделювання ймовірностей	Вивчення та перевірка ймовірнісних стратегій рулетки
Гнучкість налаштувань	Низька	Висока	Низька
Навчальний аспект	Підказки щодо математично правильних ходів	Аналіз статистичних відхилень	Пропонуються різні математичні стратегії
Візуалізація даних	Графічний інтерфейс	Текстові результати та графіки	Графічний інтерфейс
Збереження прогресу та сесій	До оновлення вкладки	До оновлення вкладки	До оновлення вкладки

Рисунок В.4 – Слайд 4

Математичні моделі

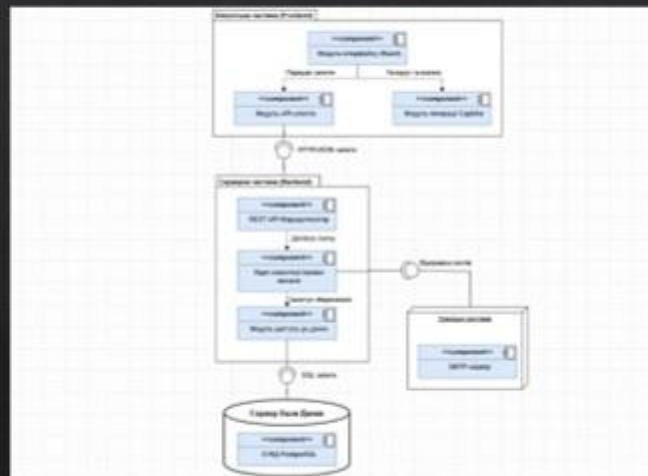
Математичне очікування: $M(x) = \sum xP(x)$

Очікування для рулетки: $M(x) = 1 * \frac{18}{38} - 1 * \frac{20}{38} \approx -0.0526$

