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

КВАЛІФІКАЦІЙНА РОБОТА

на тему

СИСТЕМА ВИБОРУ ПЕРЕМОЖЦІВ СЕРЕД ЗАРЕЄСТРОВАНИХ КОРИСТУВАЧІВ

Виконав: студент групи 1П-21
Спеціальності 121 Інженерія програмного забезпечення
Сергій ШЕМШУР
Керівник:
Наталя ФАЛЬЧЕНКО

Черкаси 2025

АНОТАЦІЯ

У ході виконання кваліфікаційної роботи було обґрунтовано актуальність створення онлайн-платформи для проведення чесних, прозорих і безпечних розіграшів. Проаналізовано існуючі методи генерації випадкових чисел, принцип Provably Fair для підтвердження достовірності результатів, а також функціонал сервісу Giveaway.com з урахуванням його переваг і недоліків. Спроєктовано архітектуру системи TrustLuck, що охоплює як зовнішню, так і внутрішню структуру платформи та взаємозв’язки між її компонентами. Реалізовано алгоритм вибору переможців на основі механізму автентифікації повідомлень HMAC і криптографічного генератора випадкових послідовностей. Проведено тестування системи на відповідність нефункціональним вимогам, зокрема щодо безпеки, навантаження та чесності алгоритму вибору переможців. Отримані результати підтверджують практичну цінність реалізованого підходу та потенціал платформи TrustLuck для подальшого розвитку й використання в реальних умовах.

ABSTRACT

As part of the qualification work, the relevance of creating an online platform for conducting fair, transparent, and secure giveaways was substantiated. Existing methods of random number generation were analyzed, along with the Provably Fair principle for verifying result authenticity and the functionality of the Giveaway.com service, considering its advantages and limitations. The architecture of the TrustLuck system was designed, covering both the external and internal structure of the platform as well as the interconnections between its components. An algorithm for selecting winners was implemented based on the HMAC message authentication mechanism and a cryptographic random sequence generator. The system was tested for compliance with non-functional requirements, particularly in terms of security, performance under load, and fairness of the winner selection algorithm. The results confirm the practical value of the implemented approach and the potential of the TrustLuck platform for further development and real-world application.

ЗМІСТ

ВСТУП

У сучасному цифровому світі розіграші та конкурси є одним із найпоширеніших способів залучення й утримання аудиторії. Проте питання чесності, безпеки та прозорості їх проведення залишається відкритим, оскільки більшість існуючих платформ не гарантують об’єктивність вибору переможців. Недовіра користувачів, відсутність незалежної перевірки результатів, а також можливість маніпуляцій ставлять під сумнів ефективність таких механізмів.

Розроблена платформа TrustLuck вирішує ці проблеми шляхом верифікації учасників розіграшу, забезпечення стійкості до високих навантажень, впровадження системи перевірки результатів за допомогою криптографічних методів, реалізації алгоритму вибору переможців на основі поєднання серверних та користувацьких даних, а також впровадження механізмів захисту від атак і несанкціонованого доступу.

Об’єкт і предмет дослідження

Об’єктом дослідження є система організації та проведення онлайн-розіграшів, що охоплює створення, управління та визначення результатів розіграшів з автоматизованою обробкою даних учасників.

Предметом дослідження є методи, алгоритми та механізми, які використовуються для забезпечення чесності, прозорості та безпеки онлайн-розіграшів

Метою кваліфікаційної роботи є розробка системи для проведення чесних, прозорих та безпечних розіграшів із використанням криптографічних методів, сучасних веб-технологій і механізмів автоматизації.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Проаналізувати існуючі методи організації онлайн-розіграшів, оцінивши їхні підходи до забезпечення чесності, безпеки та автоматизації з метою виявлення сильних і слабких сторін.

2. Розробити архітектуру системи управління розіграшами, що включатиме компоненти для взаємодії з користувачами, оброблення даних і автоматизації процесів.

3. Реалізувати криптографічний алгоритм для справедливого визначення переможців, який забезпечить можливість незалежної перевірки результатів.

4. Створити механізм верифікації учасників, який гарантуватиме захист від несанкціонованих дій і надійність реєстрації.

5. Впровадити відображення та оновлення даних у реальному часі, забезпечуючи актуальність інформації щодо розіграшу.

6. Провести тестування та аналіз системи, оцінивши її функціональність, безпеку, продуктивність і можливість перевірки результатів користувачами

Методи дослідження

В роботі було застосовано 5 методів дослідження:

– Теоретичний та порівняльний аналіз (для вибору оптимального підходу до забезпечення чесності, прозорості та безпеки)

– Системне моделювання (для розробки архітектури системи TrustLuck)

– Алгоритмічне проєктування (через створення криптографічного алгоритму для визначення переможців)

– Емпіричне програмування (безпосередньо для практичної розробки коду)

– Тестування та верифікація (для загальної перевірки коректності роботи системи, включаючи функціональність API, обробку даних і достовірність алгоритмів

РОЗДІЛ 1 АНАЛІЗ ІСНУЮЧИХ МЕХАНІЗМІВ ВИПАДКОВОЇ ГЕНЕРАЦІЇ ТА ДОСЛІДЖЕННЯ ПЛАТФОРМ ДЛЯ ПРОВЕДЕННЯ РОЗІГРАШІВ

1.1 Генератори випадкових чисел та їх застосування

Генератори випадкових чисел (Random Number Generators, RNG) – це алгоритми або апаратні пристрої, які створюють послідовність чисел, що не мають очевидного зв’язку між собою і здаються випадковими. Вони широко застосовуються в програмуванні, криптографії, іграх, симуляціях і, зокрема, в різних онлайн-розіграшах[2].

Основна мета RNG полягає у забезпеченні непередбачуваності результатів, що є вкрай важливим для вирішення завдань, де необхідна висока міра випадковості та відсутність закономірностей. Такі генератори знаходять застосування у різноманітних галузях, зокрема у математичному моделюванні, криптографії, тестуванні програмного забезпечення, розробці ігрових систем та організації розіграшів, де специфіка вимог до безпеки, якості генерації та надійності може суттєво варіюватися залежно від контексту. Тому, розрізняють кілька основних типів RNG, які відрізняються за принципом роботи, рівнем випадковості та призначенням:

1. Істинні генератори випадкових чисел (TRNG) – це пристрої або алгоритми, які генерують числа на основі фізичного процесу, а не детермінованого алгоритму. Ці фізичні процеси за своєю суттю непередбачувані та можуть включати такі явища, як електронний шум, радіоактивний розпад або інші квантово-механічні ефекти. Ключовою характеристикою TRNG є те, що вони покладаються на ентропію з фізичного світу, що робить їхні результати справді випадковими та невідтворюваними[2].

Особливості TRNG:

– джерела ентропії: істинні генератори випадкових чисел (TRNG) отримують випадковість із природних фізичних явищ, зокрема теплового чи атмосферного шуму, а також радіоактивного розпаду;

– непередбачуваність: завдяки опорі на фізичні процеси – результати, які видають TRNG, є принципово непередбачуваними;

– недетермінованість: на відміну від алгоритмічних аналогів, TRNG не спираються на початкове значення (seed), а кожен результат є унікальним і не пов’язаним із попередніми;

– сфери використання: ці генератори застосовуються в ситуаціях, де є критична потреба у високому рівні безпеки, наприклад, при створенні криптографічних ключів, забезпеченні захищеного зв’язку чи у створенні токенів для багатофакторної автентифікації.

2. Псевдовипадкові генератори випадкових чисел PRNG – це алгоритми, які використовують математичні формули або попередньо обчислені таблиці для створення послідовностей чисел, які виглядають випадковими. Ключова відмінність між PRNG і TRNG полягає в тому, що PRNG є детермінованими; вони починаються з початкового початкового значення та використовують його для створення послідовності чисел. Якщо початкове значення відоме, можна відтворити всю послідовність[2].

Особливості PRNG:

– алгоритмічний підхід: псевдовипадкові генератори (PRNG) базуються на математичних формулах, які формують послідовності чисел;

– роль початкового значення: результат роботи PRNG залежить від початкового параметра (seed), і зміна цього значення генерує нову послідовність;

– відтворюваність: при використанні однакового seed, PRNG відтворює ідентичну послідовність, що є зручним для тестування чи налагодження програм;

– ефективність: ці генератори вирізняються високою швидкістю роботи та економним використанням обчислювальних ресурсів порівняно з іншими типами;

– області застосування: PRNG оптимальні для задач, де важлива повторюваність, таких як моделювання процесів, розробка ігор чи створення процедурного вмісту.

3. Криптографічно стійкі генератори (CSPRNG) поєднують переваги PRNG із підвищеною захищеністю від криптоаналізу. Навіть при частковому розкритті внутрішнього стану алгоритму наступні значення передбачити практично неможливо, що робить їх незамінними у системах, де безпека є першочерговою[2].

Особливості CSPRNG

– криптографічна стійкість: CSPRNG розроблені з урахуванням захисту від різноманітних атак, включаючи компрометацію стану, та забезпечують стійкість до передбачення;

– використання ентропії: ці генератори інтегрують дані ентропії, отримані від TRNG, для підвищення рівня випадковості;

– неможливість відтворення: навіть за частковим знанням внутрішнього стану, майбутні результати залишаються обчислювально непередбачуваними;

– стандартизація: CSPRNG підлягають суворим тестам і мають відповідати встановленим криптографічним нормам для забезпечення безпеки;

– області використання: вони застосовуються для створення криптографічно захищених елементів, таких як ключі шифрування чи унікальні ідентифікатори. Також часто застосовуються в різних лотереях та розіграшах, де чесність має критичне значення.

Розробка системи TrustLuck, яка призначена для організації чесних онлайн-розіграшів, вимагає ретельного підходу до вибору генератора випадкових чисел (ГВЧ). Оскільки основна мета платформи – забезпечити прозорість, безпеку та можливість перевірки результатів, тип ГВЧ має

відповідати цим критеріям. Розглянемо, який саме тип генератора є найбільш доцільним для реалізації механізму випадкового вибору переможців, враховуючи особливості кожного з них.

Отже, ключовими вимогами до ГВЧ є непередбачуваність результатів, захист від маніпуляцій і можливість учасників самостійно верифікувати коректність вибору переможців. З огляду на ці потреби, PRNG не є оптимальним вибором. Хоча PRNG вирізняються швидкістю та ефективністю, їхня детермінованість – тобто залежність від початкового значення (seed) – робить їх уразливими. Якщо зловмисник отримає доступ до цього значення, він зможе передбачити результати розіграшу, що повністю суперечить принципам чесності, які лежать в основі системи TrustLuck. Таким чином, PRNG більше підходять для задач, де безпека не є пріоритетом, наприклад, для генерації випадкових подій у комп'ютерних іграх.

Інший варіант – істинні генератори випадкових чисел (TRNG), які базуються на фізичних джерелах ентропії. TRNG забезпечують найвищий рівень випадковості, адже їхні результати є недетермінованими та не залежать від початкових параметрів. На перший погляд, це здається ідеальним рішенням для TrustLuck, адже непередбачуваність – одна з ключових вимог. Проте на практиці TRNG мають суттєві обмеження. Для їхньої роботи потрібне спеціалізоване апаратне забезпечення, наприклад, плати з датчиками шуму або квантові генератори, що може бути дорогим і недоступним для типового вебзастосунку. Крім того, інтеграція такого обладнання у серверну інфраструктуру TrustLuck потребує додаткових витрат на закупівлю, налаштування та обслуговування, що ускладнює масштабування системи. Наприклад, для невеликої платформи, яка працює на стандартному сервері, використання TRNG може виявитися економічно невигідним, адже використання апаратних генераторів також залежить від фінансового становища розробника.

Оптимальним рішенням для системи TrustLuck є використання CSPRNG. Вони генерують числа з високим рівнем непередбачуваності, що захищає від

будь-яких спроб маніпуляцій. На відміну від псевдовипадкових генераторів (PRNG), які можуть бути передбачуваними при доступі до початкового значення, CSPRNG спирається на джерела ентропії, такі як системний шум, що робить його результати практично неможливими для прогнозування. Водночас, порівняно зі справжніми генераторами (TRNG), які потребують дорогого апаратного забезпечення, CSPRNG є доступним і зручним для реалізації у вебзастосунку. До того ж, у поєднанні з підходом Provably Fair (який буде розглянуто в підрозділі 1.2) розроблений механізм випадкового вибору переможців забезпечить впевненість учасників у чесності процесу з можливістю самостійно відтворити алгоритм та перевірити результати.

Розглянемо метод randomBytes з бібліотеки crypto як приклад найпоширенішого CSPRNG у програмному середовищі Node.js, яке слугує основою для розробки платформи. Згідно з офіційною документацією Node.js метод crypto.randomBytes генерує криптографічно безпечні псевдовипадкові дані[8]. Зазначено, що метод не завершується, доки не буде достатньо ентропії, і зазвичай це займає кілька мілісекунд, хоча затримки можливі якщо викликати метод відразу після завантаження системи, коли ентропія обмежена. Для генерації випадкових даних crypto.randomBytes спирається на криптографічні можливості операційної системи, що зазвичай реалізується через системні джерела ентропії, такі як /dev/urandom на Unix-подібних системах (Linux, macOS) або CryptGenRandom (чи сучасніший BCryptGenRandom) на Windows, як це прийнято в бібліотеці OpenSSL, на якій базується модуль crypto у Node.js. Метод приймає аргумент, який визначає кількість байтів, і повертає об’єкт типу Buffer (спеціальний клас, який використовується для роботи з двійковими даними), що містить криптографічно безпечні псевдовипадкові байти відповідної довжини. Ця функція є асинхронною, і в разі недостатньої ентропії вона може тимчасово блокувати цикл подій, хоча в сучасних системах із стабільним джерелом ентропії це трапляється рідко.

У межах підходу Provably Fair, зазначений метод дозволяє генерувати надійне початкове значення – server seed, яке використовується на стороні

сервера для формування випадкових результатів за допомогою алгоритму HMAC-SHA256. Це значення комбінується з client seed – параметром, сформованим на основі даних, введених користувачами, що забезпечує як непередбачуваність, так і можливість перевірки результатів. Такий підхід відповідає принципам прозорості та чесності, що є критично важливими для роботи платформи.

1.2 Підхід Provably Fair

Provably Fair (Достовірно чесний) – це технологія, яка дозволяє гравцям з усього світу самостійно та особисто перевіряти чесність розіграшів та азартних ігор. Щоб це стало можливим, використовують спеціальний код, який називають provably fair алгоритмом[1].

Термін «Provably Fair» (доказово чесний) стосується алгоритму, який використовує технології для підвищення ефективності та прозорості онлайн-рандомізації, зберігаючи при цьому найвищий рівень чесності та відкритості. Коротко кажучи, алгоритм застосовує блокчейн-технології для того, щоб забезпечити максимально випадковий результат.

Алгоритм Provably Fair може бути реалізований різними способами, але найпоширеніший з них передбачає обчислення трьох ключових змінних щоразу, коли проводиться розіграш: server seed, client seed та nonce. І client seed, і server seed однаково важливі для забезпечення чесного результату. Характеристика ключових змінних:

– server seed – випадкове значення, згенероване системою перед розіграшем, хеш якого публікується для забезпечення прозорості;

– client seed – значення, створене на основі введених даних учасником для унікальності результату;

– nonce – лічильник (ціле число), який збільшується на 1 кожного разу, коли гравець бере участь у новій грі, раунді або розіграші.

На рис. 1.1 зображено схему роботи алгоритму Provably Fair. Відповідно вимогам алгоритму випадкового вибору у системі TrustLuck:

Генерація Client seed виконується таким чином:

1. Значення може бути змінено до кінця розіграшу (зі збільшенням учасників)
2. Кожен email учасника є частиною генерації clientseed.

Так ви можете впевнитися, що сайт не знає ваш clientseed заздалегідь.

Генерація Server seed надається з боку самого сервера та включає два значення:

1. Data – нехешоване значення (використовується для верифікації та публікується після розіграшу).
2. SHA-256 – хеш від Server Seed, який публікується до початку гри/розіграшу, щоб унеможливити підробку.

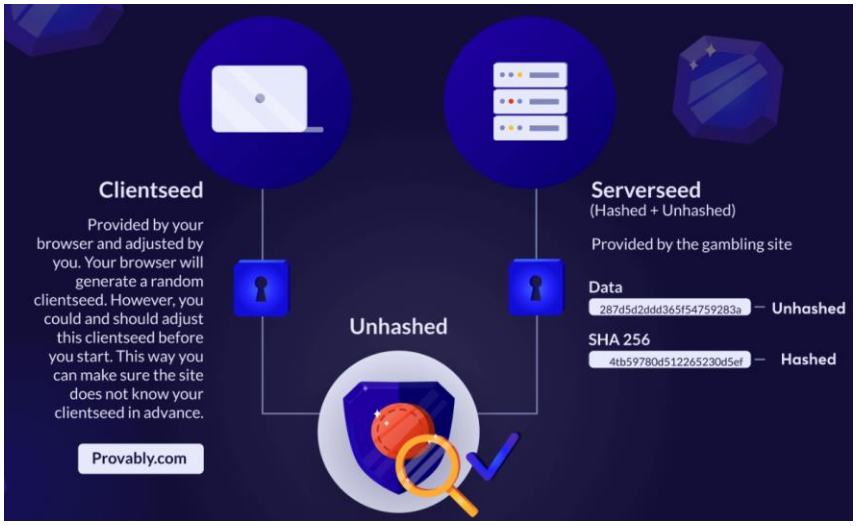


Рисунок 1.1 – Схема роботи алгоритму Provably Fair

Оптимальним способом генерації Server Seed є використання криптографічно безпечного генератора випадковості (CSPRNG), оскільки він забезпечує високий рівень ентропії та непередбачуваність.

Для визначення переможців у підході Provably Fair використовують криптографічний алгоритм автентифікації повідомлень HMAC-SHA256, який

обчислює унікальне числове значення (score) для кожного учасника на основі комбінації Server Seed (секретного ключа від сервера) та Client Seed (вхідного значення від клієнта). Отримані значення потім використовуються для чесного та прозорого вибору переможців.

HMAC (Hash-based Message Authentication Code) – це механізм, який поєднує хеш-функцію з секретним ключем для створення унікального коду (підпису) до кожного повідомлення. Такий код дозволяє перевірити, чи повідомлення було змінене, і чи воно надійшло від автентичного джерела. Хеш-функція відповідає за створення контрольної суми повідомлення, а секретний ключ забезпечує додатковий рівень безпеки, оскільки лише той, хто володіє цим ключем, може згенерувати правильний HMAC. Цей механізм може поєднувати різні хеш-функції, в залежності від вимог до безпеки, продуктивності та сумісності із системою або протоколом[5]. В таблиці 1.1 представлено порівняння хеш-функцій за їхніми перевагами, недоліками та статусом використання.

Таблиця 1.1 – Список основних хеш-функцій для HMAC

Хеш-функція	Переваги	Недоліки	Статус
MD5	Дуже швидка, мала довжина	Зламана, є колізії, уразлива до атак	Не використовується
SHA-1	Швидка, проста	Колізії знайдено, вважається небезпечною	Не рекомендується
SHA-256	Сильна безпека, стандарт де-факто	Повільніша за SHA-1	Рекомендується
SHA-384	Баланс між безпекою і швидкістю	Менш популярна	Рекомендується
SHA-512	Дуже висока безпека	Важча для старих пристроїв	Для критичних систем
SHA-3 (256)	Новий стандарт, інший підхід	Ще не всюди підтримується	Перспективна

Хеш-функцію SHA-256 найчастіше використовують у HMAC. На рис. 1.2 зображено схему роботи механізму HMAC з хеш-функцією SHA-1

У таблиці 1.2 показано залежність розміру хешу L та блоку b від типу хеш-функції H, опишемо ці позначення:

– b, block_size – розмір блоку в байтах;

– L – розмір в байтах рядки, що повертається хеш-функцією H; L залежить від вибраної хеш-функції і зазвичай менше розміру блоку;

– H, hash – хеш-функція;

Інші позначення:

– ipad – блок виду (0x36 0x36 0x36 ... 0x36), де байт 0x36 повторюється b раз

– K, key – секретний ключ (загальний для відправника та одержувача);

– K0 – змінений ключ K (зменшений або збільшений до розміру блоку (до b байт));

– opad – блок виду (0x5c 0x5c 0x5c ... 0x5c), де байт 0x5c повторюється b раз

– text – повідомлення (дані), яке передаватиметься відправником та справжність якого перевірятиметься одержувачем;

– n – довжина повідомлення text в бітах.

