Journal of Applied Informatics and Computing (JAIC)
Vol.9, No.4, August 2025, pp. 1115~1126
e-ISSN: 2548-6861

Implementation of Blockchain Smart Contract for Online Concert Ticket Transactions Based on NFTs

Anak Agung Lingga Pratyaksa Nugraha [1]*, Ni Wayan Emmy Rosiana Dewi [2]*, Fajar Purnama [3]*
* Teknologi Informasi, Universitas Udayana
aalingga09@gmail.com [1], emmyrosiana@unud.ac.id [2], fajarpurnama@unud.ac.id [3]

Article Info

Article history:

Received 2025-05-03
Revised 2025-07-03
Accepted 2025-07-26

Keyword:

Concert Tickets,
Smart Contract,
Blockchain,
Ethereum,
NFT

ABSTRACT

The development of the entertainment industry, especially music concerts, has driven the transformation of ticket sales systems from conventional to digital methods. Although online concert ticket sales offer greater convenience and reach, they still face the risks of fraud, counterfeit tickets, and unfair distribution. This study, Blockchain Smart Contract Implementation for NFT-Based Online Music Concert Ticket Transactions, aims to develop a ticket sales system using blockchain technology by integrating smart contracts and Non-Fungible Tokens (NFTs). The main objectives are to design and implement smart contracts on the Ethereum network, implement ERC-721-based digital tickets, ensure transparency in transaction history, and verify ticket authenticity through unique identifiers. This study adopts the Agile method, with implementation on the Ethereum Sepolia Testnet and testing using the meta mask digital wallet. The results show that the developed system can automatically hold funds through an escrow mechanism until the ticket is downloaded, generate unique and tamper-proof NFT tickets, display transaction details transparently, and facilitate ticket verification effectively. In conclusion, the use of smart contracts and NFTs significantly improves the security, transparency, and trustworthiness of online music concert ticket transactions.

I. INTRODUCTION

The music concert industry has become a significant part of entertainment culture, enjoyed by generations. With the rapid growth of the music industry, the demand for concert tickets is expected to continue increasing. As technology advances, many event organizers are adopting digital solutions, especially for online ticket sales, which streamline the transaction process without requiring the buyer's physical presence [1].

Digital ticketing has now become a standard practice in various entertainment sectors, including cinemas, festivals, and music concerts. However, digital ticketing systems face several challenges, such as fraud by illegal ticket merchants who often sell tickets at inflated prices or even counterfeit tickets [2]. This results in financial losses for consumers and, in some cases, the loss of proof of payment. Additionally, event organizers often lose control over ticket pricing and distribution, leading to lost revenue and reduced trust in the official ticketing system [3].

To address these issues, blockchain technology is being implemented to enhance transparency, security, and trust in digital ticketing systems [4]. With blockchain, event organizers can better manage ticket inventory, reduce fraud, and ensure a clear transaction trail. The system also increases efficiency and offers solutions to existing problems in the concert ticketing industry [5].

The use of blockchain technology in event ticketing allows for decentralized and secure transaction recording, reducing the risk of data breaches and ticket counterfeiting [6]. Tickets issued as NFTs (Non-Fungible Tokens) can be transparently monitored via blockchain, enabling verification of ownership and secure ticket transfers between buyers and sellers [7]. This technology gives organizers greater control over ticket circulation and prevents price manipulation in the secondary market [8].

Implementing tickets as NFTs, as unique blockchain-based tokens, offers an innovative solution to the problems inherent in traditional ticketing systems, such as ticket counterfeiting and price opacity in the secondary market [9]. NFTs can be customized with ERC-721 to create immutable digital tickets that are linked to a specific wallet address [10]. This process allows each ticket to be associated with the buyer's data, permanently stored on the blockchain. It reduces fraud and gives event organizers visibility and control over ticket distribution [11].

Developing this smart contract also requires a Decentralized Application (DApp). The DApp interface allows interaction with decentralized applications built on blockchain technology [12]. The DApp interface also displays data from the blockchain and facilitates decentralized transaction processes, eliminating the need for traditional logins as identities are managed through private keys in crypto wallets [13]. This technology is typically developed using libraries like ethers.js or web3.js, which allow direct integration between the frontend interface and the blockchain, often using a JavaScript framework like React to support a dynamic and responsive user experience [14].