Формула залежних подій для блекджеку: $P(A * B) = P(A) * P(B/A)$

Рисунок В.5 – Слайд 5

Компонентна архітектура програмного забезпечення



Діаграма компонентів

Рисунок В.6 – Слайд 6

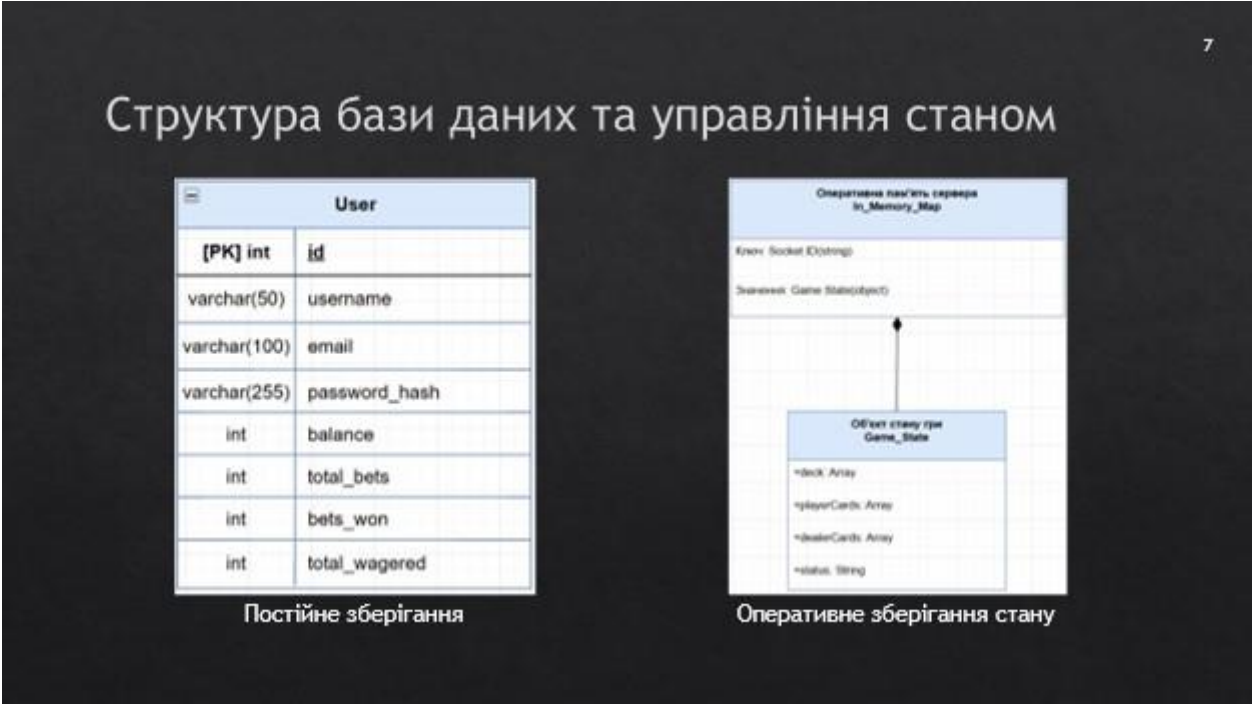


Рисунок В.7 – Слайд 7

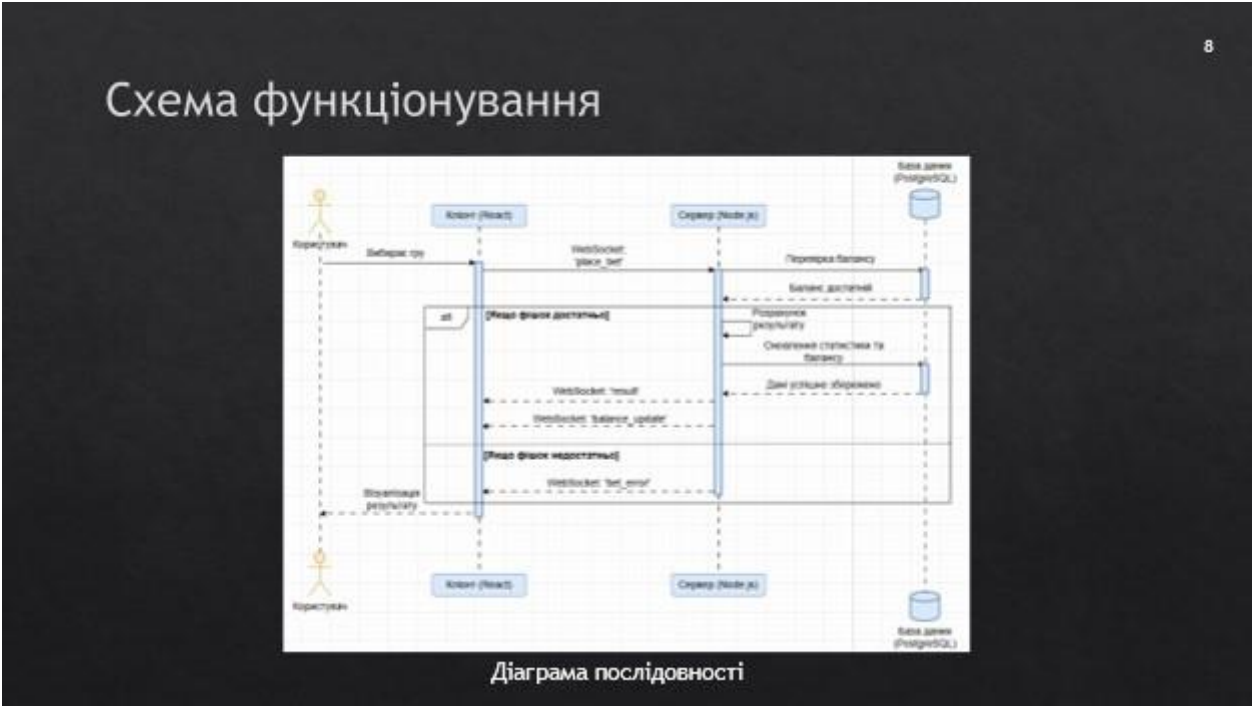


Рисунок В.8 – Слайд 8

9

Вибір мови програмування та технологічного стеку

Характеристика	Node.js	Python (Django)	PHP
Модель виконання	Асинхронна (Event Loop)	Синхронна	Синхронна
Робота з WebSocket	Нативна	Підтримується, але вимагає додаткових архітектурних надбудов	Потребує розгортання окремих зовнішніх сервісів
Мова розробки	JavaScript	Python	PHP
Швидкість обробки I/O	Дуже висока	Середня	Низька
Серіалізація даних (JSON)	Нативна	Потребує додаткових кроків	Потребує виклику спеціальних функцій
Споживання ресурсів сервера	Низьке	Середнє	Високе