У формулі наведено загальний вигляд алгоритму HMAC[5].

$$\mathrm{HMAC}_K(text) = \mathrm{H}\left((K_0 \oplus opad) \| \mathrm{H}\left((K_0 \oplus ipad) \| text\right)\right) \tag{1.1}$$

де,

«⊕» – позначає побітову виключну диз’юнкцію (XOR);

«||» – позначає конкатенацію

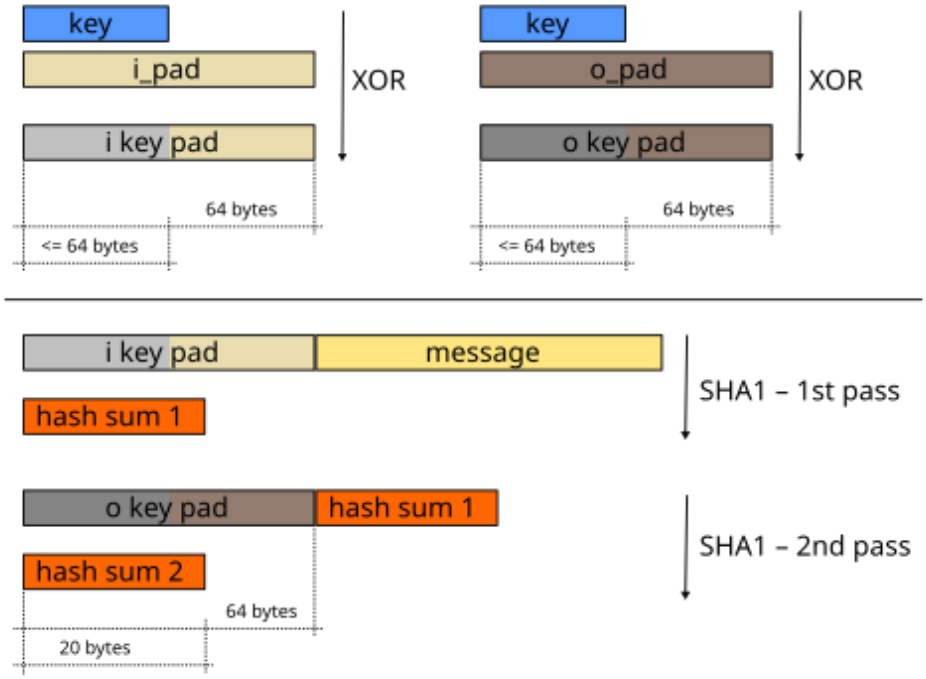


Рисунок 1.2 – Схема роботи алгоритму HMAC SHA-1

Таблиця 1.2 – Параметри хеш-функцій

Хеш-функція, H	b (блок, байти)	L (хеш, байти)
MD5	64	16
SHA-1	64	20
SHA-256	64	32
SHA-384	128	48
SHA-512	128	64
SHA3-256	136	32

Етапи роботи алгоритму HMAC:

1. Отримати K0 шляхом зменшення чи збільшення ключа K до розміру блоку (до b байт). Якщо довжина ключа K рівна розміру блоку, то копіюємо K в K0 та без змін переходимо до 2-го кроку. Якщо довжина ключа K більше розміру блоку, то до ключа K застосовуємо хеш-функцію H, отримуємо рядок розміром в L байт, додаємо нулі до правої частини цього рядка для

створення рядка розміром в b байт, копіюємо результат в K0 та переходимо до 2-го кроку. Якщо довжина ключа K менше розміру блоку, то додаємо нулі до правої частини K для створення рядка розміром в b байт, копіюємо результат в K0 та переходимо до 2-го кроку.

2. Використовуючи формулу отримати блок Si розміром в b байт за допомогою операції XOR.

$$S_i = XOR(K_0, ipad) = K_0 \oplus ipad \quad (1.2)$$

3. Використовуючи формулу отримати блок So розміром в b байт за допомогою операції XOR.

$$S_o = XOR(K_0, opad) = K_0 \oplus opad \quad (1.3)$$

4. Розбити повідомлення (дані, набір байт) text на блоки з розміром b байт.

5. Склеїти рядок (послідовність байт) Si з кожним блоком повідомлення M.

6. До рядка, отриманого на минулому кроці, застосувати хеш-функцію H.

7. Склеїти рядок So з рядком, отриманим від хеш-функції H на минулому кроці.

8. До рядка, отриманого на минулому кроці, застосувати хеш-функцію H

Ключі розміром менше L байт, вважаються небезпечними. Рекомендується вибирати ключі випадково і регулярно змінювати їх. Ключі розміром більше L байт, суттєво не збільшують стійкість функції, можуть використовуватися, якщо є сумніви у випадковості даних, що використовуються для створення ключа та отримуваних від генератора випадкових чисел. Розмір ключа K повинен бути більшим або рівним L/2 байт.

На рис 1.3 наведено фрагмент реалізації HMAC на псевдокоді

```
FUNCTION hmac( key, msg ) :
     // Якщо розмір ключа більше, ніж розмір блоку ...
     IF length( key ) > block_size THEN  :
         // Укорочуємо ключ до розміру результату хеш-функції
        key = hash (key)
        // (Розмір результату хеш-функції зазвичай менший (а не рівний), ніж розмір блоку хеш-функції)
     END_IF
    // Якщо ключ менше, ніж розмір блоку хеш-функції ...
     IF length( key ) < block_size THEN :
         // Доповнюємо ключ нульовою послідовністю
        key = key ? zeroes( block_size - length( key ))
        // оператор "?" виконує склеювання рядків (послідовностей байт)
     END_IF

    ipad = [ '\x36' * block_size ]
    // оператор "*" вказує кількість повторень послідовності байт,
    // а block_size - розмір блоку хеш-функції,
    opad = [ '\x5c' * block_size ]

    ikeypad = ipad ⊕ key
    // оператор "⊕" виконує побітове виключне АБО (xor)
    okeypad = opad ⊕ key

    RETURN hash( okeypad ? hash( ikeypad ? msg ) )
     // Оператор " ? " виконує склеювання рядків
END_FUNCTION
```

Рисунок 1.3 – Фрагмент реалізації HMAC на псевдокоді

У Provably Fair системах результат розіграшу генерується шляхом обчислення HMAC з використанням Server Seed як ключа та комбінації Client Seed і nonce як повідомлення. Це дозволяє створити унікальний та непередбачуваний результат для кожного розіграшу, оскільки навіть мінімальна зміна в Client Seed або nonce призводить до зовсім іншого HMAC-значення. Такий підхід гарантує, що:

1. Організатор не може вплинути на результати розіграшу. Server Seed створюється заздалегідь за допомогою криптографічно безпечного генератора випадкових чисел, а його хеш публікується ще до початку розіграшу. Це гарантує, що організатор не зможе підібрати бажаний ключ – він створюється випадковим чином, – і що після публікації хешу змінити Server Seed неможливо, оскільки будь-яка модифікація порушить відповідність хешу.

2. Учасник може самостійно перевірити чесність проведеного розіграшу. Всі ключові елементи – Server Seed, його хеш, Client Seed, та

алгоритм обчислення – відкриті й прозорі. Кожен учасник може взяти вихідні дані, вручну або за допомогою скрипту обчислити HMAC і впевнитися, що переможці дійсно були обрані без шахрайства.

3. Результат визначається виключно комбінацією двох незалежних значень. Server Seed – це випадково згенероване секретне значення, яке організатор фіксує, публікуючи його хеш до початку розіграшу. Client Seed – публічний параметр, сформований на основі даних, які організатор не контролює (наприклад, список email-адрес учасників або інформація з публічних джерел, таких як блокчейн чи погодні API). Разом ці значення створюють унікальний результат, що повністю виключає можливість стороннього впливу.

4. Кожен отриманий результат є унікальним та непередбачуваним. Навіть незначна зміна в Client Seed або nonce (наприклад, заміна одного символу в email або зміна порядкового номера) призведе до кардинально іншого значення HMAC. Тому заздалегідь передбачити результат або підлаштувати виграшний варіант є неможливим.

1.3 Огляд платформи Giveaway.com

Giveaway.com – це інноваційна онлайн-платформа, що спеціалізується на проведенні верифікованих розіграшів і акцій, зокрема у сфері криптовалют. Платформа позиціонує себе як майданчик для «верифікованих криптовалютних розіграшів та аірдропів», де користувачі можуть безпечно отримувати криптовалютні винагороди. Основною особливістю сервісу є акцент на забезпеченні прозорості та довіри між організаторами розіграшів і їхніми учасниками.

Специфіка функціонування платформи полягає у створенні контрольованого середовища для проведення різноманітних конкурсів, лотерей та промоакцій. На відміну від традиційних сервісів, Giveaway.com зосереджується на верифікації організаторів і прозорості процесу визначення

переможців. Це особливо актуально в контексті криптовалютних розіграшів, де часто виникають сумніви щодо справедливості та легітимності проведення акцій.

Платформа обслуговує широкий спектр користувачів – від індивідуальних блогерів до масштабних криптовалютних проєктів і традиційних брендів. На рисунку 1.4 зображена частина готових шаблонів для запуску розіграшів та проведення маркетингових кампаній. Організатори мають можливість створювати різноманітні типи розіграшів: від простих лотерей із фіксованими призами до багаторівневих конкурсів зі складними умовами участі.

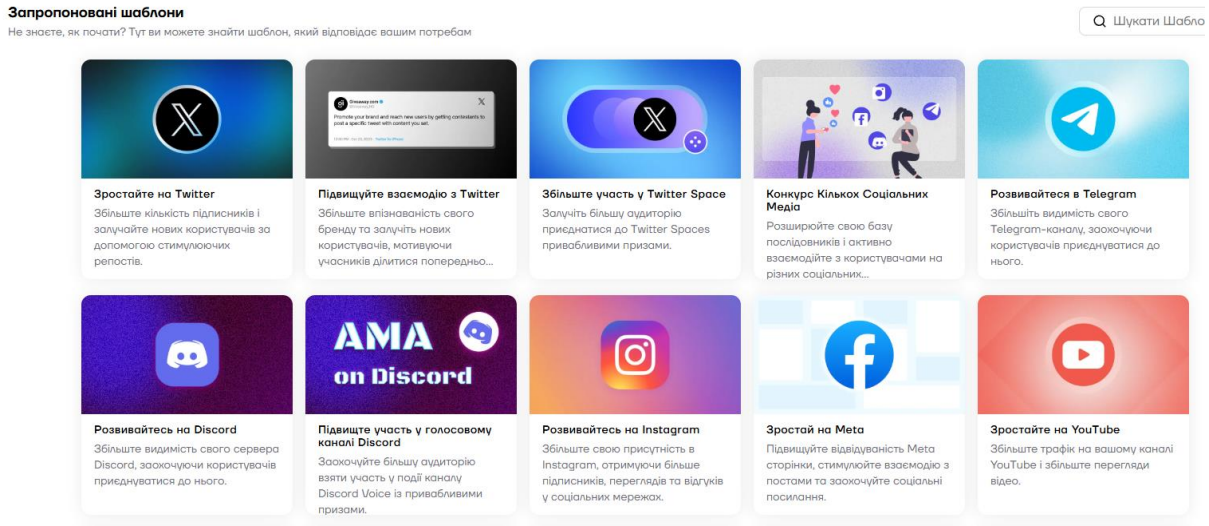


Рисунок 1.4 – Частина готових шаблонів для розіграшів та промоакцій

Функціональні можливості платформи досить різноманітні та включають:

– створення кастомізованих розіграшів із гнучкими параметрами;

– інтеграцію з популярними соціальними мережами для автоматичної перевірки виконання умов участі;

– систему багаторівневих завдань і бонусних балів;

– можливість проведення миттєвих розіграшів та запланованих кампаній;

– інструменти для відстеження статистики та аналітики ефективності;

– розширені можливості для персоналізації інтерфейсу розіграшів.

Учасники, у свою чергу, отримують доступ до верифікованих акцій із мінімальним ризиком участі у шахрайських схемах. На рис. 1.5 наведено систему привілеїв, що стимулює верифікацію проєктів на платформі Giveaway.com. Ця система передбачає надання певних бонусів як інструменту мотивації користувачів до проходження верифікації власних проєктів. Таким чином створюються додаткові переваги, що підвищують довіру до платформи та сприяють залученню активних партнерів.

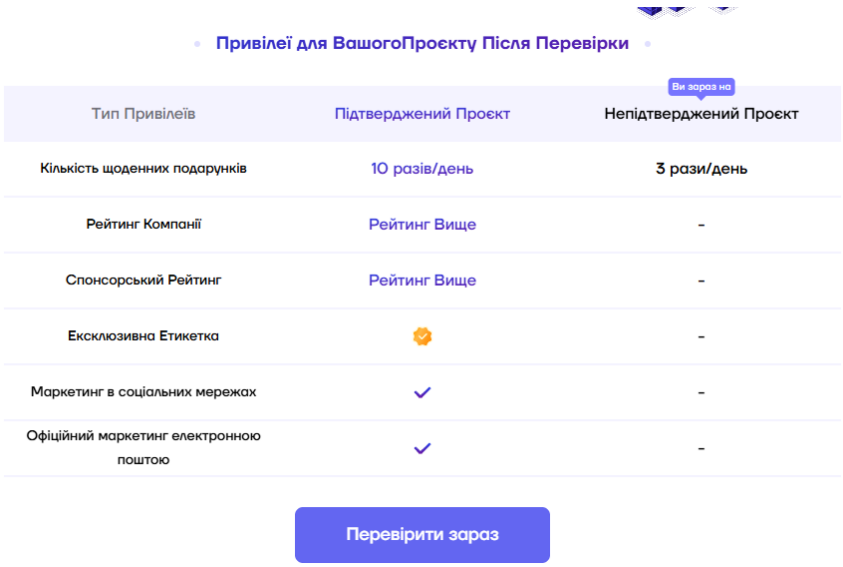


Рисунок 1.5 – Система привілеїв Giveaway.com

Важливим аспектом функціонування платформи є розгалужена система інтеграцій із соціальними мережами, блокчейн-проєктами та цифровими екосистемами. На рис. 1.6 зображено платформи, які доступні для інтеграції з Giveaway.com, що дозволяє автоматизувати перевірку виконання умов участі в розіграшах. Крім того, платформа активно співпрацює з криптовалютними проєктами, DeFi-протоколами та NFT-колекціями, надаючи їм інструменти для проведення маркетингових кампаній і залучення спільноти.

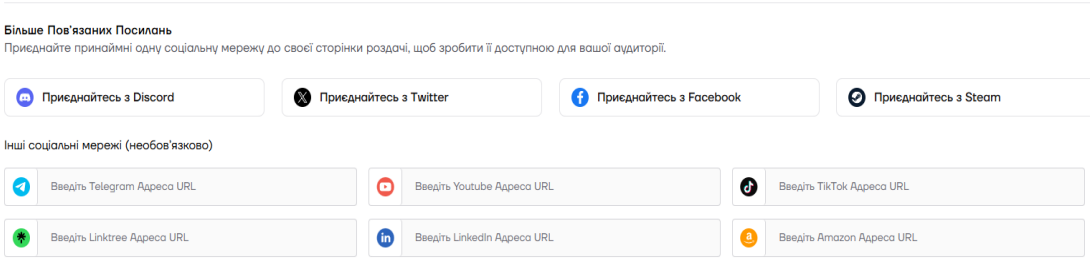


Рисунок 1.6 – Платформи, доступні для інтеграції з Giveaway.com

Переваги спонсорства та проведення розіграшів, зазначені компанією:

1. Сприяння формуванню спільноти для впровадження інновацій і налагодження мережевих зв'язків. Розіграші можуть створити відчуття єдності навколо вашого бренду або продукту, заохочуючи учасників ділитися власним досвідом і взаємодіяти з однодумцями. Це, своєю чергою, стимулює обмін ідеями, впровадження нововведень і збір зворотного зв'язку.

2. Індивідуалізоване відстеження рентабельності інвестицій (ROI). Розіграші дозволяють ефективно оцінювати ROI шляхом аналізу таких показників, як рівень залученості, трафік і конверсії. Ці дані можна адаптувати до потреб конкретного бізнесу, що сприяє точнішому вимірюванню ефективності кампаній.

3. Персоналізований та цілеспрямований досвід для аудиторії. Формат розіграшів можна адаптувати до інтересів і вподобань цільової аудиторії, створюючи релевантний та емоційно значущий досвід. Це сприяє поглибленню стосунків із клієнтами та підписниками.

4. Винагородження та визнання для підвищення лояльності. Нагороджуючи учасників за їхню активність, розіграші можуть значно посилити лояльність до бренду. Позитивний досвід, яким учасники охоче діляться з іншими, сприяє розширенню позитивного іміджу компанії.

5. Залучення автентичної аудиторії, орієнтованої на бренд. Розіграші сприяють залученню нових підписників та потенційних клієнтів, що дозволяє розширити охоплення. Фокусуючись на автентичних прихильниках бренду, можна забезпечити вищий рівень конверсії та довгострокову лояльність.

Платформа Giveaway.com реалізує універсальний підхід до системи заохочень, забезпечуючи можливість надсилання та отримання винагород як у межах традиційної (Web2), так і децентралізованої (Web3) інфраструктури. Основною метою такої моделі є забезпечення простоти, гнучкості та зручності для кінцевих користувачів.

На рис. 1.7 зображено приклад винагород, що базуються на Web3-технологіях, до яких належать криптовалюти, невзаємозамінні токени (NFT), доступ до whitelist-проєктів тощо. Зазначені інструменти сприяють залученню аудиторії, яка орієнтована на блокчейн-технології та цифрові активи.

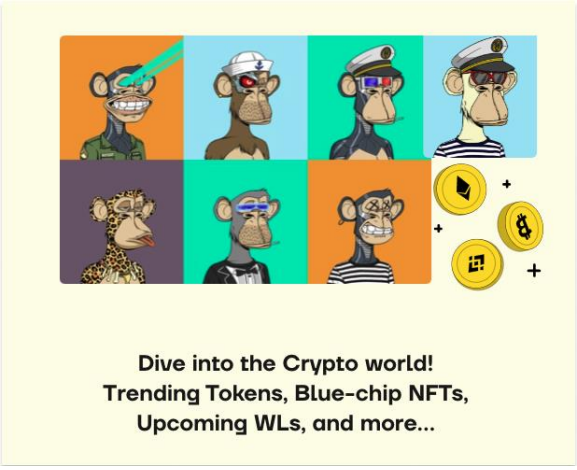


Рисунок 1.7 – Приклад Web3-сегменту винагород

На рис. 1.8 зображено винагороди у межах Web2-сегменту, де користувачі можуть отримувати електронні пристрої (смартфони, ноутбуки, планшети), предмети повсякденного вжитку (одяг, аксесуари, побутову техніку), а також нематеріальні стимули.

Рисунок 1.8 – Приклад Web2-сегменту винагород

Ключовою технологічною перевагою Giveaway.com є використання підходу Provably Fair для підтвердження чесності розіграшів. Застосовуючи цю революційну концепцію, платформа гарантує, що розіграш буде більш реалістичним та прозорим.

Алгоритм Provably Fair на платформі Giveaway.com реалізовано за допомогою трьох основних параметрів:

– Server Seed (Серверне початкове значення): серія цифр і букв, які використовуються для випадкової генерації результатів на основі алгоритму з відкритим вихідним кодом.

– Client seed (Клієнтське початкове значення): Використовується алгоритм на основі блокчейну, який є непередбачуваним. Після початку розіграшу хеш поточного блоку ETH використовується як Client seed.

– Nonce: Активне ціле число, яке збільшується для кожної участі та призначається всім учасникам як їхній унікальний PID (ідентифікатор учасника) для цього розіграшу.

Випадкові числа (для результатів розіграшу) генеруються з використанням серверного початкового значення, клієнтського початкового значення та nonce. Серверне початкове значення хешується і відображається перед початком розіграшу. Це гарантує, що результати розіграшу не можуть бути змінені в процесі розіграшу.

Далі серверне початкове значення шифрується за допомогою HmacSHA256 з секретним ключем, який виступає клієнтським початковим значенням, тобто хешем поточного блоку Etherum, у результаті чого утворюється хеш-значення, яке конвертується в ціле число. Останні 8 цифр цього числа використовуються як початкове значення для псевдовипадкового ГВЧ –лінійного конгруентного методу.

Лінійний конгруентний метод (LCG) – це класичний алгоритм генерації псевдовипадкових чисел, який зображено у формулі[7].

$$X_{n+1} = (aX_n + c)\ mod\ m \quad (1.4)$$

– Xn – поточний стан генератора

– X0 – початкове значення

– Значення glibc – це набір констант a, c, m, які використовуються у відомій реалізації LCG в бібліотеці glibc мови програмування C.

На основі початкового значення X0 випадковим чином визначаються переможці серед учасників, поки не буде обрано всіх. Якщо кількість учасників менша або дорівнює кількості переможців, кожен учасник автоматично отримує приз без застосування алгоритму випадкового вибору.

Таким чином, учасник може перевірити чесність результатів, відтворивши весь алгоритм за допомогою опублікованих значень.

Отже, проаналізувавши платформу Giveaway.com, можна виділити такі переваги та недоліки, що допоможуть у створенні та вдосконаленні системи TrustLuck, враховуючи сильні та слабкі сторони конкурента:

Переваги платформи такі:

1. Застосування алгоритму Provably Fair, що ґрунтується на серверному (Server Seed) та клієнтському (Client Seed) початкових значеннях, а також параметрі Nonce. Такий підхід забезпечує можливість самостійної верифікації результатів розіграшів усіма учасниками.

2. Підтримка автоматичної перевірки виконання умов участі завдяки широкій інтеграції з соціальними мережами. Це значно спрощує організацію та проведення кампаній, роблячи їх більш доступними та ефективними.

3. Платформа надає широкі можливості кастомізації: від базових лотерей до складних багаторівневих конкурсів із бонусними механіками. Завдяки цьому організатори можуть адаптувати сценарії кампаній відповідно до своїх цілей і цільової аудиторії.

4. Інструменти для збору й аналізу даних щодо залученості, трафіку та конверсій дозволяють організаторам комплексно оцінювати ефективність

кожної кампанії, оптимізуючи стратегії взаємодії з аудиторією та розподіл ресурсів.

5. Система привілеїв для верифікованих проєктів що стимулює довіру з боку учасників та надає доступ до додаткових переваг для організаторів, що сприяє залученню перевірених партнерів. Це створює екосистему, орієнтовану на якість та надійність.

Недоліки платформи такі:

1. Значення Client Seed формується на основі хешу поточного блоку Ethereum, а не задається безпосередньо користувачем. Це суперечить самій ідеї клієнтського впливу на генерацію випадкових чисел, знижуючи прозорість та активну участь користувача у процесі.

2. LCG, який використовується платформою для створення псевдовипадкових чисел, не є криптографічно надійним методом. Його слабкі сторони вже були детально проаналізовані й продемонстровані дослідниками, такими як Джим Рідс і Джоан Бояр. Передбачуваність цього методу робить його непридатним для застосування в контексті, де чесність і безпека є критичними – зокрема, у проведенні прозорих розіграшів.

3. Застосування лише останніх восьми цифр хешу як початкового значення для LCG значно зменшує криптографічну ентропію. Це звужує простір можливих комбінацій і підвищує ризик передбачуваності результатів, знижуючи загальний рівень захищеності.

4. Орієнтація на хеші блоків Ethereum для формування Client Seed робить платформу вразливою до технічних збоїв у самій блокчейн-мережі. Також зміни в протоколах чи оновлення стандартів Ethereum можуть вимагати модифікацій з боку платформи, що створює потенційні ризики для стабільної роботи.