The blockchain smart contract explained in previous research can be used to design and develop an online music concert ticket transaction system based on blockchain smart contracts. This system would add transaction detail features to display transparent transactions and validation features when downloading tickets. If the ticket has not been downloaded, then the funds will be held by the smart contract escrow. Developing tickets in the form of NFTs is also the basis for this research.

II. METHODS

The system requirements analysis consists of hardware and software components. The hardware includes a laptop used for testing purposes. The software components include Windows 11 as the operating system, Solidity as the programming language with supporting libraries such as Web3.js and Ethers.js, Visual Studio Code as the code development environment, and meta mask as the access gateway to the blockchain network.

A. System Development Method

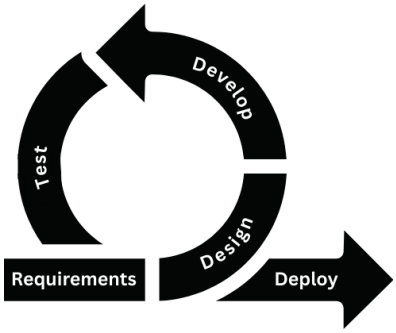

Figure 1. System Development Method

A system development method is a series of procedures, techniques, and steps in designing, building, testing, and implementing an information system or software. Figure 1 Agile Model represents an iterative and flexible approach to software development. Unlike the sequential nature of the Waterfall model, Agile breaks down the development process into short, repeatable cycles known as sprints. Each sprint typically includes the phases of Requirement gathering, Design, Development, Testing, and Deployment, executed in a collaborative and incremental manner [15].

The Requirement phase in Agile involves continuous communication with stakeholders to define and prioritize user stories or product backlog items. Requirements are not fixed upfront; instead, they evolve based on ongoing feedback. In the Design phase, lightweight and adaptive designs are created to meet the specific needs of the current sprint, often using diagrams or mockups for clarity and speed.

During the Development phase, the team builds functional software components that align with the sprint goals. Agile promotes continuous integration and frequent code commits to ensure progress and quick issue resolution. The Testing phase is integrated into the sprint cycle, where unit testing, integration testing, and user testing are conducted concurrently with development to detect bugs early and validate features.

Finally, in the Deployment phase, the software increment is released—either to production or a staging environment—for stakeholders to review and provide feedback. This cycle then repeats in the next sprint, refining and expanding the system with each iteration.

B. Blockchain Workflow

The blockchain workflow involves a verification and consensus process that ensures every transaction is recorded transparently and securely in a decentralized network. Blockchain's security and reliability come from its structure, which connects blocks to each other, the consensus mechanism, which prevents manipulation, and its decentralized nature, which ensures that no single entity has complete control over the data [16].

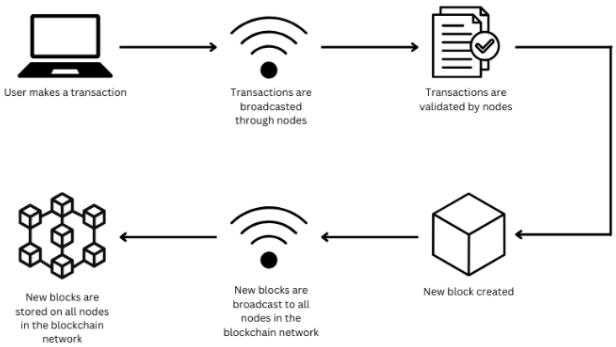

Figure 2. Blockchain Workflow

The blockchain workflow begins when a transaction is made. This transaction is then broadcast to the blockchain network through various nodes, which are computers participating in the network. Each node that receives the

transaction will perform a validation process to ensure the transaction is valid. Once the transaction is successfully validated, it will be combined with other validated transactions in a new block. This block is given a unique mark or hash that serves as a special identity, which also connects it to the previous block, forming a chain of transaction blocks. This new block is then broadcast to the entire blockchain network so that each node receives the same copy of the block. Next, all nodes in the network store the new block, adding it to their blockchain.