Рисунок В.9 – Слайд 9

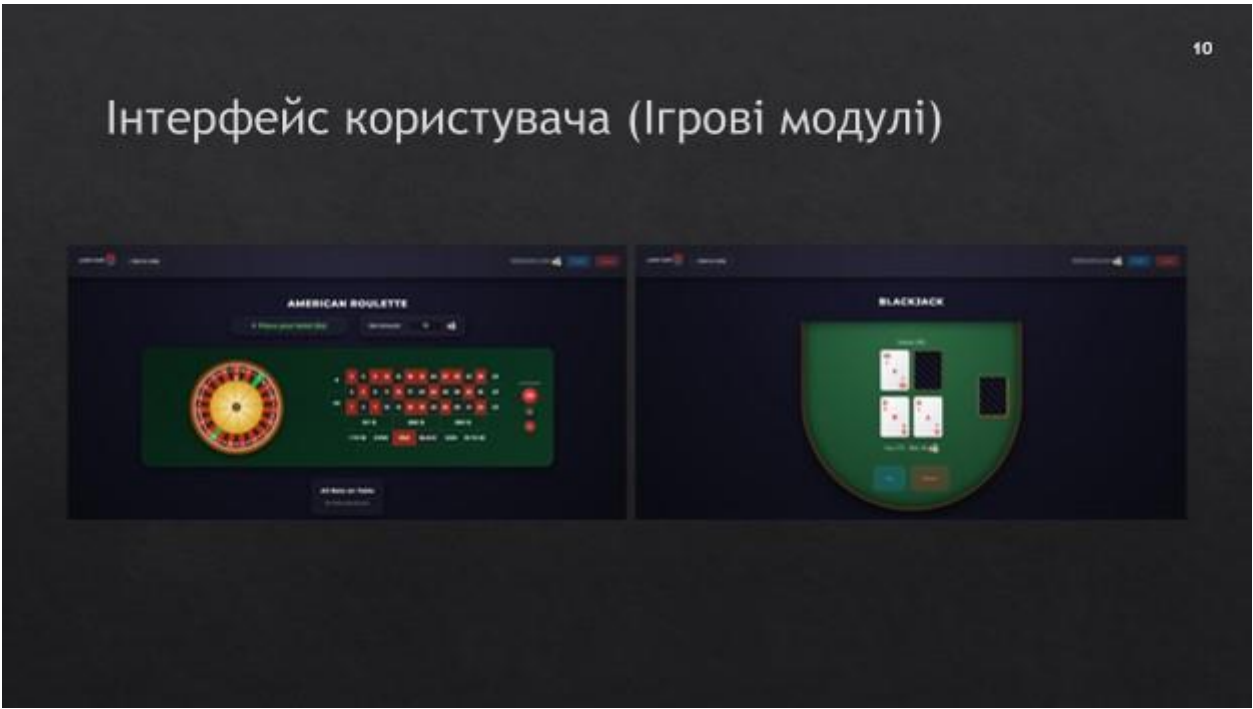


Рисунок В.10 – Слайд 10

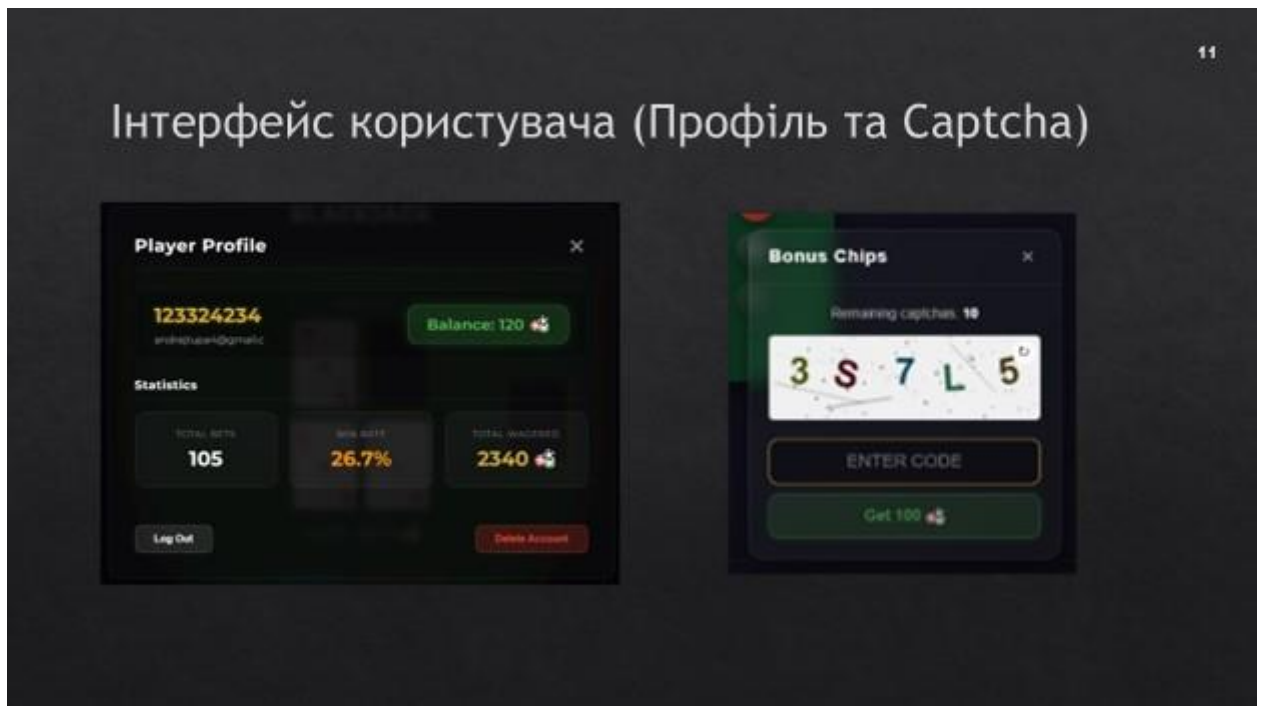