5. Повна залежність від внутрішніх інструментів платформи обмежує можливості організаторів у створенні унікальних сценаріїв кампаній. Зокрема, відсутня підтримка незалежної інтеграції сторонніх сервісів для перевірки умов участі або гнучкого налаштування дизайну розіграшів.

6. У разі технічних збоїв або перевантаження серверів при високому трафіку, всю відповідальність за стабільність роботи бере на себе платформа. Це може створювати репутаційні та операційні ризики для організаторів.

7. Одночасна підтримка традиційних (Web2) і децентралізованих (Web3) форматів винагород створює складність у формуванні чіткої цільової аудиторії.

Тому, для розробки та реалізації ефективної системи TrustLuck, на основі виявлених недоліків і переваг, слід врахувати такі ключові аспекти:

1. Система має дозволяти учасникам самостійно впливати на Client Seed, наприклад, через унікальні дані, пов’язані з їхньою участю. Це підвищить прозорість і залученість користувачів у процес визначення переможців.

2. Замість лінійного конгруентного методу (LCG), який є передбачуваним і непридатним для криптографічних цілей, слід застосовувати сучасні криптографічно безпечні алгоритми.

3. Доцільно використовувати повноцінне хеш-значення з високою ентропією для генерації початкового значення.

4. Щоб уникнути ризиків, система має базуватися на внутрішніх криптографічних механізмах, що не залежать від зовнішніх інфраструктур, забезпечуючи стабільність і надійність.

5. Важливо передбачити можливість підключення зовнішніх механізмів перевірки умов участі та впровадження особистого дизайну для розіграшів, щоб розширити гнучкість для організаторів.

РОЗДІЛ 2 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ TRUSTLUCK

2.1 Функціональні та нефункціональні вимоги

Метою даної роботи є розробка системи для проведення чесних, прозорих та безпечних розіграшів із використанням криптографічних методів, сучасних веб-технологій і механізмів автоматизації. Для досягнення цієї мети необхідно чітко визначити функціональні та нефункціональні вимоги, врахувати потреби цільової аудиторії, а також забезпечити відповідність системи сучасним стандартам криптографічної безпеки та автоматизації. У цьому підрозділі представлено аналіз вимог до системи TrustLuck, включаючи її функціональні та нефункціональні потреби.

До системи, яка розробляється, представлені такі функціональні вимоги:

1. Механізм інтеграції системи у сторонні вебресурси. З метою надання організаторам можливості налаштовувати індивідуальні умови участі, а також створювати унікальний дизайн сторінки розіграшу, пропонується реалізувати систему у форматі модального вікна, яке легко інтегрується у будь-який сторонній вебресурс. Інтеграція здійснюється за допомогою підключення скрипту, що містить логіку управління розіграшем. Ініціація модального вікна відбувається шляхом виклику функції з передачею єдиного параметра – ідентифікатора розіграшу. У результаті на вебсторінці організатора відкривається інтерактивне модальне вікно з вмістом конкретного розіграшу, що значно спрощує процес впровадження функціоналу на зовнішні ресурси.

2. Створення та управління розіграшами. Система має надавати організаторам функціонал для створення розіграшів із налаштуванням таких параметрів, як: назва, тривалість розіграшу, максимальна кількість учасників, кількість переможців та опис призів. Адміністративний інтерфейс повинен підтримувати редагування, завершення або видалення розіграшів.

3. Автоматизація переходу між етапами. Це стосується автоматичного переходу системи з одного стану до іншого

4. Верифікація учасників через email. З метою мінімізації участі автоматизованих облікових записів (ботів) у розіграші, пропонується впровадити механізм обов’язкової верифікації електронної пошти користувача. Для підтвердження реєстрації учаснику надсилається унікальний шестизначний код, який необхідно ввести у відповідному полі. Такий підхід сприяє підвищенню достовірності даних користувачів.

5. Прозорий та безпечний механізм вибору переможців. Пропонується використати підхід Provably Fair, розглянутий в 1 першому розділі, який дозволить учасникам верифікувати результати розіграшу. Початкове серверне значення (Server Seed) доцільно формувати за допомогою CSPRNG з додаванням унікальних параметрів, таких як ідентифікатор розіграшу та мітка часу його створення. Це дозволить гарантувати унікальність кожного розіграшу та унеможливить передбачення результатів. Початкове клієнтське значення (Client Seed) рекомендується генерувати на основі агрегованих даних учасників (наприклад, електронних адрес), з подальшим обчисленням хешу від списку всієї бази. Такий підхід мінімізує ризики зовнішнього втручання або маніпуляцій з боку організаторів, забезпечивши незалежність процесу.

6. Оновлення даних у реальному часі. Система має відображати актуальну інформацію про розіграш (кількість учасників, час до завершення) без необхідності оновлення сторінки.

7. Автоматизація повідомлень щодо виграшних призів. Після завершення розіграшу система має автоматично визначати переможців, надсилати їм email-повідомлення з описом призу або інструкціями щодо подальших дій.

Перелік нефункціональних вимог до системи описано у таблиці 2.1.

Таблиця 2.1 – Нефункціональні вимоги до системи TrustLuck

Категорія	Опис вимог
Продуктивність	Час відповіді на запити до API не повинен перевищувати 1 секунди при навантаженні до 500 одночасних користувачів.
	Оновлення кількості учасників у розіграші має здійснюватися в режимі реального часу із затримкою не більше 1 секунди при 1000 активних підключень.
	Процес вибору переможців повинен виконуватися не довше ніж за 3 секунди для розіграшів з кількістю учасників до 5000.
Безпека	Кожен email може бути зареєстрований у розіграші лише один раз.
	Повинен бути впроваджений механізм доведеної чесності (Provable Fairness) для перевірки достовірності результатів.
	Необхідно обмежити кількість запитів до API з одного джерела (rate limiting) для захисту від зловмисного навантаження.
Масштабованість	Платформа повинна підтримувати розіграші до 10 000 учасників без значного зниження продуктивності.
	Структура таблиць у MySQL має бути оптимізована для швидкої перевірки реєстрації та пошуку учасників.
	Інформація про активні розіграші повинна зберігатися в оперативній пам’яті для забезпечення високої швидкодії.
Користувацький досвід	Інтерфейс повинен коректно відображатися на мобільних пристроях (від 360x640 пікселів) та на десктопах (до 1920x1080 пікселів).
Користувацький досвід	Повідомлення про помилки мають бути зрозумілими, українською мовою, навіть для некваліфікованих користувачів.
	Усі елементи інтерфейсу мають бути локалізовані українською мовою.
Технічне обслуговування	Необхідно вести журнал основних дій (створення розіграшу, реєстрація учасників, вибір переможців) з точним часом.
	Має зберігатися історія розіграшів із кількістю учасників для

	подальшого аналізу та статистики.

Враховуючи сформульовані функціональні та нефункціональні вимоги до програмного продукту, наступним кроком є розробка його архітектури. На цьому етапі буде детально представлено структуру системи, визначено її основні компоненти, описано схеми взаємодії між ними, а також обґрунтовано вибір відповідних технологій. Проєктування архітектури здійснюватиметься із застосуванням методу системного моделювання, який дозволить комплексно дослідити структуру та функціонування системи.

2.2 Архітектура системи TrustLuck

При побудові архітектури системи TrustLuck було використано архітектурний шаблон програмування MVC (Model-View-Controller), оскільки він забезпечує чітке розмежування логіки, гнучкість у налаштуванні інтерфейсу, масштабованість та безпеку, що є ключовими аспектами для реалізації вимог до системи. У процесі проєктування також було застосовано контекстну діаграму, яка відображає основні зовнішні взаємодії та потоки даних, а також діаграму архітектури MVC, що деталізує внутрішню структуру системи та взаємозв’язки її основних модулів. Зазначені діаграми надають комплексне уявлення про архітектуру системи та підтверджують її відповідність встановленим вимогам.

Контекстна діаграма називається найвищим рівнем діаграми потоку даних. Це техніка, яка використовується бізнес-аналітиками, щоб допомогти зрозуміти деталі та межі проекту або системи, яка має бути розроблена. Крім того, цей візуальний посібник відображає потік інформації між зовнішніми компонентами та системою[3].

На рис. 2.1 зображена контекстна діаграма, яка ілюструє взаємодію системи TrustLuck з користувачем, адміністратором та службою електронної пошти, відображаючи структуру та напрямки інформаційної взаємодії між

учасниками системи. Вона забезпечує чітке розуміння ролі системи TrustLuck у загальній інфраструктурі.

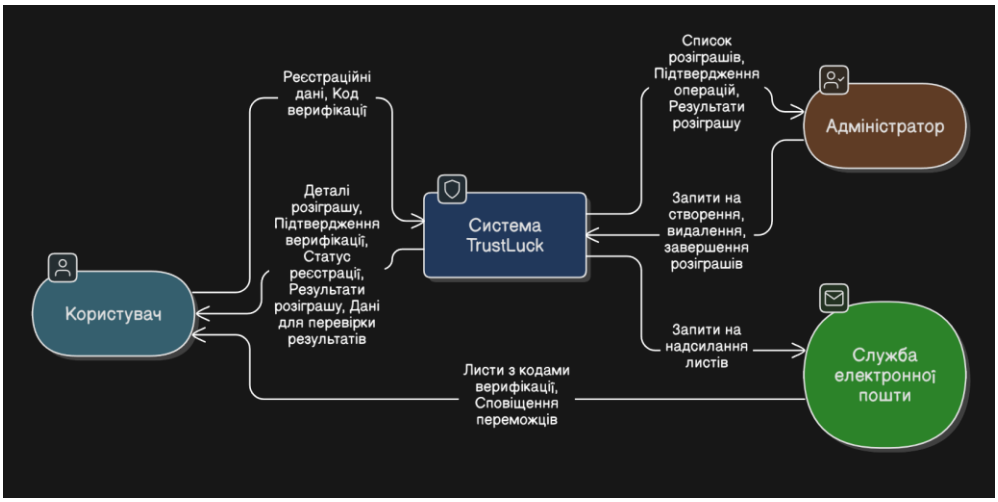


Рисунок 2.1 – Контекстна діаграма до системи TrustLuck

Контекстна діаграма (див. рис. 2.1) відображає систему як єдиний центральний процес, який взаємодіє з трьома зовнішніми сутностями:

– користувач (учасник): основний актор, який взаємодіє із системою TrustLuck через вебінтерфейс з метою участі в розіграшах;

– адміністратор: відповідає за керування розіграшами через адміністративний інтерфейс, надсилаючи відповідні запити до системи;

– служба електронної пошти: виконує функцію комунікації з користувачами, надсилаючи відповідні листи на електронні адреси учасників на основі запитів від системи.

Потоки даних відображають інформаційний обмін між системою TrustLuck та її зовнішніми сутностями, забезпечуючи функціональність платформи. Основні потоки включають:

– взаємодію з користувачем: користувачі надають реєстраційні дані (електронну пошту) та коди верифікації для підтвердження участі в розіграшах. Система надає деталі розіграшу, підтвердження верифікації, статус реєстрації, результати розіграшу та дані для перевірки результатів;

– взаємодію з адміністратором: адміністратори надсилають запити на створення, видалення або завершення розіграшів із відповідними параметрами. Система повертає список розіграшів з вказаними параметрами, підтвердження операцій і результати розіграшу;

– взаємодію зі службою електронної пошти: система надсилає запити на відправлення листів для верифікації, генеруючи унікальні коди для кожного учасника, та сповіщення для переможців;

– комунікацію служби електронної пошти з користувачем: служба надсилає користувачам листи з кодами верифікації для підтвердження участі та сповіщення переможців з деталями розіграшу та інструкціями щодо отримання призів.

Отже, контекстна діаграма підтверджує, що архітектура системи TrustLuck узгоджується з ключовими вимогами щодо створення та управління розіграшами, верифікації учасників та встановлення комунікації з користувачами через автоматизацію повідомлень за допомогою зовнішньої служби електронної пошти. Вона чітко відображає взаємодію системи з зовнішніми сутностями – Користувачем, Адміністратором та Службою електронної пошти, забезпечуючи підтримку основних процесів у системі.

Для забезпечення повної відповідності системи TrustLuck зазначеним функціональним та нефункціональним вимогам, необхідно визначити її архітектурну побудову, ключові компоненти та їх взаємозв’язки, що реалізуються відповідно до шаблону MVC (Model-View-Controller).

MVC (Model-View-Controller) – це архітектурний шаблон для розробки програмного забезпечення, який допомагає розділити програму на три основні частини[4], які зображені на рис. 2.2.

Три основні компоненти MVC представлені як:

– MODEL, що відповідає за обробку даних та бізнес-логіку програми. Він зберігає інформацію і визначає, як ця інформація обробляється і змінюється.

– VIEW, що відповідає за інтерфейс користувача та представлення даних. Він представляє інформацію з MODEL у зручному для користувача вигляді.

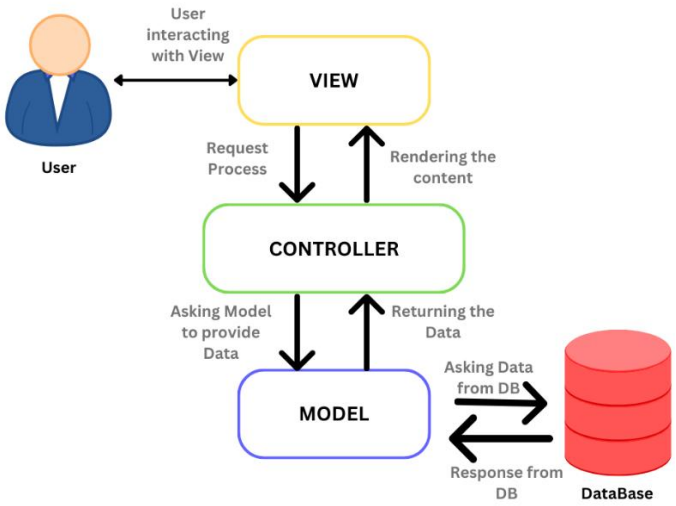


Рисунок 2.2 – Архітектурний шаблон MVC

– CONTROLLER, що взаємодіє з користувачем і відправляє команди до MODEL та VIEW. Він контролює потік інформації між MODEL і VIEW та реагує на дії користувача.

MVC перетворює складну розробку застосунків на більш керований процес. Сьогодні цей патерн використовується в багатьох сучасних вебзастосунках, оскільки він дозволяє робити додаток масштабованим, підтримуваним і легко розширюваним.

Нижче наведено покроковий опис процесу взаємодії користувача з додатком, реалізованого згідно з архітектурним шаблоном MVC:

1. взаємодія користувача через VIEW: початком є етап взаємодії користувача з додатком, як правило, через користувацький інтерфейс, представлений у VIEW. Ця взаємодія може бути такою ж простою, як заповнення форми або натискання кнопки, а може включати більш складні дії. VIEW фіксує ці дії користувача і надсилає запит до CONTROLLER;

2. CONTROLLER – мозок операцій: CONTROLLER, який часто розглядається як мозок шаблону MVC, перехоплює ці запити і визначає, як вони повинні бути оброблені. Він враховує логіку та бізнес-правила додатку;

3. отримання даних з MODEL: CONTROLLER запитує дані у MODEL, вказуючи, що йому потрібно. MODEL відповідає за цілісність даних та формулює точні запити до бази даних і зв'язується з сервером, надсилаючи їх;

4. база даних відповідає на запити: сервер бази даних починає діяти, обробляє запити і отримує дані. Ці дані обережно укомплектовуються та надсилаються назад до MODEL у вигляді набору результатів;

5. MODEL готує дані: MODEL обробляє дані, забезпечуючи їх відповідність структурі додатку. Потім він передає дані до CONTROLLER;

6. CONTROLLER приймає рішення: CONTROLLER, відповідно до отриманих даних від MODEL, приймає рішення на основі дій користувача і логіки програми;

7. представлення даних: коли дані призначені для відображення або представлення користувачеві, CONTROLLER співпрацює з VIEW. VIEW використовує ці дані для відповідного відображення у користувацькому інтерфейсі.

Щоб реалізувати взаємодію користувача з системою TrustLuck з метою участі в розіграшах необхідно розробити компоненти, що будуть контролювати процес реєстрації, верифікації, оновлення деталей та статусу розіграшу в реальному часі, отримання результатів та даних для їхньої перевірки. Для цього рекомендовано створити контролер розіграшів, який реалізує основну логіку обробки запитів та забезпечить узгоджену взаємодію між зовнішнім інтерфейсом і внутрішніми компонентами, пов’язаних із участю в розіграші, та модель розіграшу, що включатиме управління даними розіграшів і учасників та забезпечить взаємодію бази даних та контролера для коректної обробки запитів. Такий Контролер слугуватиме проміжною ланкою між користувацьким інтерфейсом (Виглядом) і базою даних, забезпечуючи коректне виконання

основних операцій, тоді як Вигляд відповідатиме за відображення даних користувачу та передачу його дій до Контролера. Зокрема, контролер розіграшів повинен:

– надсилати інформацію про розіграш – використовувати метод, що надаватиме клієнту актуальні дані щодо розіграшу: назву, дату та час до завершення, максимальну кількість учасників та поточну кількість зареєстрованих учасників.

– надсилати код підтвердження на електронну адресу – використовувати метод, що генеруватиме випадковий код, який зберігатиметься в пам'яті сервера з обмеженим часом дії. Після чого цей код відправлятиметься користувачу на електронну пошту через поштовий сервіс. Це допоможе мінімізувати участь автоматизованих облікових записів (ботів).

– підтверджувати код верифікації – використовувати метод, що перевірятиме код, введений користувачем. Якщо код ще дійсний, електронна адреса додається до списку учасників у відповідній таблиці розіграшу в базі даних системи.

– перевіряти реєстрацію електронної адреси – використовувати метод, що дозволить визначити, чи вже зареєстрована конкретна електронна адреса для певного розіграшу, щоб уникнути дублювання участі.

Для забезпечення контрольованого доступу до логіки розіграшів та захисту системи від потенційних зловживань, пов’язаних із розіграшем, доцільно створити маршрутизатор, який визначатиме відповідні шляхи (ендпоінти) API, зіставлятиме їх з логікою контролера, а також застосовуватиме обмеження частоти запитів (rate limiting) з метою запобігання зловживанням і атакам. Це дозволить забезпечити структуровану та безпечну взаємодію між клієнтською частиною та сервером, а також спростить підтримку та масштабування функціональності системи.

Щоб забезпечити ефективне керування розіграшами зі сторони адміністратора, необхідно розробити окремі компоненти, які реалізовуватимуть логіку створення, завершення, видалення та перегляду розіграшів. Для цього

рекомендовано створити контролер адміністрування, який відповідатиме за обробку службових запитів, пов’язаних із управлінням розіграшами, а також використовувати наявну модель розіграшу з додаванням відповідних запитів до бази даних для забезпечення повноцінної взаємодії під час адміністрування. Зокрема, контролер адміністрування повинен:

– створювати нові розіграші – реалізувати метод, що прийматиме параметри розіграшу (назва, тривалість, кількість переможців, максимальна кількість учасників, опис призів), обчислюватиме кінцеву дату завершення, генеруватиме Server Seed та його хеш для публікації перед початком розіграшу, а також ініціалізуватиме таймер до завершення.

– отримувати перелік усіх розіграшів – реалізувати метод, що повертатиме список наявних розіграшів разом з збереженими параметрами, розрахованим часом завершення та кількістю учасників. Це дозволить адміністраторам моніторити стан кожного активного або завершеного розіграшу.

– видаляти розіграш – реалізувати метод, що дозволить повністю видалити розіграш з бази даних, а також видаляти відповідний таймер, якщо він ще активний.

– завершувати розіграш достроково – реалізувати метод, який примусово завершуватиме розіграш, фіксуючи поточних учасників та запускаючи механізм вибору переможців на основі попередньо згенерованого Server Seed та випадкового Client Seed, зберігаючи результати у базі даних та повідомляючи переможців шляхом надсилання відповідних листів з інструкціями щодо отримання призів

Для керування адміністративною частиною платформи необхідно створити окремий маршрутизатор, який визначатиме шляхи API та спрямовуватиме запити до відповідних методів контролера. Використання окремих контролерів і маршрутизаторів для користувацької та адміністративної частин забезпечить чітке розмежування основного функціоналу платформи і службової логіки, що використовується адміністраторами.

Оскільки адміністративний інтерфейс не передбачений для публічного доступу, доцільно використовувати його виключно в локальному середовищі або в умовах обмеженого доступу. Такий підхід дозволяє мінімізувати ризики несанкціонованого втручання та витоку конфіденційних даних.

Для додаткового захисту в реалізації адміністративного маршрутизатора слід застосувати базову HTTP-автентифікацію (basic authentication), яка дозволить обмежити доступ до маршрутизатора лише авторизованим користувачам. Це рішення забезпечить мінімальний, але необхідний рівень безпеки під час розробки та тестування системи, водночас залишаючи потенціал для впровадження більш складних механізмів автентифікації.

Користувачі системи TrustLuck мають постійно отримувати актуальну інформацію щодо деталей розіграшу. Для цього необхідно розробити два додаткові компоненти: менеджер таймерів і модуль WebSocket, які забезпечуватимуть автоматичне завершення розіграшів за допомогою таймера з відліком часу до завершення і динамічне оновлення даних про кількість учасників.

Менеджер таймерів відповідатиме за створення та окремого керування таймерами для кожного розіграшу. Під час створення розіграшу цей компонент обчислюватиме час до його завершення на основі заданої тривалості, після чого запускатиме єдиний таймер для всіх учасників на сервері. Для оптимізації рекомендовано обробляти таймер на стороні користувача, отримуючи актуальний час від сервера під час завантаження інтерфейсу. Таймер працюватиме в пам’яті сервера, незалежно від клієнтських підключень, забезпечуючи надійне завершення розіграшу у визначений час. Після закінчення таймера менеджер викликатиме функцію, яка ініціюватиме вибір переможців і оновлюватиме статус розіграшу, а також видалятиме таймер з серверу, враховуючи можливість завершити/видалити розіграш з панелі адміністратора.