C. System Processing Flow

System Processing Flow is a representation of the sequential steps or stages that occur within a system, beginning with the input provided by the user and culminating in the output generated by the system. This flow describes how the system responds to user actions, processes data, executes programmed logic, and produces expected results. By illustrating the end-to-end process, the system processing flow helps ensure that each function is executed correctly, efficiently, and according to predefined rules. This is especially important in systems involving secure transactions, such as blockchain-based ticketing platforms, where each stage must be verifiable and auditable. The following figure presents a detailed flowchart that describes the overall system processing flow.

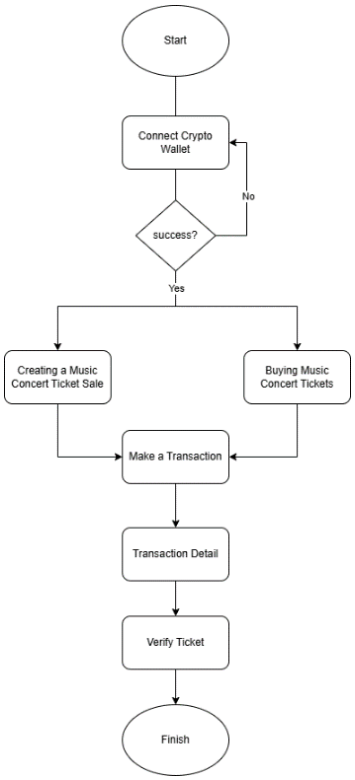

Figure 3. System Processing Flow

Figure 3 illustrates the step-by-step methodology applied during the development of the blockchain-based ticketing system, ensuring that each phase is completed systematically. The system development flow begins when the user connects their crypto wallet to initiate interaction with the music concert ticketing system. At this stage, users must choose between two primary actions: creating a music concert ticket sale or buying music concert tickets. If the user is an event organizer or seller, they proceed by creating a ticket sale, which involves setting up event details, uploading concert posters, determining ticket prices, and specifying the sales period. On the other hand, if the user is a buyer, they browse the available concerts and select the tickets they wish to purchase. Regardless of the chosen path, both actions lead to making a transaction. For sellers, this transaction may involve the payment of a listing or service fee, while for buyers, it involves purchasing the selected ticket. After the transaction is completed, the system moves to the transaction detail phase, where all relevant information regarding the transaction, including ticket and buyer data, is recorded. Subsequently, the system proceeds to the ticket verification phase to ensure the authenticity and validity of the ticket associated with the transaction.

The following is a detailed explanation of the flowchart illustrated above, which outlines the sequential steps involved in the system's operation. This explanation provides a comprehensive overview of each process stage, highlighting the interactions between the user and the system, decision points, and the resulting outcomes based on user actions. By breaking down the flowchart step by step, it becomes easier to understand how the system manages ticket sales and purchases, ensures secure transactions, and maintains data integrity throughout the entire process.

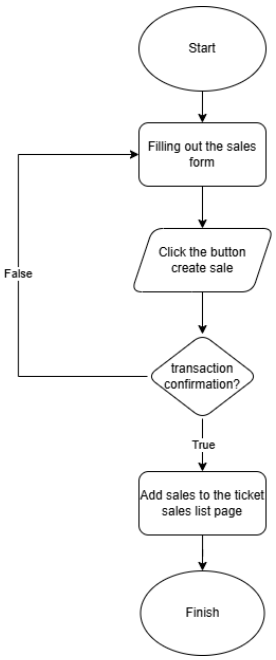

Figure 4. Create Sale Flowchart

Figure 4 illustrate user as a ticket seller, is required to complete a structured form that captures detailed information regarding a music concert event. The form collects critical event data, including the event name, organizer, event date, ticket price, ticket sale deadline, and digital uploads of both

the event poster and the ticket image intended for distribution to buyers. Upon completion of the form, the user initiates the sale creation process by selecting the “Create Sale” option. This action prompts the user’s connected cryptocurrency wallet (e.g., MetaMask) to confirm a blockchain transaction. This transaction may involve a ticket creation fee and is responsible for invoking the corresponding smart contract function to record the ticket sale details onto the blockchain ledger.

If the user confirms the transaction, the smart contract processes the request, and the ticket sale data is permanently stored on the blockchain. As a result, the ticket becomes publicly available on the platform for prospective buyers, and its status can be monitored through the user interface by both the seller and potential purchasers.