Рисунок В.11 – Слайд 11

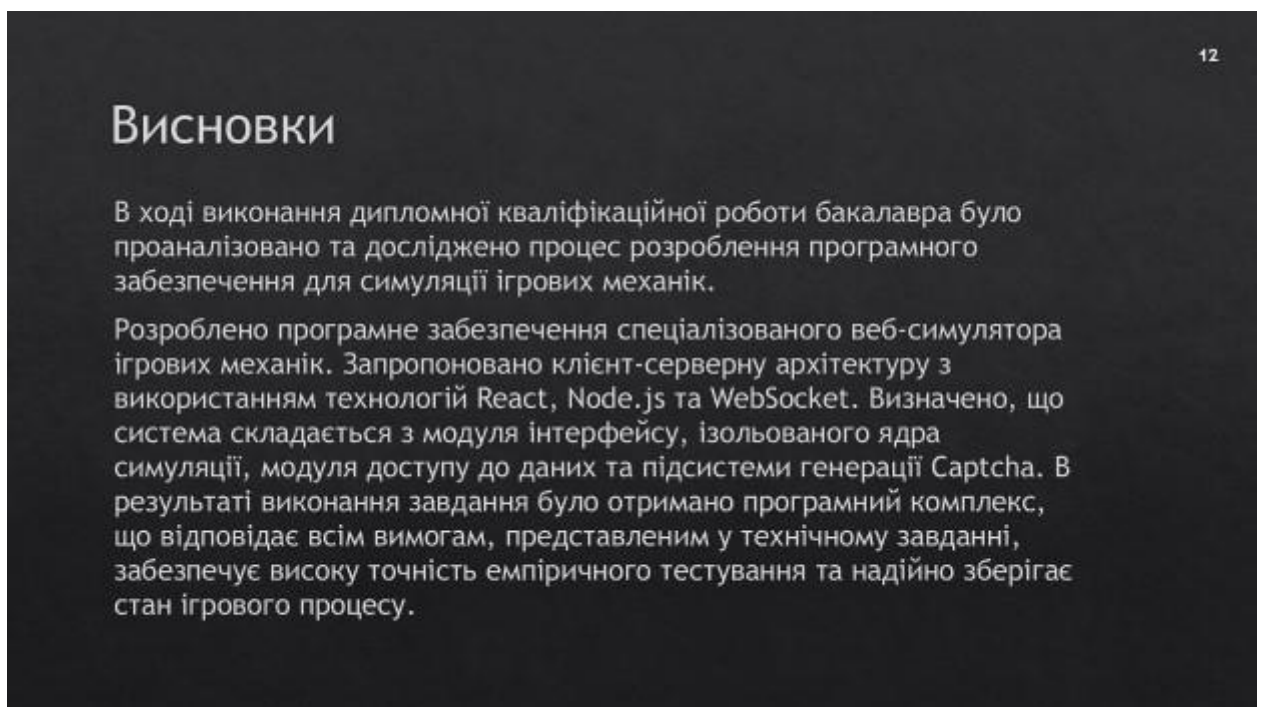


Рисунок В.12 – Слайд 12