Модуль WebSocket забезпечуватиме передачу оновлень у режимі реального часу від сервера до користувацького інтерфейсу. Після кожної нової

реєстрації учасника чи зміни статусу розіграшу користувачам надсилатимуться актуальні дані, дозволяючи динамічно оновлювати інтерфейс без перезавантаження сторінки.

Щоб забезпечити можливість верифікації користувачів та сповіщень переможців необхідно налаштувати комунікацію між сервером та клієнтом, що можна реалізувати за допомогою служби електронної пошти. Цей компонент надсилатиме верифікаційні коди для підтвердження реєстрації учасників, а також сповіщення переможців, включаючи інформацію про розіграш та інструкції щодо отримання призів.

Для визначення результатів розіграшу необхідно розробити компонент який використовуватиме алгоритм на основі підходу Provably Fair, механізму автентифікації повідомлень HMAC та криптографічно безпечного генератора випадкових чисел. Після завершення розіграшу, ініційованого менеджером таймерів або адміністративним керуванням, служба вибору переможців комбінуватиме попередньо згенерований Server Seed із Client Seed та генеруватиме унікальні значення для кожного учасника. Алгоритм обиратиме переможців зі списку учасників, отриманого з бази даних. Результати зберігатимуться в базі даних, а дані для перевірки публікуватимуться відразу після завершення розіграшу для незалежної перевірки користувачами.

База даних слугуватиме центральним сховищем для всіх даних системи TrustLuck, забезпечуючи їх цілісність і доступність. Вона зберігатиме інформацію про деталі розіграшів, учасників та результати розіграшів. Використовуючи реляційну базу даних, система підтримуватиме швидкі запити для читання та запису, які будуть зазначені в моделі розіграшів та використовуватимуться в контролерах.

Для дотримання однієї з ключових вимог щодо забезпечення прозорості та чесності системи TrustLuck необхідно створити механізм незалежної перевірки результатів розіграшу, який частково або повністю відтворюватиме алгоритм, залежно від вимог щодо конфіденційності даних. Оскільки платформа гарантує захист персональних даних, слід реалізувати алгоритм,

який не розкриватиме електронні адреси інших користувачів. З цією метою рекомендується використовувати хешування даних користувачів за допомогою хеш-функції SHA-256 та публікувати після завершення розіграшу відповідний хеш, а також, згенеровані службою вибору переможців - Server Seed, Client Seed і їхні хеші. На основі цих параметрів користувачі зможуть самостійно перевірити: яке місце вони посіли, яке число було згенероване для їхньої електронної адреси, який поріг був встановлений для перемоги, а також свій статус – переможець чи ні.

На рис. 2.3 зображено діаграму архітектури веб-додатку TrustLuck, що відповідає зазначеним компонентам та відображає їх взаємозв'язки.

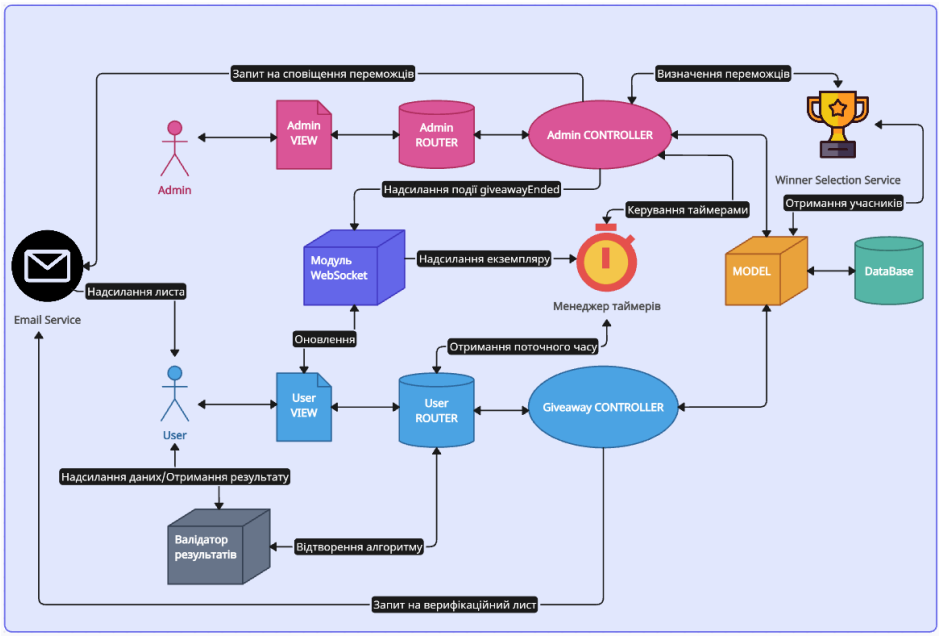


Рисунок 2.3 – Діаграма архітектури веб-додатку TrustLuck

Діаграма відображає архітектуру на основі єдиного серверу, де взаємодія між компонентами структурована через контролери та маршрутизатори. Користувачі працюють з інтерфейсом User VIEW, який взаємодіє з системою через User ROUTER, який передає запити до Giveaway CONTROLLER. Цей контролер реалізує логіку надсилання параметрів розіграшу для оновлення інтерфейсу, генерацію та зберігання верифікаційних кодів з встановленим

часом дії, повторну відправку та перевірку кодів, додавання користувача до учасників, перевірку на дублювання електронної адреси, надання даних для перевірки результатів, повернення списку переможців, отримання поточного часу таймеру та обчислення часу до завершення. Для адміністрування (Admin VIEW) передбачено окремий інтерфейс Admin VIEW, Admin ROUTER та Admin CONTROLLER, що забезпечують можливість створення, завершення та видалення розіграшів адміністратором, а також ініціюють вибір переможців через Winner Selection Service. WebSocket забезпечує оновлення інтерфейсу користувача в реальному часі, а Timer Manager контролює логіку керування таймерами окремо для кожного розіграшу. Email Service інтегрується для верифікації та сповіщень переможців, а логіка вибору переможців базується на підході Provably Fair з використанням HMAC та SHA-256. Валідатор результатів забезпечує можливість перевірки результатів розіграшу шляхом відтворення алгоритму вибору переможців із використанням параметрів, наданих користувачем.

На таблиці 2.2 зображено перелік основних компонентів системи, їх призначення та взаємодію з іншими елементами архітектури.

Таблиця 2.2 – Основні компоненти системи

Компонент	Призначення	Взаємодія з іншими компонентами
1	2	3
User View	Відображає користувацький інтерфейс з розіграшами та передає дії користувача	Передає запити від User до User Router; отримує дані від User Router або оновлення від WebSocket та оновлює інтерфейс
User Router	Обробляє користувацькі API-запити, захищає від надмірних запитів	Оброблює запити від User View та валідатора результатів, отримує поточний час від менеджера таймерів через Giveaway Controller
Giveaway Controller	Обробляє дії користувача - керує реєстрацією та верифікацією, надсилає деталі розіграшу, отримує останнє значення з таймера та список переможців	Взаємодіє з модулями з Model; оброблює запити від User Routes; надсилає запит на надсилання верифікаційного листа в Email Service
Admin View	Надає інтерфейс для керування розіграшами	Передає запити від Admin до Admin Router; отримує дані від Admin Router та оновлює інтерфейс.
Admin	Обробляє адмінські API-	Оброблює запити від Admin View через Admin

Router	запити, виконує автентифікацію	Controller та повертає дані до Admin View

Продовження таблиці 2.2

1	2	3
Admin Controller	Керує розіграшами, ініціює вибір переможців, оброблює кінець розіграшу	Отримує запити від Admin Router; передає їх до Model; ініціює вибір переможців через Winner Selection Service; надсилає листи через Email Service; генерує події для WebSocket; керує логікою таймерів разом з менеджером таймерів
Model	Модуль роботи з розіграшами, який забезпечує повний набір функцій для створення та управління розіграшами і їх учасниками	Отримує запити від Giveaway Controller та Admin Controller, взаємодіє з Database
Database	Зберігає дані про розіграші та учасників	Отримує запити від Model, зберігає і надає дані
Модуль WebSocket	Забезпечує оновлення в реальному часі	Отримує події від Admin Controller, надсилає оновлення до User View та екземпляр об'єкта з даними розіграшу до менеджера таймерів.
Менеджер таймерів	Відповідає за керування таймерами та ініціює завершення розіграшу після спливу встановленого часу.	Взаємодіє з User Router надсилаючи поточне значення часу в таймері, отримує екземпляр об'єкту з WebSocket та надсилає його в Admin Controller при завершенні таймеру, керує логікою таймерів з Admin Controller
Winner Selection Service	Обирає переможців за чесним алгоритмом	Отримує запити від Admin Controller; отримує список учасників з Model; обчислює переможців та повертає результат в Admin Controller
Email Service	Надсилає коди та сповіщення	Отримує запити від Giveaway Controller, Admin Controller; надсилає листи до User
Валідатор результатів	Відтворює алгоритм вибору переможців із подальшою публікацією даних щодо зайнятого місця, згенерованого числа, мінімального порогового значення для перемоги та статусу	Надсилає запити до User Router; отримує дані від User Router; отримує дані від User; надсилає дані до User
User	Користувач, після проходження етапу верифікації – учасник розіграшу	Отримує листи від Email Service; надсилає дані до валідатора; отримує дані від валідатора; взаємодіє з User View

2.3 Вибір інструментів та технологій

Для створення платформи TrustLuck було обрано набір сучасних інструментів і технологій, що відповідають вимогам масштабованості, безпеки, прозорості та зручності взаємодії для користувачів і адміністраторів. Вибір технологічного стеку визначався їхньою здатністю забезпечити стабільну роботу, захист даних, ефективну обробку запитів і відповідність функціональним та нефункціональним потребам системи, включаючи підтримку інтерактивного інтерфейсу та надійної обробки даних розіграшів.

На серверній стороні обрано Node.js з фреймворком Express.js для реалізації контролерів розіграшів і адміністрування, а також маршрутизаторів. Node.js забезпечує високу продуктивність завдяки асинхронній моделі обробки запитів, що критично для обробки великої кількості одночасних підключень, наприклад, під час оновлення даних про учасників у реальному часі. Express.js спрощує створення RESTful API, забезпечуючи чітке розмежування логіки користувацької та адміністративної частин через модулі giveawayRoutes.js та adminRoutes.js. Для захисту адміністративних ендпоінтів використано express-basic-auth, що реалізує базову HTTP-автентифікацію, достатню для локального або тестового середовища, як зазначено в вимогах.

Для оновлення даних в реальному часі використано Socket.IO у модулі WebSocket. Ця бібліотека забезпечує двосторонню комунікацію між сервером і клієнтом, дозволяючи миттєво оновлювати інтерфейс без перезавантаження сторінки. Socket.IO обрано через його простоту інтеграції з Express.js і підтримку стійких з’єднань, що ідеально підходить для динамічного відображення статусу розіграшу.

Для бази даних обрано MySQL з бібліотекою mysql2/promise, яка підтримує асинхронні запити та транзакції, що є необхідним для реалізації компонента моделі розіграшів. MySQL забезпечує швидкий доступ до даних, надійність транзакцій і ефективне зберігання великих обсягів інформації про

розіграші та учасників. Сервер бази даних розгорнуто за допомогою програмного пакета XAMPP, що забезпечує зручне локальне середовище для розробки та тестування. Використання пулів з'єднань додатково оптимізує використання серверних ресурсів, що є важливим для досягнення масштабованості системи.

Для реалізації служби електронної пошти обрано бібліотеку Nodemailer з інтеграцією поштового сервісу Gmail, щоб забезпечити надійну доставку верифікаційних кодів і сповіщень переможцям. Nodemailer обрано за його гнучкість і підтримку HTML-шаблонів для створення привабливих листів, що підвищує користувацький досвід.

Для генерації Server Seed у Provably Fair обрано метод randomBytes з модуля crypto, оскільки він забезпечує криптографічно стійку, непередбачувану випадковість, що гарантує чесність результатів. Він використовує надійні джерела ентропії ОС і є стандартним інструментом у Node.js для безпечної генерації випадкових чисел.

На клієнтській стороні використано HTML, CSS і JavaScript для створення інтерактивного інтерфейсу. JavaScript забезпечує динамічну обробку подій, що спрощує взаємодію користувача з системою.

Таким чином, обрані технології гарно взаємодіють, забезпечуючи ефективну роботу всіх компонентів системи TrustLuck. Node.js і Express.js формують надійну серверну основу, Socket.IO відповідає за оновлення даних в реальному часі для зручності перегляду, MySQL гарантує цілісність даних, а криптографічні методи та Nodemailer забезпечують прозорість і комунікацію. Цей стек технологій оптимально відповідає вимогам безпеки, масштабованості та користувацького досвіду.

2.4 Механізм вибору переможців на основі підходу Provably Fair

У процесі розробки системи TrustLuck було визначено одну з ключових вимог – забезпечення прозорого, чесного та безпечного механізму визначення

переможців. Для реалізації цієї вимоги було запропоновано використати підхід Provably Fair, який передбачає використання механізму автентифікації повідомлень HMAC та криптографічного генератора випадкових чисел (CSPRNG) для створення Server Seed. Крім того, для забезпечення незалежності та незмінності результатів розіграшу передбачено використання хеш-функції SHA-256: для створення Client Seed шляхом хешування списку електронних адрес учасників, а також для генерації хешу від Server Seed, який публікується до початку розіграшу. У ДОДАТКУ А представлено фрагменти коду компонента winnerService.js, який реалізує описаний підхід вибору переможців відповідно до концепції Provably Fair.

На першому фрагменті зображено імпорт необхідних модулів та реалізацію 3 основних функцій для генерації Client Seed, Server Seed та переможців.

На початку здійснюється імпорт необхідних модулів. Зокрема, використовується вбудований модуль crypto із Node.js, який забезпечує виконання криптографічних операцій, таких як генерація випадкових чисел, хешування з використанням SHA-256, а також створення HMAC-повідомлень. Крім того, імпортується функція getParticipants із модуля giveawayModel.js, що використовується для отримання списку електронних адрес учасників розіграшу.

Функція generateServerSeed відповідає за створення Server Seed. Вона приймає ідентифікатор розіграшу (giveawayId) та дату його створення (createdAt). На першому етапі функція генерує криптографічно безпечну випадкову послідовність з 16 байтів за допомогою crypto.randomBytes, після чого перетворює її у шістнадцятковий формат. Отримана послідовність поєднується з параметрами розіграшу, що забезпечує унікальність ключа. Згенеровані значення логуються разом із часовою міткою відповідно до київського часового поясу.

Client Seed генерується за допомогою функції generateClientSeed, яка приймає масив електронних адрес учасників. Усі адреси об’єднуються в один

рядок, після чого над ним виконується хешування за допомогою хеш-функції SHA-256. Отриманий результат повертається у шістнадцятковому представленні. Таким чином, клієнтський ключ безпосередньо залежить від бази учасників, що виключає можливість модифікації результатів після завершення реєстрації. Як і у випадку з серверними ключами, клієнтські ключі логуються з часовою позначкою.

Функція generateWinners реалізує алгоритм визначення переможців розіграшу. Вона приймає на вхід список електронних адрес учасників, згенеровані раніше Server Seed та Client Seed, а також параметр numWinners, що визначає кількість переможців. Для кожної адреси електронної пошти виконується хешування (із переведенням у нижній регістр) за допомогою алгоритму SHA-256 з метою уніфікації вхідних даних. Далі формується повідомлення у вигляді рядка, що складається з Client Seed та хешу електронної адреси, об'єднаних через двокрапку (clientSeed:emailHash). Це повідомлення хешується за допомогою HMAC-SHA256, де в ролі ключа виступає Server Seed. Отримане HMAC-значення перетворюється на велике ціле число типу BigInt, яке виступає унікальним числовим показником (score) для відповідного учасника. Далі всі учасники впорядковуються за зростанням цього показника. Такий підхід забезпечує детермінований і прозорий процес вибору: за однакових вхідних даних (електронні адреси, Server Seed, Client Seed) результат буде незмінним, що виключає можливість маніпуляцій. Попри те, що алгоритм працює детерміновано, кінцеві числові значення мають ефект криптографічної випадковості. Це досягається шляхом багатокрокової трансформації вхідних даних, що у результаті забезпечує чесну й непередбачувану генерацію результатів, яка при цьому є повністю відтворюваною для перевірки. Після сортування обираються перші numWinners учасників – саме вони визначаються як переможці. Функція повертає масив хешів переможців, а також структуру (мапу), яка зіставляє кожен хеш із відповідною електронною адресою. Усі результати фіксуються у журналі подій із зазначенням часової мітки для забезпечення прозорості та можливості подальшої перевірки.

У другому фрагменті представлена асинхронна функція selectWinners, яка є останнім кроком у механізмі вибору переможців. Вона поєднує генерацію Server Seed, Client Seed та виклик функції generateWinners, формуючи остаточний результат розіграшу.

На першому кроці функція викликає getParticipants для отримання списку електронних адрес учасників відповідного розіграшу. Якщо список виявляється порожнім, функція фіксує це у журналі, після чого повертає об'єкт із порожніми значеннями, що запобігає подальшим обчисленням. У разі наявності учасників відбувається послідовна генерація усіх необхідних компонентів. Спочатку формується Server Seed шляхом виклику функції generateServerSeed, яка приймає ідентифікатор розіграшу (giveawayId) та дату його створення (createdAt). Після цього обчислюється його хеш за допомогою SHA-256, який публікується до початку розіграшу, забезпечуючи незмінність результатів.

Далі викликається функція generateClientSeed, яка на основі списку електронних адрес генерує клієнтський ключ (Client Seed). Його використання гарантує, що результат розіграшу залежить не лише від серверної частини, а й від даних, пов'язаних із учасниками розіграшу. Фінальним етапом є передача згенерованих даних до функції generateWinners, яка реалізує основний алгоритм визначення переможців.

Результатом роботи функції selectWinners є об'єкт, який включає:

– масив хешів переможців (winners);
– серверний ключ (serverSeed);
– хеш серверного ключа (serverSeedHash);
– клієнтський ключ (clientSeed);
– мапу відповідностей хешів до електронних адрес (emailMap).

Після завершення всіх обчислень функції selectWinners та generateServerSeed експортуються з модуля для подальшого використання в інших компонентах системи. Зокрема, функція generateServerSeed може використовуватись окремо для попереднього обчислення хешу від Server Seed до початку розіграшу.

2.5 Інтеграція поштової служби для автоматизації повідомлень

З метою забезпечення ефективної комунікації між сервером та кінцевим користувачем вирішено впровадити службу електронної пошти. Вона частково виконує функцію верифікації користувачів шляхом надсилання листів із згенерованими кодами для підтвердження, а також забезпечує інформування переможців розіграшу з детальними інструкціями щодо отримання призів, які визначаються адміністратором під час створення розіграшу.

У ДОДАТКУ Б зображено перший фрагменту коду компоненту emailService.js, який реалізує функціонал створення поштового транспорту та надсилання листів з кодом верифікації. У першому блоці коду здійснюється підключення модулів: nodemailer – для роботи з SMTP-протоколом, та файлу конфігурації, що містить параметри підключення. В якості поштового сервера було обрано сервіс Gmail, який забезпечує відправку до 500 листів на добу. За потреби цей ліміт можна розширити шляхом використання альтернативних сервісів, проте в межах перевірки працездатності відповідного функціоналу наявного обсягу є цілком достатньо.

Основну роль надсилання верифікаційних листів відіграє функція sendVerificationEmail. Вона приймає дві вхідних параметри: електронну адресу користувача та згенерований код підтвердження. Далі формується об’єкт mailOptions, у якому задаються параметри повідомлення: відправник, одержувач, тема листа, текстове та HTML-представлення повідомлення. На рис. 2.4 наведено зовнішній вигляд верифікаційного листа, який має адаптивний дизайн та зручне для сприйняття оформлення, що сприяє покращенню комунікації з користувачем.

Використовуючи раніше налаштований транспорт transport, функція sendVerificationEmail асинхронно виконує надсилання електронного повідомлення. У разі успішного надсилання результат логується в консолі. У випадку виникнення помилки здійснюється її обробка з поверненням об’єкта, що містить відповідне повідомлення про невдачу.

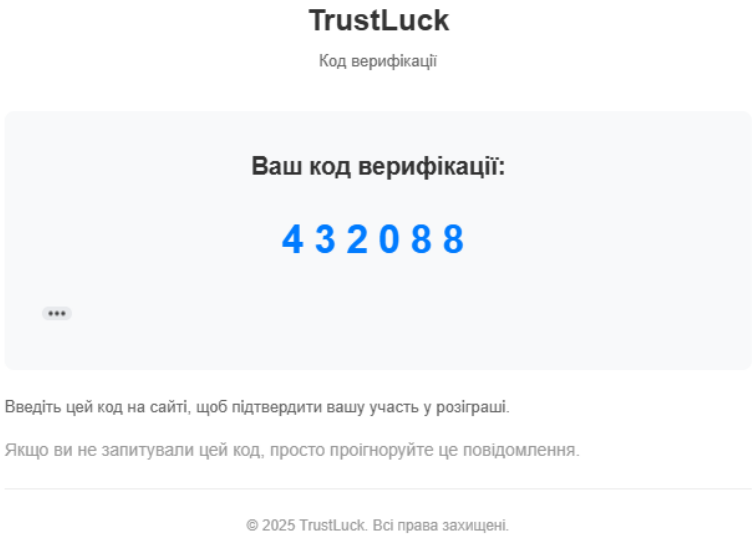


Рисунок 2.4 – Зовнішній вигляд верифікаційного листа

Таким чином, впровадження модуля надсилання верифікаційних листів сприяє ефективній автентифікації користувачів і забезпечує дотримання вимог щодо запобігання участі в розіграші з боку автоматизованих запитів (ботів), що підвищує рівень безпеки та достовірності процесу.

У ДОДАТКУ Б також представлено другий фрагмент коду компоненту emailService.js, який відповідає за інформування учасників щодо перемоги у розіграші. Функція sendWinnerEmail використовується для надсилання персоналізованих повідомлень тим учасникам, які були визначені як переможці.

Вона приймає такі необхідні параметри – електронну адресу отримувача, назву розіграшу (giveawayTitle) та опис призу (prizeDescription). У межах функції формується об’єкт mailOptions, де задаються всі параметри листа: відправник, одержувач, тема, текстова й HTML-версії повідомлення. HTML-шаблон оформлений у єдиному візуальному стилі з верифікаційним листом, проте адаптований до події перемоги. Зовнішній вигляд повідомлення зображено на рис. 2.5. Повідомлення містить привітання, назву розіграшу та

додаткові інструкції щодо отримання призу, які адміністратор задає під час створення розіграшу.

Рисунок 2.5 – Зовнішній вигляд верифікаційного листа

Процес надсилання листа реалізується через метод transport.sendMail, який асинхронно виконує передачу повідомлення через поштовий сервер. Успішні надсилання записуються в консоль, а в разі помилки обробка здійснюється через try-catch з поверненням відповідної інформації про збій.

Наприкінці коду в компоненті emailService.js здійснюється експорт основних функцій – generateVerificationCode, sendVerificationEmail та sendWinnerEmail – з метою їх подальшого використання в інших частинах застосунку, зокрема в контролерах, відповідальних за обробку розіграшів і адміністрування.

2.6 Розробка користувацького інтерфейсу та його функціональної частини

З метою надання організаторам розіграшів можливості гнучкого налаштування індивідуальних умов участі, а також створення унікального дизайну сторінки розіграшу, доцільним є впровадження механізму інтеграції

системи на сторонні веб-ресурси. Для реалізації цього функціоналу пропонується розробити користувацький інтерфейс у форматі модального вікна, що дозволяє вбудовувати його на сторінку розіграшу. Такий підхід забезпечує незалежність зовнішнього оформлення основного веб-ресурсу від логіки та функціональних особливостей самої системи, сприяючи тим самим підвищенню її універсальності, масштабованості та зручності використання. Модальне вікно завантажується на основну сторінку шляхом підключення файлу з кодом, що відповідає за функціонування інтерфейсу користувача, за допомогою HTML-елемента <script> та виклику функції openModal із переданням ідентифікатора розіграшу. Вікно містить 11 етапів взаємодії з користувачем. Опис кожного етапу та його призначення наведено в таблиці 2.3

Таблиця 2.3 – Етапи взаємодії користувача з користувацьким інтерфейсом

Етап	Призначення
1	2
Email Step	Є початковим етапом до моменту досягнення максимальної кількості учасників або завершення розіграшу. На цьому етапі надсилається запит на отримання деталей розіграшу та відображається відповідна інформація користувачеві. Передбачено поле для введення email, прапорці для підтвердження згоди з умовами використання та політикою конфіденційності, а також кнопку «Взяти участь у розіграші», яка надсилає лист верифікації та переводить користувача до наступного етапу – Verification Step.
Verification Step	Використовується для верифікації користувачів та їх залучення до розіграшу. Здійснює перевірку коду верифікації надісланого на попередньому етапі, а також надає можливість повторної відправки коду.
Success Step	Є фінальним етапом до процесу завершення розіграшу. Відображає інформацію про деталі розіграшу та повідомляє користувачів про успішну реєстрацію
Duplicate Step	Відображається у випадку, якщо користувач намагається зареєструватися з електронною адресою, яка вже міститься в таблиці бази даних відповідного розіграшу. Повідомляє про те, що ця електронна адреса вже бере участь у розіграші.
Winner Step	Активується після завершення розіграшу для користувачів, які стали переможцями. Відображає привітання з перемогою та повідомляє про надсилання листа з інструкціями щодо отримання призу.
Loser Step	Активується після завершення розіграшу для зареєстрованих учасників, які не стали переможцями та повідомляє про невдачу.

Продовження таблиці 2.3

1	2
Not Participated Step	Відображається після завершення розіграшу для користувачів, які не брали участі. Повідомляє про відсутність реєстрації в розіграші.
Provable Fairness Success Step	Доступний для відображення під час активного розіграшу на етапах Email та Success. Коротко пояснює принцип роботи механізму Provably Fair та відображає згенерований хеш до Server Seed для перевірки незмінності значення після завершення розіграшу.
Provable Fairness Result Step	Доступний після завершення розіграшу. Надає дані для незалежної перевірки результатів: Server Seed, його хеш, Client Seed та хеш від електронної адреси користувача. Містить посилання на інструмент для перевірки результатів.
Loading Step	Відображається під час завантаження даних розіграшу або після завершення розіграшу, коли система очікує оновлення статусу. Показує анімацію завантаження для інформування користувача щодо обробки запиту.
Max Participants Reached Step	Активується у разі досягнення максимальної кількості учасників. Повідомляє користувача про неможливість участі та відображає деталі розіграшу.

Для того, щоб забезпечити створення, управління та проведення прозорих і безпечних розіграшів із можливістю верифікації результатів, необхідно розробити комплексну систему, яка включає фронтенд і бекенд-компоненти, інфраструктурні елементи та сервіси для обробки даних і комунікації. Система TrustLuck складається з кількох ключових компонентів, які взаємодіють між собою для реалізації функціоналу обробки розіграшів. Нижче наведено детальний опис кожного компонента системи.

Клієнтська частина системи TrustLuck реалізує інтерактивні інтерфейси для користувачів і адміністраторів, забезпечуючи зручну взаємодію з платформою через веб-браузер:

– admin.html: Інтерфейс адміністратора для створення, видалення та завершення розіграшів. Містить форму для введення параметрів розіграшу (назва, тривалість, максимальна кількість учасників, кількість переможців, опис призу) та таблицю для відображення списку активних і завершених розіграшів.

– index.html: Головна сторінка розіграшу, яку можна незалежно змінювати, яка використовує скрипт для підключення модального вікна для взаємодії з системою

– modal.html: Модальне вікно, яке відображає 11 різних етапів взаємодії користувача з розіграшем. Воно динамічно змінює вміст залежно від стану розіграшу та дій користувача.

– modalTemplate.js: JavaScript-модуль, що керує логікою модального вікна. Відповідає за ініціалізацію розіграшу, відображення етапів, обробку подій, взаємодію з сервером через HTTP-запити та WebSocket для оновлення даних у реальному часі, а також управління таймером розіграшу.

Серверна частина системи забезпечує обробку запитів, управління розіграшами, зберігання даних і координацію між клієнтами та сервісами:

– server.js: Центральний серверний модуль, який ініціалізує Express-сервер, налаштовує маршрути для API, WebSocket-з'єднання через Socket.IO та таймери для автоматичного завершення розіграшів. Координує взаємодію між клієнтськими запитами та іншими серверними компонентами.

– giveawayModel.js: Модель даних для роботи з розіграшами в базі даних MySQL. Надає методи для створення розіграшів, додавання учасників, отримання інформації, оновлення, видалення та завершення. Для кожного розіграшу створюється окрема таблиця учасників для ефективного зберігання даних.

– giveawayController.js: Контролер, який обробляє запити користувачів, пов'язані з участю в розіграшах. Містить функції для отримання інформації про розіграш, відправки верифікаційних кодів, перевірки кодів, перевірки статусу реєстрації та верифікації результатів.

– adminController.js: Контролер для адміністративних операцій, таких як створення, видалення та завершення розіграшів. Також відповідає за ініціалізацію таймерів і автоматичне завершення розіграшів після закінчення їхньої тривалості.

– winnerService.js: Сервіс для генерації Server Seed за допомогою CSPRNG та Client Seed використовуючи хеш-функцію SHA-256 і вибору переможців за допомогою криптографічного алгоритму HMAC-SHA256. Забезпечує прозорість і можливість перевірки результатів розіграшу.

– emailService.js: Сервіс для відправки електронних листів через Nodemailer. Генерує верифікаційні коди, відправляє їх учасникам і сповіщає переможців про результати, використовуючи SMTP-сервер поштового сервісу Gmail.

– timerManager.js: Модуль для управління таймерами розіграшів. Відстежує залишковий час і викликає функцію завершення розіграшу після спливу таймера, забезпечуючи автоматизацію процесу.

– timeUtils.js: Утиліта для роботи з часом, яка парсить тривалість розіграшу, обчислює залишковий час і перезапускає таймери для синхронізації з сервером.

Інфраструктурні компоненти забезпечують стабільну роботу системи, зберігання даних і комунікацію в реальному часі:

– MySQL: Реляційна база даних для зберігання даних розіграшів у загальній таблиці giveaways та даних учасників у динамічно створюваних таблицях giveaway_<id>. Забезпечує надійне та структуроване зберігання інформації.

– Socket.IO: Бібліотека для реалізації WebSocket-з’єднань, яка дозволяє оновлювати дані в реальному часі, наприклад, кількість учасників або сповіщення про завершення розіграшу.

– Nodemailer: Бібліотека для відправки електронних листів через SMTP-сервер, що забезпечує надійну доставку верифікаційних кодів і повідомлень переможцям.

– Express: Фреймворк для створення REST API, який обробляє HTTP-запити від клієнтів і забезпечує маршрутизацію до відповідних контролерів.

Ці компоненти разом формують цілісну систему, яка забезпечує створення та управління розіграшами, участь користувачів, верифікацію, оновлення даних у реальному часі, комунікацію між сервером та клієнтом, автоматичне завершення розіграшів та прозору перевірку результатів.

В ході розробки застосовуються такі методи та події, що використовуються на кількох етапах модального вікна або обробляються на

сервері поза етапами, забезпечуючи гнучке керування розіграшами та підтримку користувацького інтерфейсу:

1. Метод updateGiveawayInfo (з компоненту modalTemplate.js) оновлює HTML-елементи інформацією про розіграш, отримуючи дані з об’єкта modalConfig (назва, максимальна та поточна кількість учасників) і змінної serverEndTime, заповнюючи поля #giveawayTitle, #maxParticipants, #currentParticipants і #giveawayDate на етапах Email Step, Success Step і MaxParticipantsEnded Step, відображаючи повідомлення "Час закінчення не визначено" за відсутності часу завершення.

2. Метод startCountdown (з компоненту modalTemplate.js) ініціалізує та оновлює клієнтський таймер зворотного відліку, надсилаючи запит до ендпоінту GET /api/get-time із параметром giveawayId для отримання залишкового часу від таймера який обробляється на сервері у форматі { days, hours, minutes, seconds }, періодично оновлюючи елемент таймеру через встановлення інтервалу на етапах Email Step, Success Step і MaxParticipantsEnded Step, перенаправляючи користувача на етап loading після вичерпання часу.

3. Подія participantUpdate (з компоненту modalTemplate.js) обробляє WebSocket-повідомлення, надіслані сервером через Socket.IO, оновлюючи кількість учасників у реальному часі після кожної нової реєстрації, змінюючи елемент поточних учасників з анімацією fade-in на етапах Email Step, Success Step і MaxParticipantsEnded Step.

4. Метод getGiveawayInfo (з контролеру giveawayController.js) є серверним і повертає через GET /api/giveaway-info детальну інформацію про розіграш (назва, кількість переможців, максимальна та поточна кількість учасників, хеш від Server Seed), отримуючи дані з бази через getGiveaway для відображення на етапах Email Step, Verification Step, Success Step і MaxParticipantsEnded Step, повертаючи JSON або помилку, якщо розіграш не знайдено.

5. Метод getParticipantCount (з моделі giveawayModel.js) повертає поточну кількість учасників для заданого giveawayId із таблиці giveaways у базі даних MySQL, викликається в рамках серверних методів, таких як verifyCode або getGiveawayInfo, і підтримує відображення актуальної кількості учасників на етапах Email Step, Verification Step, Success Step і MaxParticipantsEnded Step.

6. Метод getTime (з контролеру giveawayController.js) обробляє серверний запит GET /api/get-time, повертаючи залишковий час розіграшу через компонент timerManager у форматі { days, hours, minutes, seconds }. Викликається клієнтським startCountdown для запуску таймера.

7. Метод initializeTimers (з компоненту timerManager.js) виконується на сервері під час запуску додатка, ініціалізуючи таймери для всіх активних розіграшів, отримуючи дані з таблиці giveaways і викликаючи startTimer для кожного розіграшу, забезпечуючи синхронізацію серверних таймерів для точного відстеження часу завершення розіграшів.

Для забезпечення участі користувачів у розіграшах на платформі TrustLuck необхідно реалізувати етап Email Step, який є початковим кроком взаємодії користувача з системою. На цьому етапі користувач через модальне вікно вводить свою електронну адресу, яка перевіряється на валідність і унікальність у контексті поточного розіграшу. Якщо адреса ще не зареєстрована і не досягнуто ліміт учасників, система генерує та відправляє шестизначний верифікаційний код на вказану пошту, після чого користувач переходить до етапу Verification Step для введення цього коду. У разі, якщо email уже зареєстрований або перевищено максимальну кількість учасників, відображаються етапи Duplicate Step та Max Participants Reached Step відповідно. Email Step забезпечує безпечну ідентифікацію учасника, запобігає дублюванню реєстрацій і встановлює основу для прозорої участі в розіграші.

Для розробки етапу Email Step було використано такі методи:

1. Метод openModal (з компоненту modalTemplate.js): функція openModal активується при натисканні користувачем кнопки участі в розіграші на головній сторінці, приймаючи ідентифікатор розіграшу (giveawayId) як

параметр. Вона ініціалізує модальне вікно, завантажуючи HTML-вміст із файлу modal.html, приєднує користувача до WebSocket-кімнати розіграшу через Socket.IO для отримання оновлень у реальному часі, і виконує запит до сервера для отримання даних розіграшу, таких як назва, тривалість, максимальна кількість учасників і поточна кількість учасників. Функція також перевіряє статус розіграшу та реєстрацію користувача, викликаючи checkUserRegistration, і визначає, який етап відобразити: Email Step для нових користувачів, Success Step для зареєстрованих або інші етапи (наприклад, MaxParticipantsReached Step, якщо ліміт учасників вичерпано). Викликається GET-запит до маршруту /api/giveaway-info?giveawayId=<id>, який повертає JSON із даними розіграшу, включаючи Server Seed Hash для попереднього перегляду на етапі Provable Fairness Success Step. У результаті модальне вікно відображається з відповідним вмістом, а дані розіграшу заповнюють поля інтерфейсу, готуючи користувача до введення email або переходу до іншого етапу.

2. Метод checkUserRegistration (з компоненту modalTemplate.js): функція checkUserRegistration перевіряє, чи зареєстрований користувач у поточному розіграші, використовуючи хеш електронної адреси, збережений у localStorage. Вона надсилає POST-запит до маршруту /api/check-registration із параметрами emailHash і giveawayId, отримуючи від сервера підтвердження статусу реєстрації. Якщо користувач уже зареєстрований, функція оновлює статус у localStorage і повертає true, що призводить до відображення Success Step. У разі відсутності реєстрації або помилки (наприклад, якщо дані в localStorage застарілі), функція очищає збережені дані та повертає false, дозволяючи відобразити форму Email Step. Цей метод забезпечує коректне визначення стану користувача, запобігаючи повторній реєстрації, і сприяє плавному переходу між етапами. Результатом є точне визначення, чи потрібно показувати форму для введення email, чи перенаправити користувача до іншого етапу.

3. Метод handleFormSubmit (з компоненту modalTemplate.js): функція handleFormSubmit обробляє подію відправки форми з електронною адресою,

викликаючись при натисканні кнопки "Взяти участь у розіграші". Вона перевіряє валідність введеного email за допомогою HTML5-валідатора, а також перевіряє, чи погодився користувач із умовами використання. Якщо перевірки пройдені, функція хешує email за допомогою sha256, надсилає POST-запит до маршруту /api/send-verification із параметрами email, emailHash і giveawayId, і зберігає хеш у localStorage через saveRegistrationData. У разі успішної відправки верифікаційного коду користувач переходить до Verification Step, а також запускається таймер для повторної відправки коду (startResendCooldown). Якщо email уже зареєстрований, відображається Duplicate Step; якщо досягнуто ліміт учасників, відображається Max Participants Reached Step. Результатом є відправка верифікаційного коду на email користувача або інформування щодо проблеми.

4. Метод sendVerification (з контролеру giveawayController.js): Серверна функція sendVerification обробляє POST-запит до маршруту /api/send-verification, отримуючи email, emailHash і giveawayId. Вона перевіряє існування розіграшу через getGiveaway, порівнює поточну кількість учасників із максимальним лімітом за допомогою getParticipantCount і перевіряє унікальність email через isEmailRegistered. Якщо всі умови виконані, функція генерує шестизначний верифікаційний код через generateVerificationCode, зберігає його в пам’яті сервера (у Map verificationCodes) разом із терміном дії (10 хвилин), і викликає sendVerificationEmail для відправки коду користувачу. У разі помилок, таких як перевищення ліміту учасників або повторна реєстрація, повертається JSON із відповідним повідомленням про помилку. Результатом є успішна відправка email із кодом і повернення JSON із підтвердженням успіху, що дозволяє клієнту перейти до Verification Step.

5. Метод sendVerificationEmail (з компоненту emailService.js): Функція sendVerificationEmail формує HTML-лист із шестизначним верифікаційним кодом і відправляє його на вказану електронну адресу через Nodemailer, використовуючи SMTP-сервер сервісу Gmail. Лист містить чіткий дизайн із кодом, інструкціями для введення та попередження про ігнорування листу,

якщо запит не був ініційований користувачем. Функція повертає об’єкт із статусом успіху та messageId (якщо відправка вдала) або повідомленням про помилку. Вона викликається в межах sendVerification і не має прямого HTTP-маршруту. Результатом є доставка верифікаційного коду до поштової скриньки користувача, що дозволяє йому продовжити процес на етапі Verification Step.

6. Метод sha256 (modalTemplate.js): функція sha256 використовує криптографічний API браузера (crypto.subtle) для створення SHA-256 хешу введеної електронної адреси, перетворюючи її в нижній регістр перед хешуванням. Вона застосовується в handleFormSubmit для анонімізації email перед відправкою на сервер, що підвищує конфіденційність і безпеку даних. Хеш передається в запит до /api/send-verification і зберігається в localStorage через saveRegistrationData для подальшого використання в процесі верифікації та перевірки статусу. Результатом є унікальний хеш email, який використовується для ідентифікації учасника без розкриття оригінальної адреси.

7. Метод saveRegistrationData (modalTemplate.js): Функція saveRegistrationData зберігає дані про реєстрацію (emailHash, giveawayTitle, giveawayId, status) у localStorage браузера. Вона викликається після успішного запиту до /api/send-verification у handleFormSubmit, фіксуючи стан "unregistered" до завершення верифікації. Цей метод дозволяє системі запам’ятати інформацію про користувача для коректного відображення етапів при повторному відкритті модального вікна, наприклад, щоб уникнути повторного введення email. Результатом є збереження даних у localStorage, що забезпечує безперервність взаємодії користувача з розіграшем.

На рис. 2.6 зображено вигляд інтерфейсу користувача для етапу Email Step

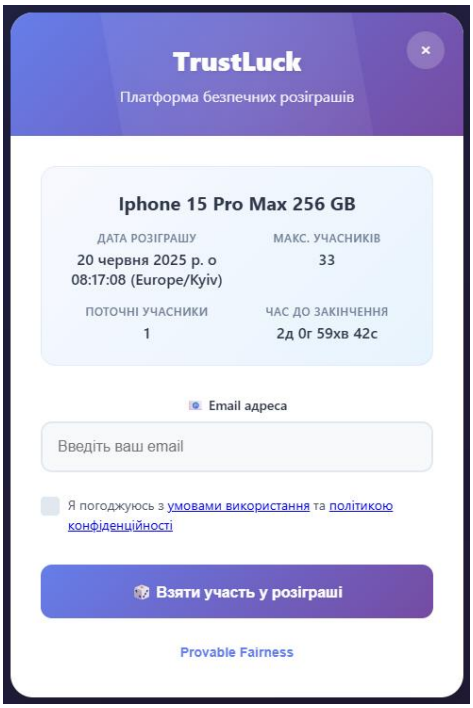


Рисунок 2.6 – Інтерфейс етапу Email Step для участі в розіграші

Етап Email Step завершується успішною відправкою верифікаційного коду на email користувача, збереженням хешу email у localStorage і переходом до Verification Step. Система гарантує унікальність і валідність email, захищаючи дані через хешування та надаючи чіткі повідомлення про помилки (наприклад, при повторній реєстрації чи перевищенні ліміту учасників). Це створює надійну основу для подальшої верифікації та участі в розіграші, забезпечуючи прозорість і зручність для користувача.

Для того, щоб завершити процес реєстрації користувача в розіграші платформи TrustLuck і підтвердити його електронну адресу, необхідно реалізувати етап Verification Step. На цьому етапі користувач вводить шестизначний верифікаційний код, отриманий на свою електронну адресу, через спеціальну форму в модальному вікні. Система перевіряє правильність введеного коду, порівнюючи його з тим, що зберігається на сервері, і, у разі успіху, додає користувача до списку учасників розіграшу, оновлює кількість учасників і переводить його до етапу Success Step. Якщо код невірний, або закінчився термін його дії, відображається відповідне повідомлення про

помилку. Користувач також має можливість повторно надіслати код, якщо він не отримав листа, з урахуванням 30-секундного таймера затримки для запобігання зловживанню надмірними запитами. Цей етап забезпечує надійну верифікацію учасника, гарантуючи, що лише підтверджені користувачі можуть брати участь у розіграші, і підтримує прозорість процесу.

Для розробки етапу Verification Step було використано такі методи:

1. Метод setupVerificationInputs (з компоненту modalTemplate.js): ініціалізує обробники подій для шести полів введення верифікаційного коду, забезпечуючи зручне введення. Він додає слухачі подій для введення цифр, обробки клавіші Backspace (перехід до попереднього поля при видаленні) і вставки коду з буфера обміну. Кожен введений символ перевіряється на відповідність цифрі, і при заповненні поля фокус автоматично переходить до наступного. При вставці шестизначного коду з буфера він розподіляється по полях, і викликається checkVerificationInputs для оновлення стану кнопки підтвердження. Результатом є налаштована форма введення, яка полегшує користувачу введення коду і знижує ймовірність помилок.

2. Метод checkVerificationInputs (з компоненту modalTemplate.js): перевіряє, чи заповнені всі шість полів введення верифікаційного коду однією цифрою кожне, і відповідно оновлює стан кнопки підтвердження (verifyBtn). Якщо всі поля заповнені, кнопка стає активною, дозволяючи відправити форму; інакше вона залишається неактивною. Метод викликається при кожній зміні вмісту полів через setupVerificationInputs. Результатом є динамічне керування доступністю кнопки підтвердження, що підвищує зручність і запобігає відправці неповного коду.

3. Метод handleVerificationSubmit (з компоненту modalTemplate.js): обробляє подію відправки форми верифікації, викликаючись при натисканні кнопки "Підтвердити". Він отримує код із полів через getVerificationCode, перевіряє, що код складається з шести цифр, і відправляє POST-запит до маршруту /api/verify-code із параметрами emailHash, code і giveawayId. У разі успішної верифікації функція оновлює дані в localStorage через

saveRegistrationData, встановлюючи статус "registered", оновлює кількість учасників у модальному вікні та переводить користувача до Success Step через showStep. Якщо код невірний або користувач уже зареєстрований, відображається повідомлення про помилку або Duplicate Step. Результатом є завершення верифікації, додавання користувача до розіграшу або інформування про проблему.

4. Метод verifyCode (з контролеру giveawayController.js): обробляє POST-запит до маршруту /api/verify-code, отримуючи emailHash, code і giveawayId. Він перевіряє наявність коду в пам’яті сервера (Map verificationCodes), порівнює його з отриманим кодом і контролює, чи не минув термін дії (10 хвилин). Якщо код валідний, функція викликає addParticipant для додавання користувача до таблиці giveaway_<id> у базі даних, видаляє код із пам’яті, оновлює лічильник учасників через getParticipantCount і надсилає оновлення через Socket.IO до всіх клієнтів у кімнаті giveaway_<id> за допомогою події participantUpdate. У разі помилок, таких як невалідний код або повторна реєстрація, повертається JSON із повідомленням. Результатом є успішна реєстрація учасника та оновлення даних розіграшу або повідомлення про помилку.

5. Метод addParticipant (з моделі giveawayModel.js): додає користувача до розіграшу, отримуючи giveawayId і email. Вона хешує email через generateEmailHash, перевіряє ліміт учасників і унікальність email у транзакції MySQL, вставляє дані в таблицю giveaway_<id> і оновлює лічильник participant_count у таблиці giveaways. Якщо email уже зареєстрований або ліміт перевищено, повертається помилка. Результатом є додавання учасника до бази даних і оновлення даних розіграшу, що підтверджує його участь.

6. Метод handleResendCode (з компоненту modalTemplate.js): обробляє запит на повторну відправку верифікаційного коду, викликаючись при натисканні кнопки "Надіслати повторно". Він перевіряє, чи минув таймер затримки (30 секунд), відправляє POST-запит до /api/send-verification із параметрами emailHash і giveawayId, і, у разі успіху, оновлює таймер затримки

через startResendCooldown, відображаючи повідомлення про успішну відправку. Якщо відправка не вдалася, відображається повідомлення про помилку. Результатом є повторна відправка коду на email користувача, що дозволяє продовжити верифікацію.

7. Метод startResendCooldown (з компоненту modalTemplate.js): запускає 30-секундний таймер затримки для кнопки повторної відправки коду, оновлюючи її текст із відліком секунд і блокуючи повторні запити до завершення таймера. Він викликається після успішної відправки коду в handleResendCode або handleFormSubmit. Результатом є контроль частоти повторних відправок, що запобігає зловживанню і забезпечує стабільність системи.

8. Метод saveRegistrationData (з компоненту modalTemplate.js): зберігає оновлені дані реєстрації (emailHash, giveawayTitle, giveawayId, status) у localStorage після успішної верифікації в handleVerificationSubmit. Він фіксує статус "registered", дозволяючи системі запам’ятати, що користувач завершив реєстрацію, і коректно відображати Success Step при повторному відкритті вікна. Результатом є збереження стану учасника, що забезпечує безперервність взаємодії.

9. Метод getVerificationCode (з компоненту modalTemplate.js): об’єднує значення шести полів введення в один шестизначний код, викликаючись у handleVerificationSubmit. Результатом є підготовлений код для відправки на сервер, що спрощує обробку даних.

10. Метод clearVerificationInputs (з компоненту modalTemplate.js): очищає всі поля введення коду після невдалої верифікації, викликаючи checkVerificationInputs для оновлення стану кнопки підтвердження. Він застосовується в handleVerificationSubmit при помилках. Результатом є скидання форми для повторного введення коду.

Етап Verification Step завершується успішною верифікацією коду, додаванням користувача до списку учасників у базі даних, оновленням кількості учасників у реальному часі через Socket.IO і переходом до Success

Step. Система гарантує, що лише підтверджені email-адреси беруть участь у розіграші, захищає від повторних реєстрацій і надає зручний механізм повторної відправки коду. У разі помилок користувач отримує чіткі повідомлення, що забезпечує прозорість і зручність взаємодії, створюючи основу для подальшої участі в розіграші.

На рис. 2.7 зображено вигляд інтерфейсу користувача для етапу Success Step

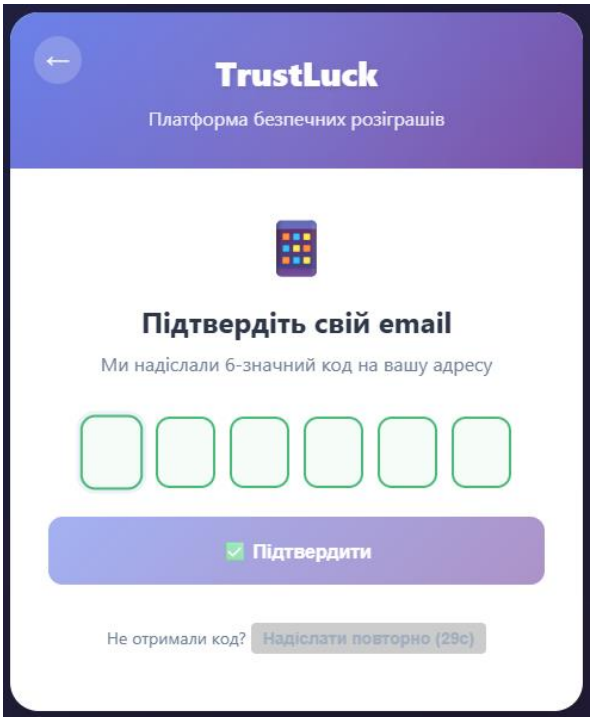


Рисунок 2.7 – Інтерфейс етапу Verification Step для підтвердження участі в розіграші

Етап Success Step на платформі TrustLuck є завершальним етапом процесу реєстрації, де користувач, після успішної верифікації коду, отримує підтвердження участі в розіграші та очікує його завершення. На цьому етапі відображається повідомлення про успішну реєстрацію, а також деталі розіграшу, включаючи назву, дату завершення, максимальну та поточну кількість учасників, таймер до завершення. Основна мета – підготувати користувача до події завершення розіграшу, яке відбувається автоматично після вичерпання серверного таймера або вручну за рішенням адміністратора.

Завершення розіграшу передбачає визначення переможців на сервері, після чого користувач перенаправляється на відповідний етап: Winner Step (у разі перемоги), Loser Step (якщо не виграв) або NotParticipated Step (якщо не брав участі).

На етапі Success Step завершення розіграшу супроводжується подією giveawayEnded. Коли розіграш завершується, сервер надсилає WebSocket-подію giveawayEnded через Socket.IO, яка містить результати, включаючи список переможців (хеші їхніх електронних адрес), Server Seed, Cliend Seed та хеш від Server Seed. Клієнтська логіка в компоненті modalTemplate.js обробляє цю подію, порівнюючи хеш електронної адреси користувача (emailHash), збережений у localStorage методом saveRegistrationData, зі списком переможців. Якщо збіг є, користувачу присвоюється статус "winner, після чого він перенаправляється на Winner Step. Якщо користувач зареєстрований, але не переміг, – на Loser Step. Якщо не брав участі, – на Not Participated Step. Результатом події є автоматичне перенаправлення на відповідний етап із відображенням результатів.

Отже, на етапі Success Step користувач очікує завершення розіграшу, яке відбувається через серверне визначення переможців. Основне на цьому етапі – це WebSocket-подія giveawayEnded, яка надсилає результати та ініціює перенаправлення на Winner Step, Loser Step або NotParticipated Step залежно від статусу користувача, визначеного за даними з localStorage, збереженими через saveRegistrationData.

На рисунку 2.8 відображено вигляд інтерфейс користувача на етапі Succes Step.

Для інформування користувача про неможливість повторної реєстрації в розіграші на платформі TrustLuck необхідно реалізувати етап Duplicate Step. Цей етап активується, коли на етапі Email Step система виявляє, що введена email-адреса вже зареєстрована в поточному розіграші.

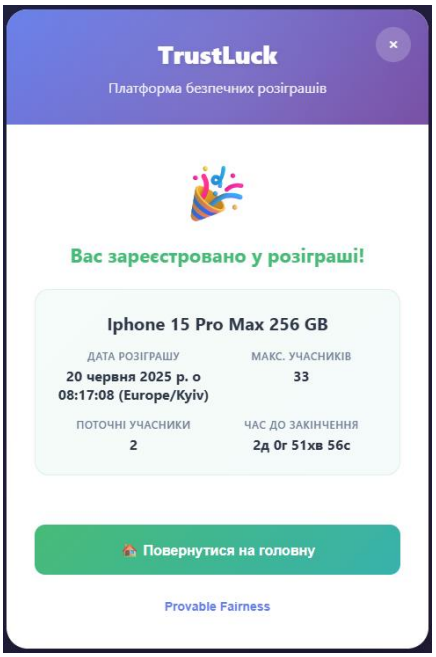


Рисунок 2.8 – Інтерфейс користувача на етапі Success Step

В методі sendVerification при обробці POST-запиту до /api/send-verification функція викликає isEmailRegistered, яка перевіряє наявність хешу електронної адреси користувача в таблиці бази даних розіграшу. Якщо email вже є в базі, повертається JSON із помилкою "Email already registered", що на клієнтській стороні викликає метод showStep для переходу на Duplicate Step.

На рис 2.9 зображено вигляд інтерфейсу користувача на етапі Duplicate Step

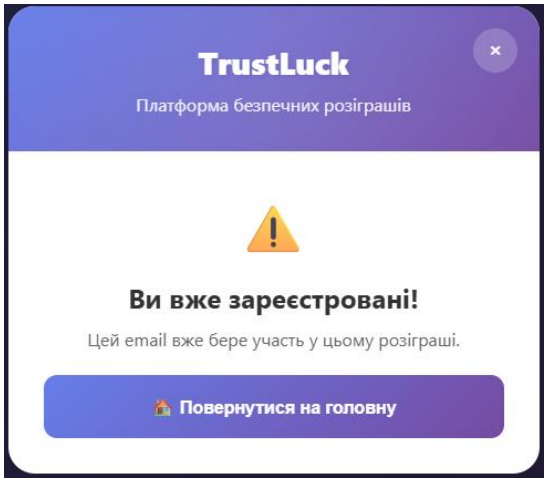


Рисунок 2.9 – Інтерфейс користувача на етапі Duplicate Step

Етап Provable Fairness Success Step є частиною механізму Provable Fairness, який забезпечує прозорість вибору переможців і можливість перевірки чесності результатів для всіх учасників. Цей етап відображається, коли розіграш ще активний, і дозволяє користувачам ознайомитися з принципами роботи криптографічного алгоритму, який визначає переможців, а також переглянути хеш від Server Seed, що публікується заздалегідь. Основна мета етапу – продемонструвати користувачам, що результати розіграшу не можуть бути підроблені, оскільки Server Seed зафіксовано до завершення розіграшу, а Client Seed формується на основі хешу списку email учасників після завершення. Цей етап відображається для активних розіграшів, коли користувач натискає кнопку "Provable Fairness" на етапах Email Step, Success Step або Duplicate Step.

Для розробки етапу Provable Fairness Success Step було використано такі методи:

1. Методи getGiveawayInfo (з контролеру giveawayController.js) та openModal (з компоненту modalTemplate.js) забезпечують генерацію (імпортуючи функцію generateServerSeed з компоненту winnerService.js) та передачу хешу від Server Seed через API зі шляхом /api/giveaway-info для його відображення в інтерфейсі користувача.

2. Метод updateFairnessFields (з компоненту modalTemplate.js): оновлює поля інтерфейсу, пов’язані з Provable Fairness, зокрема заповнює поле для серверного ключа значенням його хешу, отриманим з конфігу, який генерувався в методі getGiveawayInfo та заповнювався методом openModal. Це дозволяє користувачу побачити хеш від Server Seed, який може бути використано для перевірки результатів після завершення розіграшу.

На рис 2.10 зображено вигляд інтерфейсу користувача на етапі Provable Fairness Success Step

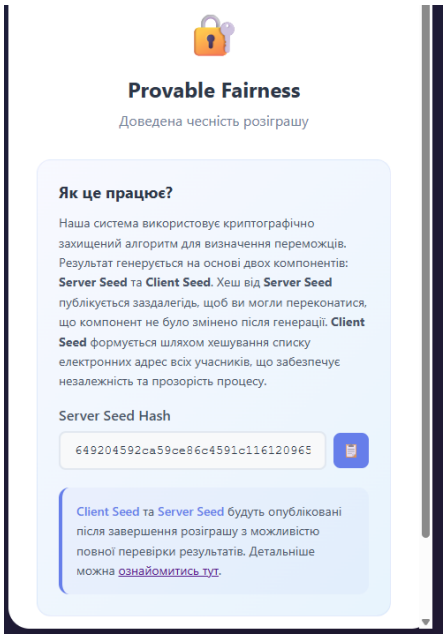


Рисунок 2.10 – Інтерфейс користувача на етапі Provable Fairness Success Step

Етапи Winner Step та Loser Step у системі TrustLuck призначені для інформування користувачів про результати завершеного розіграшу. Winner Step відображається для учасників, які були обрані переможцями, повідомляючи їх про перемогу та відправку листу з інструкціями щодо отримання призу на електронну адресу. Loser Step відображається для зареєстрованих учасників, які не стали переможцями, із повідомленням про невдачу. Обидва етапи активуються після завершення розіграшу, коли система визначила переможців за допомогою механізму вибору переможців. Ці етапи забезпечують чітке сповіщення користувачів про їхній статус у розіграші.

Для розробки етапів Winner Step та Loser Step було використано такі методи:

1. Метод getGiveawayWinners (з контролеру giveawayController.js): обробляє GET-запит до маршруту /api/giveaway-winners?giveawayId={giveawayId} і повертає дані про переможців розіграшу. Вона викликає модель getGiveaway із giveawayModel.js, щоб отримати інформацію про розіграш, зокрема статус (status), список переможців (winners),

serverSeed, serverSeedHash і clientSeed. Якщо розіграш завершений (status === 'ended'), функція повертає JSON-об’єкт із цими даними; інакше повертається порожній список переможців. Логування запиту фіксує час і деталі для дебагінгу. Результатом є надання клієнту даних, необхідних для визначення переможців і відображення Winner/Loser Step.

2. Метод saveRegistrationData (з компоненту modalTemplate.js): зберігає дані про реєстрацію користувача в localStorage, включаючи email-хеш, назву розіграшу, giveawayId і статус ('registered' або 'winner'). На етапі Winner Step, якщо користувач є переможцем, статус оновлюється до 'winner', що дозволяє зберегти цей стан для подальших перевірок. Результатом є збереження стану користувача локально, що підвищує ефективність і зменшує кількість серверних запитів.

3. Подія giveawayEnded (з компоненту modalTemplate.js): обробляє подію завершення розіграшу, отриману через Socket.IO, коли сервер сповіщає всіх клієнтів у кімнаті giveaway_{giveawayId}. Вона оновлює локальні змінні (winners, serverSeed, serverSeedHash, clientSeed, giveawayStatus) і зупиняє таймер відліку (clearInterval(countdownTimer)). Далі викликається showStep для відображення Winner Step, якщо email-хеш користувача є в списку переможців, або Loser Step, якщо ні. Подія запускається сервером у функції finishGiveaway (adminController.js). Результатом є миттєве сповіщення користувача про результати розіграшу в реальному часі.

4. Метод finishGiveaway (з контролеру adminController.js): завершує розіграш, викликаючи selectWinners із winnerService.js для визначення переможців на основі Server Seed і Client Seed. Вона оновлює статус розіграшу в базі даних (status: 'ended'), зберігає переможців і надсилає email-повідомлення переможцям через sendWinnerEmail (emailService.js). Функція також викликає подію giveawayEnded для сповіщення клієнтів через Socket.IO. Вона використовується як у ручному завершенні (метод endGiveawayHandler), так і в автоматичному через таймер (метод endGiveawayByTimer). Результатом є

завершення розіграшу, збереження результатів і сповіщення всіх учасників, що запускає відображення Winner/Loser Step.

5. Метод sendWinnerEmail (з компоненту emailService.js): надсилає email-повідомлення переможцям із інформацією про розіграш (назва, опис призу). Вона використовує nodemailer із конфігурацією gmail для відправки HTML-листа, який містить привітання та деталі призу. Викликається в finishGiveaway для кожного переможця. Результатом є успішна відправка листа, що підтверджує перемогу користувача, відображену в Winner Step.

На рис 2.11 зображено вигляд інтерфейсу користувача на етапах Winner/Loser Step

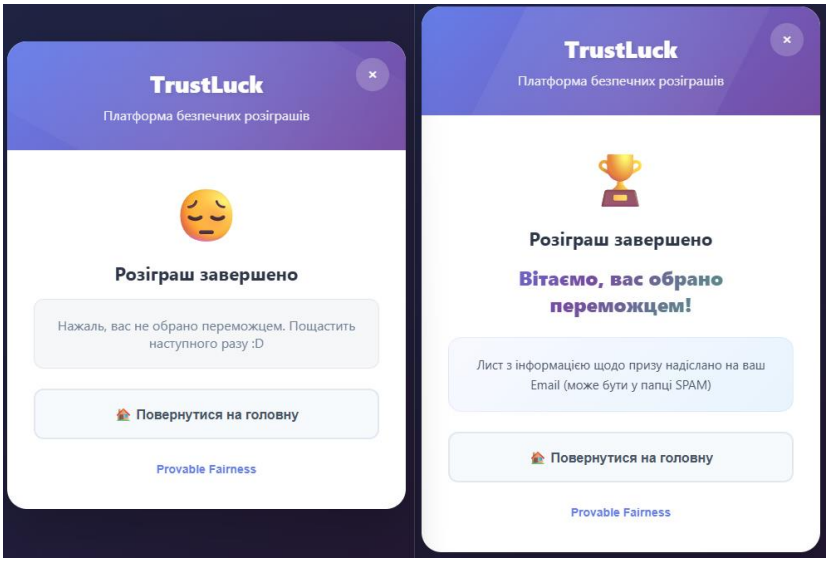


Рисунок 2.11 - Інтерфейс користувача на етапах Winner/Loser Step

Етап Not Participated Step у системі TrustLuck призначений для інформування користувачів, які не брали участі в розіграші, про те, що вони не були зареєстровані як учасники. Цей етап відображається, коли користувач намагається відкрити вікно з розіграшем після його завершення, але його хеш відсутній в базі учасників розіграшу. Мета етапу – повідомити користувача про відсутність його участі в розіграші.

Для розробки етапу Not Participated Step було використано метод checkUserRegistration (з компоненту modalTemplate.js), який перевіряє, чи був користувач зареєстрований у розіграші, використовуючи email-хеш, збережений у localStorage. Він відправляє POST-запит до маршруту /api/check-registration із параметрами emailHash і giveawayId. Сервер повертає JSON-об’єкт із полем isRegistered, яке вказує, чи є email-хеш у базі учасників. Якщо isRegistered дорівнює false або в localStorage відсутні дані про реєстрацію, функція очищає локальні дані (clearRegistrationData) і повертає false, що призводить до відображення Not Participated Step для завершеного розіграшу. Результатом є точне визначення статусу користувача, що дозволяє системі перейти до відповідного етапу.

На рис 2.12 зображено вигляд інтерфейсу користувача на етапі Not Participated Step

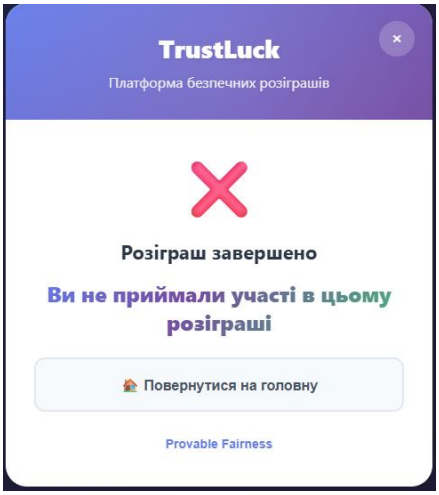


Рисунок 2.12 - Інтерфейс користувача на етапі Not Participated Step

Етап Provable Fairness Result Step у системі TrustLuck призначений для забезпечення прозорості та можливості самостійної перевірки результатів завершеного розіграшу користувачами. На цьому етапі користувачам надаються дані, необхідні для верифікації чесності вибору переможців, зокрема серверне зерно (serverSeed), клієнтське зерно (clientSeed) та можливість перевірити результати через спеціальну форму. Етап активується після

завершення розіграшу, коли користувач натискає кнопку Provable Fairness на етапах Winner Step, Loser Step або Not Participated Step. Мета етапу – дозволити користувачам переконатися, що результати розіграшу були сформовані чесно, опублікувавши дані які використовуються для перевірки алгоритму. У підсумку користувач отримує доступ до інструментів для перевірки результатів, включаючи можливість скопіювати serverSeed і clientSeed для самостійної верифікації, а також виконати автоматичну перевірку через запит до сервера.

Для реалізації етапу Provable Fairness Result Step на платформі TrustLuck, який активується після завершення розіграшу, було впроваджено механізм публікації значень Server Seed і Client Seed для використання на сторінці verify.html (інтерфейс користувача для перевірки результатів). Ці значення є необхідними для перевірки результатів розіграшу. Вказана сторінка забезпечує прозору верифікацію процесу визначення переможців, дозволяючи користувачам відтворити алгоритм і дізнатися отримане місце та статус за вказаною електронною адресою. Вигляд інтерфейсу користувача на цьому етапі зображено на рис. 2.13.

Рисунок 2.13 - Інтерфейс користувача на етапі Provable Fairness Result Step

Для зберігання значень Server Seed і Client Seed на етапі Provable Fairness Result Step використовуються локальні змінні (serverSeed, clientSeed) у файлі компоненту modalTemplate.js. Вони оновлюються через GET-запит до /api/giveaway-winners і відображаються в інтерфейсі за допомогою функції updateFairnessFields.

Етап Max Participants Reached Step у системі TrustLuck призначений для інформування користувачів про те, що в активному розіграші досягнуто максимальної кількості учасників, і тому нові учасники не можуть приєднатися.

Для розробки етапу Max Participants Reached Step було використано такі методи:

1. Ключовим є метод openModal (з компоненту modalTemplate.js), який ініціалізує модальне вікно через GET-запит до /api/giveaway-info, отримуючи дані про розіграш (title, maxParticipants, currentParticipants) від getGiveawayInfo (з контролеру giveawayController.js); якщо currentParticipants >= maxParticipants і checkUserRegistration (POST-запит до /api/check-registration) підтверджує відсутність реєстрації, викликається showStep для відображення maxParticipantsReachedStep.
2. Метод getGiveawayInfo залежить від getGiveaway і getParticipantCount (з моделі giveawayModel.js), які виконують SQL-запити для отримання max_participants і participant_count із бази даних. При спробі реєстрації sendVerification (POST-запит до /api/send-verification) перевіряє ліміт через getParticipantCount і повертає помилку "Maximum participants limit reached", що обробляється в handleFormSubmit, викликаючи відображення етапу Max Participants Reached Step.

Посилання на повний програмний код до реалізованого проєкту TrustLuck вказано у ДОДАТКУ С.

2.7 Розробка адміністративного інтерфейсу та його функціональної частини

Розробка адміністративного інтерфейсу та його функціональної частини є ключовим етапом створення системи TrustLuck, оскільки він забезпечує зручний інструмент для управління розіграшами. Адміністративний інтерфейс розроблено з урахуванням принципів простоти, безпеки та функціональності, що дозволяє ефективно створювати, видаляти та завершувати розіграші, а також переглядати історію. Функціональна частина реалізована за допомогою серверних компонентів, які інтегруються з клієнтським інтерфейсом через API, забезпечуючи надійну обробку запитів і захист даних. У цьому підрозділі розглядаються структура адміністративного інтерфейсу, його функціональні можливості, а також технології та методи, використані для їх реалізації.

Адміністративний інтерфейс реалізований у файлі admin.html, який є основною сторінкою для доступу до адміністративних функцій. Інтерфейсу адміністративної панелі складається з двох основних секцій: форми для створення нового розіграшу, зображеної на рис. 2.14 і таблиці зі списком всіх наявних розіграшів, представленої на рис. 2.15.

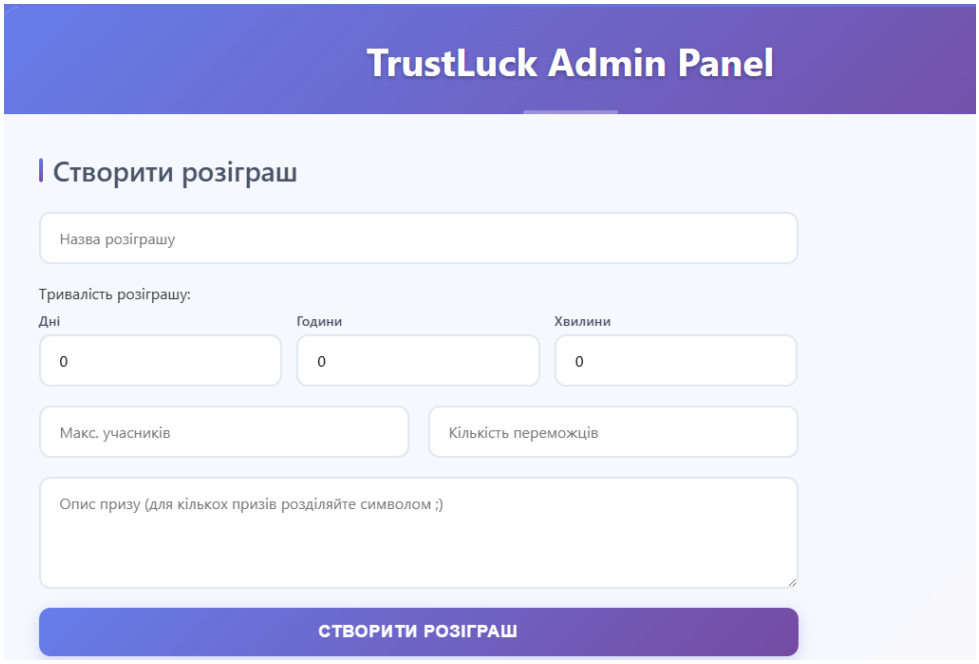


Рисунок 2.14 – Вигляд секції для створення розіграшу

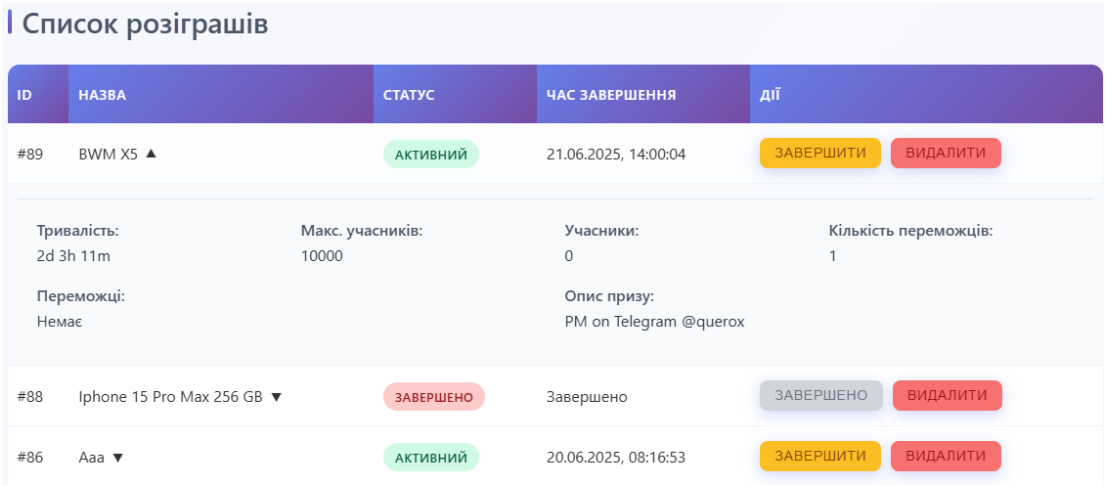


Рисунок 2.15 – Вигляд секції з списком історії розіграшів

Форма створення розіграшу містить поля для введення назви, тривалості (у днях, годинах і хвилинах), максимальної кількості учасників, кількості переможців і опису призу. Ці поля дозволяють адміністратору налаштувати параметри розіграшу, забезпечуючи гнучкість у конфігурації. Таблиця розіграшів відображає ідентифікатор, назву, статус, час завершення та доступні дії (видалення та завершення), що забезпечує зручний огляд і керування всіма розіграшами. Для захисту доступу до адміністративного інтерфейсу використовується базова HTTP-автентифікація, реалізована через middleware express-basic-auth у файлі маршрутизатора adminRoutes.js. Вона вимагає введення логіна та пароля, що забезпечує базовий рівень безпеки та є прийнятним для локального розміщення, хоча в продуктивному середовищі рекомендується використовувати більш надійні методи автентифікації, такі як JWT або OAuth.

Функціональна частина адміністративного інтерфейсу реалізована через маршрути, визначені у файлі adminRoutes.js, які обробляються контролером з adminController.js. Основні маршрути включають: GET /api/admin/ для відображення сторінки admin.html, GET /api/admin/giveaways для отримання списку всіх розіграшів, POST /api/admin/giveaways для створення нового розіграшу, DELETE /api/admin/giveaways/:id для його видалення та POST /api/admin/giveaways/:id/end для завершення розіграшу з вибором переможців.

Кожен маршрут захищений middleware express-basic-auth, що гарантує доступ лише авторизованим користувачам.

Для створення розіграшу використовується метод createGiveawayHandler з adminController.js, який обробляє POST-запит до /api/admin/giveaways. Цей метод перевіряє наявність обов'язкових полів (назва, тривалість, максимальна кількість учасників, кількість переможців), генерує Server Seed і його хеш за допомогою generateServerSeed з компоненту winnerService.js, а також створює запис у базі даних через createGiveaway з моделі giveawayModel.js. Для забезпечення коректного відліку часу розіграшу метод додає таймер у timerManager з компоненту timerManager.js, який автоматично завершує розіграш після закінчення заданого часу. У разі успішного створення розіграшу адміністратору повертається ідентифікатор розіграшу та сформовані зерна.

Метод getAllGiveawaysHandler обробляє GET-запит до /api/admin/giveaways і повертає список усіх розіграшів, отриманих через getAllGiveaways з моделі giveawayModel.js. Для кожного розіграшу додатково обчислюється кількість учасників (participantCount) і час завершення (endTime) за допомогою getParticipantCount і parseDuration з компоненту timeUtils.js, що забезпечує адміністратору повну інформацію про стан розіграшів. Метод deleteGiveawayHandler (DELETE /api/admin/giveaways/:id) видаляє розіграш і пов'язану таблицю учасників, а також зупиняє відповідний таймер. Завершення розіграшу реалізовано через endGiveawayHandler (POST /api/admin/giveaways/:id/end), який викликає finishGiveaway для вибору переможців за допомогою selectWinners з компоненту winnerService.js, надсилає листи переможцям через метод sendWinnerEmail з компоненту emailService.js та сповіщає всіх учасників через Socket.IO надсилаючи подію giveawayEnded.

Інтеграція з базою даних здійснюється через модуль giveawayModel.js, який використовує пул з'єднань mysql2/promise для виконання транзакцій. Наприклад, метод createGiveaway створює запис у таблиці giveaways і окрему таблицю для учасників (giveaway_{id}), забезпечуючи ізоляцію даних для кожного розіграшу. Метод addParticipant перевіряє ліміт учасників і

унікальність електронної адреси, що запобігає дублюванню. Для синхронізації кількості учасників використовується syncParticipantCount, що гарантує актуальність даних у таблиці giveaways.

Для забезпечення автоматичного завершення розіграшів застосовується timerManager.js, який ініціалізується в server.js і викликає endGiveawayByTimer з контролеру adminController.js після закінчення часу розіграшу. Це дозволяє системі автоматично обробляти завершення без втручання адміністратора.

Таким чином, адміністративний інтерфейс і його функціональна частина забезпечують повний контроль над розіграшами, від створення до завершення, з інтеграцією механізму вибору переможців та автоматичним сповіщенням учасників. Використання Express.js, Socket.IO, MySQL і модульної структури коду забезпечує надійність, масштабованість і зручність адміністрування системи TrustLuck.

2.8 Реалізація інтерфейсу для перевірки достовірності результатів розіграшу

Інтерфейс для перевірки достовірності результатів є ключовим елементом веб-платформи TrustLuck, що забезпечує користувачам можливість самостійно перевірити результати розіграшу, підтверджуючи їх чесність і прозорість за допомогою відтворення криптографічного алгоритму, використовуваного у системі. Він забезпечує введення необхідних даних – електронної адреси, ідентифікатору розіграшу, значень Server Seed і Client Seed – для верифікації участі та отриманого статусу.

Інтерфейс verify.html побудовано як просту HTML-форму з мінімалістичним дизайном, що включає поля для введення даних і кнопку для відправки форми. Форма відправляє запит до серверного API, який обробляє введені дані та повертає результат перевірки.

На клієнтській стороні verify.html інтегрується з modalTemplate.js, який забезпечує користувача даними для перевірки, а саме Server Seed чи Client Seed

на етапі Provable Fairness Result Step. Функція SHA-256 генерує хеш для введеної електронної адреси, що використовується для безпечної передачі даних на сервер і порівняння з записами в базі даних. Ці методи забезпечують зручність і безпеку взаємодії користувача з формою.

На серверній стороні основна логіка перевірки реалізована в контролері giveawayController.js через метод verifyResult. Цей метод обробляє POST-запит до ендпоінту /api/verify-result, отримуючи дані з форми verify.html. Він перевіряє наявність усіх необхідних полів, валідність ідентифікатора розіграшу та його статус. Далі метод підтверджує правильність Server Seed і Client Seed, порівнюючи їх із збереженими в базі даних значеннями. Для цього він звертається до бази через метод getGiveaway, який повертає деталі розіграшу. Якщо Server Seed не збігається, повертається помилка.

Для перевірки Client Seed метод verifyResult використовує список електронних адрес учасників, отриманий через getParticipants. Цей список хешується для створення Client Seed, який порівнюється з введеним значенням. Якщо Client Seed валідний, для кожного учасника обчислюється унікальне числове значення (score) за допомогою HMAC-SHA256, використовуючи Server Seed і хеш електронної адреси. Логіка обчислення score базується на методі generateWinners, який сортує учасників за їх оцінками та визначає переможців. У verifyResult ця логіка відтворюється для визначення рангу користувача та його статусу. Метод повертає детальну інформацію, що зображена на рис. 2.16, включаючи дату реєстрації, зайняте місце, хеш від Server Seed, згенероване score, поріг для входу в переможці та отриманий статус.

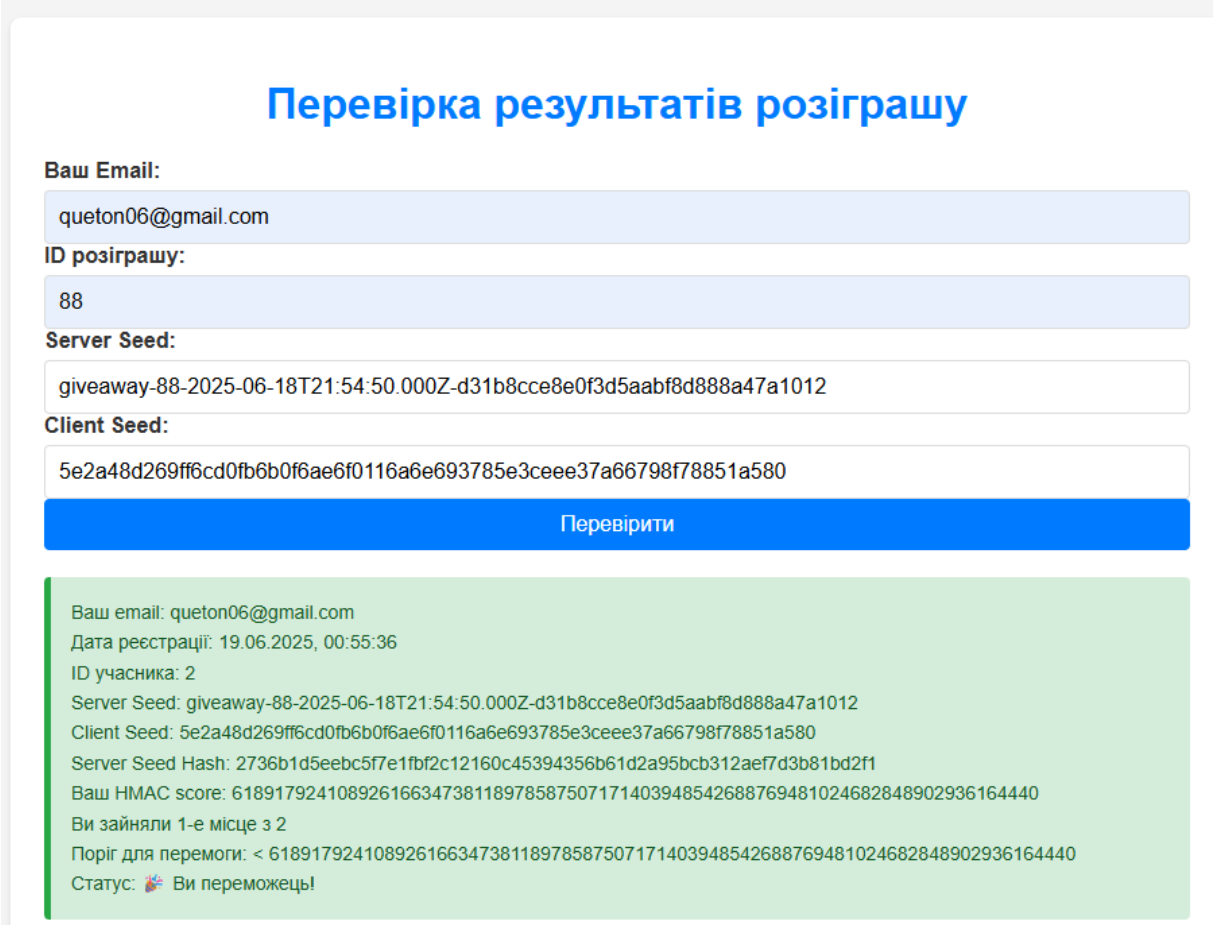


Рисунок 2.16 – Інтерфейс для перевірки достовірності результатів

Таким чином, verify.html є важливим компонентом системи TrustLuck, що реалізує принцип Provably Fair. Інтерфейс поєднує простоту використання з надійною криптографічною верифікацією, забезпечуючи довіру користувачів до результатів розіграшу.

2.9 Захист від загроз та вразливостей

В ході розробки системи TrustLuck було використано низку механізмів для захисту від потенційних загроз і вразливостей, що забезпечують безпеку даних, захист від несанкціонованого доступу та стабільність роботи платформи. Ці механізми охоплюють захист адміністративного доступу, обмеження кількості запитів, безпечне зберігання та обробку даних, а також забезпечення прозорості та цілісності процесу розіграшу. Нижче детально описано

застосовані заходи безпеки, їх функціональність і роль у захисті системи від можливих атак.

Для захисту адміністративного інтерфейсу від несанкціонованого доступу використано базову HTTP-автентифікацію, реалізовану через middleware express-basic-auth у файлі adminRoutes.js. Цей механізм вимагає введення логіна та пароля доступу до маршрутів /api/admin/*, включаючи створення, видалення та завершення розіграшів. Middleware налаштовано з параметром challenge: true, що викликає запит автентифікації в браузері, та unauthorizedResponse: 'Unauthorized', що повертає повідомлення про помилку в разі неправильних облікових даних. Хоча цей метод забезпечує базовий рівень захисту, він ефективний для обмеження доступу до адміністративних функцій у локальному середовищі. Для продуктивного використання рекомендується замінити його на більш надійні методи, такі як JWT або OAuth, з безпечним зберіганням паролів у хешованому вигляді.

Для запобігання атакам типу "відмова в обслуговуванні" (DoS) або зловживанню API використано обмеження частоти запитів (rate limiting) через бібліотеку express-rate-limit у файлі giveawayRoutes.js. Зокрема, маршрут /api/verify-result, який відповідає за верифікацію результатів розіграшу, обмежено до 10 запитів з однієї IP-адреси за 15 хвилин (windowMs: 15 * 60 * 1000, max: 10). У разі перевищення ліміту користувачу повертається повідомлення "Too many verification requests, please try again later". Це знижує ризик перевантаження сервера через надмірні запити та захищає від автоматизованих атак, таких як брутфорс.

Безпека обробки електронних адрес користувачів забезпечується через їх хешування за допомогою алгоритму SHA-256 на клієнтській та серверній стороні. У базі даних зберігаються як хешовані електронні адреси (email_hash), так і оригінальні адреси (email). Хешування застосовується для забезпечення конфіденційності даних під час запитів на отримання списку переможців через API-ендпоінт /api/giveaway-winners, що необхідно для оновлення інтерфейсу користувача щодо статусу. Також під час реєстрації учасника оригінальна

електронна адреса використовується для надсилання верифікаційних листів, тоді як її хеш зберігається для безпечної перевірки участі. Це дозволяє мінімізувати ризики витоку персональних даних і забезпечити конфіденційність при обробці запитів на клієнтському інтерфейсі.

Для захисту від дублювання участі в розіграші реалізовано перевірку унікальності електронних адрес у методі addParticipant (з моделы giveawayModel.js). Цей метод виконує транзакцію, перевіряючи наявність хешу електронної адреси в таблиці giveaway_{id} перед додаванням нового учасника. У разі виявлення дубля повертається помилка "Email already registered", що запобігає множинним реєстраціям з однієї адреси. Додатково метод перевіряє ліміт учасників (max_participants), що захищає від перевищення встановленої кількості учасників і можливих атак шляхом масової реєстрації.

Механізм доведеної чесності (Provable Fairness), реалізований у компоненті winnerService.js, забезпечує захист від маніпуляцій результатами розіграшу. Значення Server Seed генерується за допомогою функції generateServerSeed із використанням криптографічно безпечного методу crypto.randomBytes. Його хеш публікується заздалегідь і зберігається в базі даних, дозволяючи користувачам перевірити, що Server Seed не змінювалося після завершення розіграшу. Client Seed формується шляхом хешування всіх електронних адрес учасників, що унеможливлює вплив адміністратора на вибір переможців. Метод generateWinners використовує HMAC-SHA256 для створення унікальних числових значень для кожного учасника, забезпечуючи детермінований і прозорий процес вибору переможців.

Для захисту від втрати даних і забезпечення цілісності операцій із базою даних використано транзакції MySQL у методах createGiveaway, addParticipant і deleteGiveaway (з моделі giveawayModel.js). Наприклад, при створенні розіграшу (createGiveaway) транзакція забезпечує атомарність створення запису в таблиці giveaways і відповідної таблиці учасників (giveaway_{id}). У разі помилки транзакція відкочується, запобігаючи неконсистентності даних. Пул з’єднань (mysql2/promise) у giveawayModel.js оптимізує обробку запитів до бази

даних, зменшуючи ризик перевантаження та забезпечуючи стабільність при високій кількості запитів.

Верифікаційні коди, які надсилаються користувачам через sendVerificationEmail (з компоненту emailService.js), мають обмежений час дії (10 хвилин), що реалізовано через verificationCodes Map у контролері giveawayController.js. Коди автоматично видаляються після закінчення терміну дії за допомогою setInterval, що зменшує ризик повторного використання старих кодів у разі їх перехоплення. Надсилання листів здійснюється через захищений транспорт nodemailer із конфігурацією gmail, що забезпечує безпечну передачу повідомлень.

Таким чином, у системі TrustLuck реалізовано комплексний набір механізмів захисту, включаючи базову автентифікацію, обмеження частоти запитів, хешування даних, транзакції бази даних та механізм Provably Fair. Ці заходи мінімізують ризики несанкціонованого доступу, маніпуляцій даними, DoS-атак і забезпечують конфіденційність і прозорість для користувачів.

РОЗДІЛ 3. ТЕСТУВАННЯ СИСТЕМИ TRUSTLUCK ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Методи тестування та критерії оцінки

Для оцінки якості платформи TrustLuck було застосовано комплексний підхід до тестування, що охоплює як функціональні, так і нефункціональні вимоги. Тестування проводилося з метою перевірки продуктивності, безпеки, масштабованості, користувацького досвіду та технічного обслуговування системи. Основним методом тестування обрано мануальне тестування, що дозволило детально оцінити відповідність системи вимогам через виконання тест-кейсів у контрольованих умовах. Нижче описано методи тестування та критерії оцінки для кожної категорії вимог.

Для оцінки продуктивності системи тестування проводилося шляхом симуляції запитів до API за допомогою Postman, для перевірки часу відповіді. Тестування включало сценарії з різною кількістю одночасних користувачів (до 500) для оцінки часу обробки запитів. Критерії оцінки: час відповіді на API-запити (наприклад, /api/giveaway-info, /api/verify-result) не перевищує 1 секунди, оновлення кількості учасників у реальному часі через WebSocket-з’єднання (participantUpdate) відбувається з затримкою до 1 секунди при 1000 активних підключень, а процес вибору переможців (finishGiveaway) завершується за 3 секунди для розіграшів із 5000 учасниками.

Для перевірки безпеки системи проводилося тестування унікальності реєстрації email, механізму доказової чесності та захисту від надмірних запитів. Тестування унікальності email передбачало спроби повторної реєстрації одного email у розіграші через API (/api/send-verification). Механізм доказової чесності перевірявся шляхом введення коректних і некоректних Server Seed та Client Seed у форму verify.html та аналізу відповіді від /api/verify-result. Для захисту від зловмисного навантаження тестувалися обмеження кількості запитів (rate limiting) до ендпоінту /api/verify-result. Критерії оцінки: неможливість

повторної реєстрації email, коректна верифікація результатів лише з правильними насіннями, обмеження запитів до 100 за 15 хвилин з однієї IP-адреси.

Тестування масштабованості проводилося шляхом симуляції розіграшів із кількістю учасників до 10 000. Використовувалися скрипти для генерації масових реєстрацій через /api/send-verification та /api/verify-code. Перевірка швидкості пошуку учасників здійснювалася через SQL-запити до таблиць giveaway_{id}. Для оцінки зберігання даних в оперативній пам’яті тестувалася робота timerManager.js, який кешує таймери активних розіграшів. Критерії оцінки: стабільна робота системи при 10 000 учасниках, час пошуку учасника в базі даних не перевищує 100 мс, таймери розіграшів доступні без затримок.

Тестування інтерфейсу проводилося на різних пристроях (мобільні з роздільною здатністю від 360x640 пікселів, десктопи до 1920x1080 пікселів) для перевірки адаптивності. Перевірялися повідомлення про помилки в інтерфейсі (modal.html, verify.html) на зрозумілість і локалізацію українською мовою. Критерії оцінки: коректне відображення елементів інтерфейсу на всіх пристроях, повідомлення про помилки зрозумілі та українською мовою, усі тексти локалізовані.

Історія розіграшів перевірялася через запит до /api/admin/giveaways, який повертає список розіграшів із кількістю учасників. Критерії оцінки: збереження історії розіграшів із коректними даними про учасників.

3.2 Розробка тест-кейсів для тестування системи

Під час розробки системи TrustLuck було визначено 11 тест-кейсів і вручну протестовано на відповідність критеріям оцінки. Результати тест-рану наведено на рис. 3.1. Для відстеження виконання тестів і фіксації результатів тестування було використано сервіс TestRail.

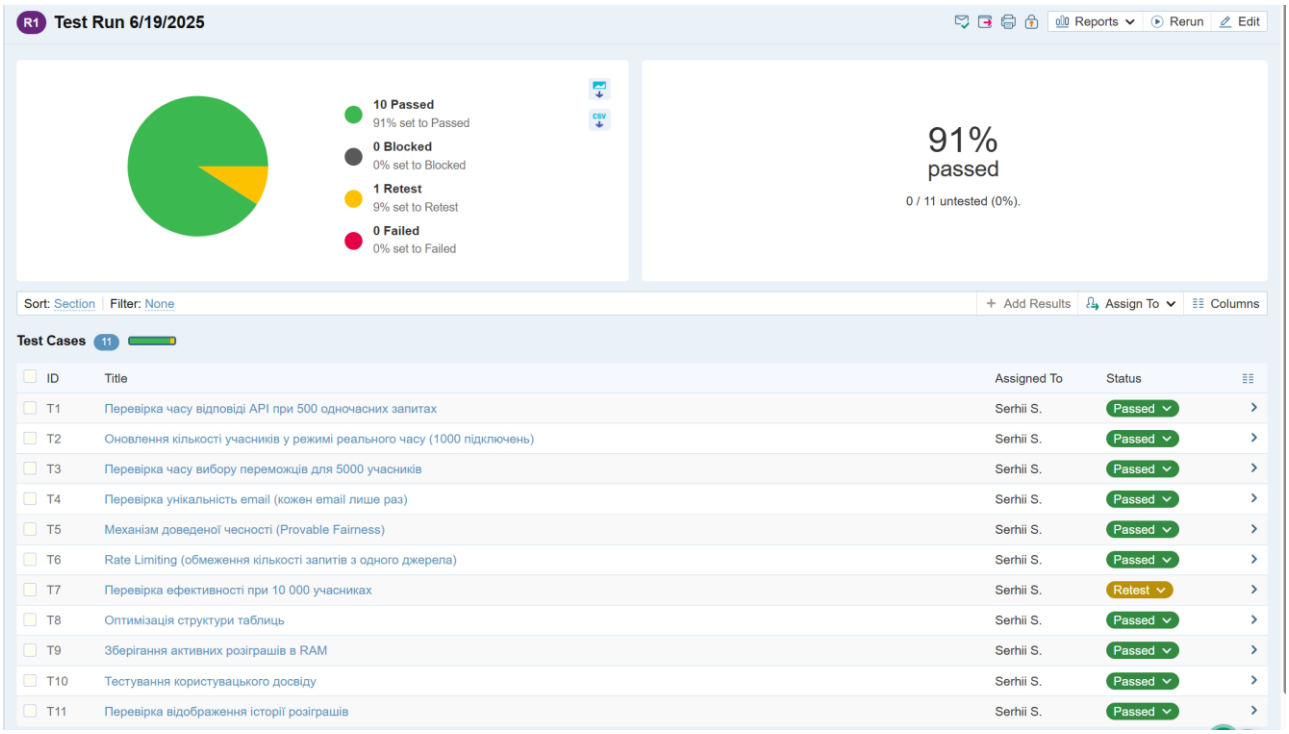


Рисунок 3.1 – Test run тестування системи TrustLuck

ВИСНОВКИ

Було розроблено та реалізовано систему TrustLuck – веб-платформу для проведення прозорих і справедливих розіграшів із використанням підходу Provably Fair. У процесі дослідження було проаналізовано сучасні генератори випадкових чисел, їх застосування в цифрових системах, а також здійснено огляд існуючих платформ, таких як Giveaway.com, з метою виявлення їхніх переваг і недоліків.

У першому розділі визначено ключові принципи, що лежать в основі випадковості та чесності в системах розіграшів, акцентовано увагу на важливості прозорості процесу визначення переможців. Розглянуто технологію Provably Fair, яка дозволяє кожному учаснику перевірити достовірність результату, що є критично важливим для формування довіри до сервісу.

Другий розділ був присвячений практичній реалізації системи TrustLuck. Було визначено функціональні та нефункціональні вимоги, спроєктовано архітектуру системи, обрано сучасні технології для фронтенду та бекенду. Особливу увагу приділено реалізації алгоритму вибору переможців на основі механізму HMAC з генерацією серверного початкового значення за допомогою CSPRNG. Також впроваджено модулі автоматизації поштових повідомлень, створено користувацький та адміністративний інтерфейси, а також інтерфейс для перевірки достовірності результатів.

У третьому розділі проведено всебічне тестування системи TrustLuck, розроблено тест-кейси, визначено критерії оцінки ефективності та безпеки. Результати тестування засвідчили стабільну роботу системи, відповідність вимогам і захищеність від базових типів атак та вразливостей.

Загалом, TrustLuck – це функціональна, безпечна та прозора платформа, що забезпечує високу довіру з боку користувачів. Її архітектура та принципи побудови можуть бути масштабовані для застосування в інших сферах, де критично важливо забезпечити чесний випадковий вибір.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Provably fair system [Електронний ресурс] // Provably.com. – Режим доступу: https://www.provably.com/ (дата звернення: 19.04.2025).

2. Генератори випадкових чисел та сфери їх використання [Електронний ресурс] // MyKyivRegion. – Режим доступу: https://mykyivregion.com.ua/news/generatori-vipadkovix-cisel-ta-sferi-yix-vikoristannya (дата звернення: 12.03.2025).

3. Приклади діаграм контексту [Електронний ресурс] // MindOnMap. – Режим доступу: https://www.mindonmap.com/uk/blog/context-diagram-example/ (дата звернення: 22.05.2025).

4. Побудова динамічних вебзастосунків за допомогою MVC [Електронний ресурс] // Hillel IT School. – Режим доступу: https://blog.ithillel.ua/articles/building-dynamic-web-applications-using-mvc (дата звернення: 24.05.2025).

5. HMAC [Електронний ресурс] // Wikipedia. – Режим доступу: https://uk.wikipedia.org/wiki/HMAC (дата звернення: 25.04.2025).

6. HMAC [Електронний ресурс] // Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/HMAC (дата звернення: 25.04.2025).

7. Лінійний конгруентний метод [Електронний ресурс] // Wikipedia. – Режим доступу: https://uk.wikipedia.org/wiki/Лінійний_конгруентний_метод (дата звернення: 16.05.2025).

8. Метод crypto.randomBytes [Електронний ресурс] // Node.js Documentation. – Режим доступу: https://nodejs.org/api/crypto.html#cryptorandombytessize-callback (дата звернення: 15.03.2025).

ДОДАТКИ

ДОДАТОК А

Перший фрагмент коду компоненту winnerService.js

```
const crypto = require('crypto');
const { getParticipants } = require('../models/giveawayModel');

function generateServerSeed(giveawayId, createdAt) {
  const randomPart = crypto.randomBytes(16).toString('hex');
  const textSeed = `giveaway-${giveawayId}-${createdAt.toISOString()}-${randomPart}`;
  console.log(`Generated serverSeed at ${new Date().toLocaleString('uk-UA', { timeZone: 'Europe/Kyiv' })}:`, textSeed);
  return textSeed;
}

function generateClientSeed(emails) {
  const clientSeed = crypto.createHash('sha256').update(emails.join('')).digest('hex');
  console.log(`Generated clientSeed at ${new Date().toLocaleString('uk-UA', { timeZone: 'Europe/Kyiv' })}:`, clientSeed);
  return clientSeed;
}

function generateWinners(emails, serverSeed, clientSeed, numWinners) {
  const results = emails.map((email) => {
    const emailHash = crypto.createHash('sha256').update(email.toLowerCase()).digest('hex');
    const message = `${clientSeed}:${emailHash}`;
    const hmac = crypto.createHmac('sha256', serverSeed).update(message).digest();
    const score = BigInt(`0x${hmac.toString('hex')}`);
    return { email, emailHash, score };
  });

  results.sort((a, b) => (a.score < b.score ? -1 : 1));
  const winners = results.slice(0, numWinners).map(r => r.emailHash);
  console.log(`Generated winners at ${new Date().toLocaleString('uk-UA', { timeZone: 'Europe/Kyiv' })}:`, winners);
  return { winners, emailMap: new Map(results.map(r => [r.emailHash, r.email])) };
}
```

Другий фрагмент коду компоненту winnerService.js

```
async function selectWinners(giveawayId, numWinners, createdAt) {
  const emails = await getParticipants(giveawayId);

  if (emails.length === 0) {
    console.log(`No participants for giveaway ${giveawayId} at ${new Date().toLocaleString('uk-UA', { timeZone: 'Europe/Kyiv' })}`);
    return {
      winners: [],
      serverSeed: '',
      serverSeedHash: '',
      clientSeed: '',
      emailMap: new Map()
    };
  }

  const serverSeed = generateServerSeed(giveawayId, createdAt);
  const serverSeedHash = crypto.createHash('sha256').update(serverSeed).digest('hex');
  const clientSeed = generateClientSeed(emails);
  const { winners, emailMap } = generateWinners(emails, serverSeed, clientSeed, numWinners);

  return {
    winners,
    serverSeed,
    serverSeedHash,
    clientSeed,
    emailMap
  };
}

module.exports = { selectWinners, generateServerSeed };
```

ДОДАТОК Б

Перший фрагмент коду компонента emailService.js

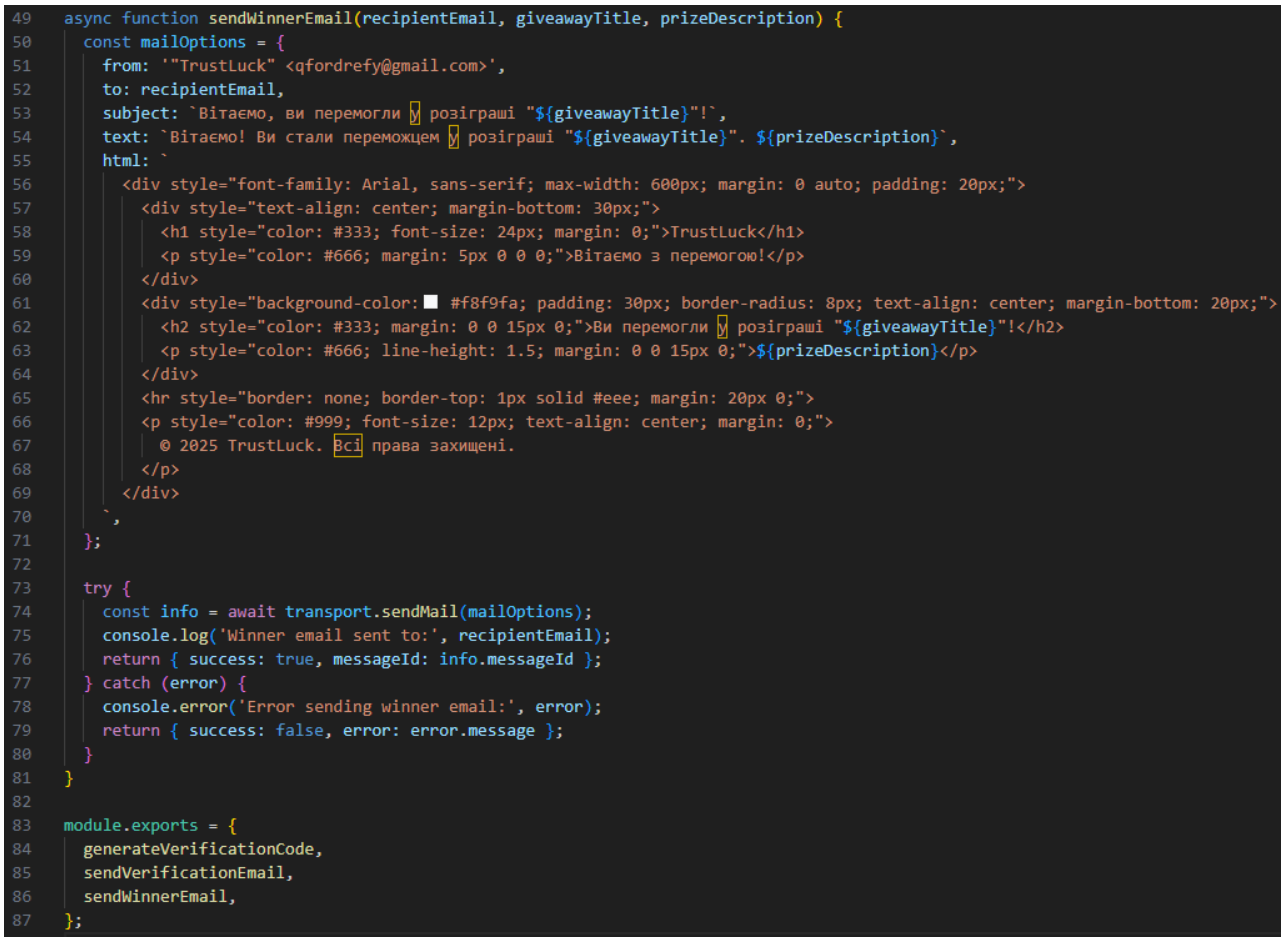
```
const nodemailer = require('nodemailer');
const config = require('../../config/config');
const transport = nodemailer.createTransport(config.mailtrap);

function generateVerificationCode() {
  return Math.floor(100000 + Math.random() * 900000).toString();
}

async function sendVerificationEmail(recipientEmail, verificationCode) {
  const mailOptions = {
    from: '"TrustLuck" <qfordrefy@gmail.com>',
    to: recipientEmail,
    subject: 'Код верифікації - TrustLuck',
    text: `Ваш код верифікації: ${verificationCode}`,
    html: `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto; padding: 20px;">
        <div style="text-align: center; margin-bottom: 30px;">
          <h1 style="color: #333; font-size: 24px; margin: 0;">TrustLuck</h1>
          <p style="color: #666; margin: 5px 0 0 0;">Код верифікації</p>
        </div>
        <div style="background-color: #f8f9fa; padding: 30px; border-radius: 8px; text-align: center; margin-bottom: 20px;">
          <h2 style="color: #333; margin: 0 0 15px 0;">Ваш код верифікації:</h2>
          <div style="font-size: 32px; font-weight: bold; color: #007bff; letter-spacing: 8px; margin: 20px 0;">${verificationCode}</div>
        </div>
        <p style="color: #666; line-height: 1.5; margin: 0 0 15px 0;">
          Введіть цей код на сайті, щоб підтвердити вашу участь у розіграші.
        </p>
        <p style="color: #999; font-size: 14px; margin: 15px 0 0 0;">
          Якщо ви не запитували цей код, просто проігноруйте це повідомлення.
        </p>
        <hr style="border: none; border-top: 1px solid #eee; margin: 20px 0;">
        <p style="color: #999; font-size: 12px; text-align: center; margin: 0;">
          © 2025 TrustLuck. Всі права захищені.
        </p>
      </div>
    `,
  };

  try {
    const info = await transport.sendMail(mailOptions);
    console.log('Verification email sent to:', recipientEmail);
    return { success: true, messageId: info.messageId };
  } catch (error) {
    console.error('Error sending verification email:', error);
    return { success: false, error: error.message };
  }
}
```

Другий фрагмент коду компонента emailService.js



```
async function sendWinnerEmail(recipientEmail, giveawayTitle, prizeDescription) {
  const mailOptions = {
    from: '"TrustLuck" <qfordrefy@gmail.com>',
    to: recipientEmail,
    subject: `Вітаємо, ви перемогли у розіграші "${giveawayTitle}"!`,
    text: `Вітаємо! Ви стали переможцем у розіграші "${giveawayTitle}". ${prizeDescription}`,
    html: `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto; padding: 20px;">
        <div style="text-align: center; margin-bottom: 30px;">
          <h1 style="color: #333; font-size: 24px; margin: 0;">TrustLuck</h1>
          <p style="color: #666; margin: 5px 0 0 0;">Вітаємо з перемогою!</p>
        </div>
        <div style="background-color: #f8f9fa; padding: 30px; border-radius: 8px; text-align: center; margin-bottom: 20px;">
          <h2 style="color: #333; margin: 0 0 15px 0;">Ви перемогли у розіграші "${giveawayTitle}"!</h2>
          <p style="color: #666; line-height: 1.5; margin: 0 0 15px 0;">${prizeDescription}</p>
        </div>
        <hr style="border: none; border-top: 1px solid #eee; margin: 20px 0;">
        <p style="color: #999; font-size: 12px; text-align: center; margin: 0;">
          © 2025 TrustLuck. Всі права захищені.
        </p>
      </div>
    `,
  };

  try {
    const info = await transport.sendMail(mailOptions);
    console.log('Winner email sent to:', recipientEmail);
    return { success: true, messageId: info.messageId };
  } catch (error) {
    console.error('Error sending winner email:', error);
    return { success: false, error: error.message };
  }
}

module.exports = {
  generateVerificationCode,
  sendVerificationEmail,
  sendWinnerEmail,
};
```

ДОДАТОК С

Повний програмний код до реалізованого проєкту TrustLuck розміщено у відкритому репозиторії за посиланням: https://github.com/que1work/TrustLuck.