Conversely, if the user rejects the transaction request, the sale creation process is terminated, and no data is committed to the blockchain. Consequently, the corresponding ticket listing is not displayed on the platform, thereby preserving the integrity of published ticket sales and ensuring that only validated and blockchain-confirmed data are visible to users.

This mechanism ensures a decentralized, transparent, and secure approach to digital ticketing for music events by leveraging blockchain-based smart contracts.

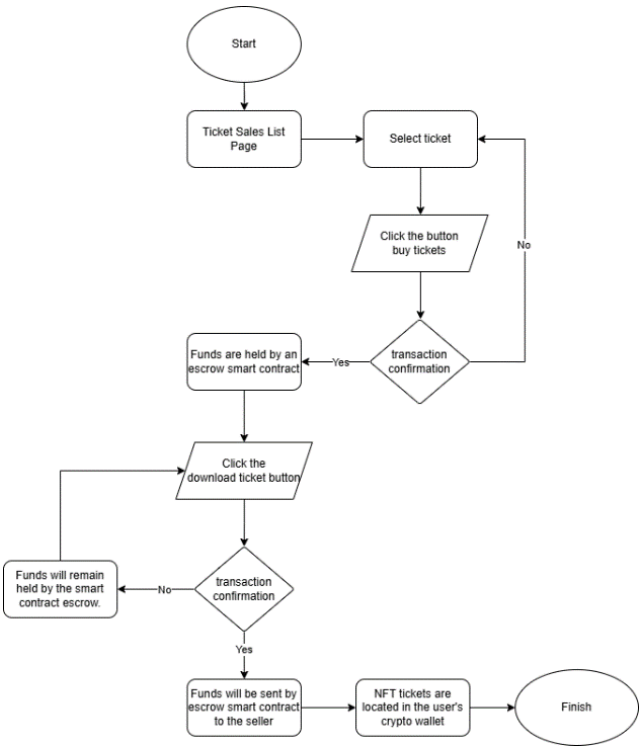

Figure 5. Buy and Download Ticket Flowchart

Figure 5 illustrate flowchart above illustrates the step-by-step process of purchasing and downloading NFT-based concert tickets in a blockchain-integrated ticketing system. The process begins when the user accesses the Ticket Sales List Page, where a list of available concert tickets is displayed along with relevant event details. The user then selects a desired ticket and initiates the purchase by clicking the "Buy Tickets" button. At this point, the system prompts a transaction confirmation via the user’s connected crypto wallet. This confirmation is essential to authorize the transfer of funds. If the user chooses not to proceed, the transaction is canceled, and the system redirects them back to the ticket list without recording any data on the blockchain.

If the transaction is confirmed, the system triggers a smart contract function that places the buyer’s payment into a secure escrow. This escrow mechanism ensures that funds are temporarily held in a neutral smart contract address, preventing immediate transfer to the seller until the buyer successfully receives the ticket. Once escrow is active, the system allows the user to proceed by clicking the "Download Ticket" button, which initiates the second phase of the transaction.

At this stage, a second wallet prompt appears for final confirmation. If the user rejects this confirmation, the system halts the process, and the funds remain securely locked in the escrow smart contract. However, if the user approves the transaction, the smart contract automatically releases the funds to the seller and simultaneously transfers the NFT ticket to the buyer's crypto wallet. The NFT contains a unique identifier and metadata that ensures the authenticity and ownership of the digital ticket. This final step completes the ticketing transaction, providing a tamper-proof, transparent, and verifiable proof of ownership on the blockchain, thereby significantly reducing the risks of fraud, ticket duplication, and unauthorized resale.

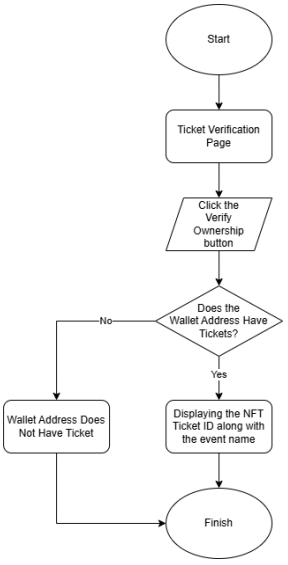

Figure 6. Ticket Verification Flowchart

Figure 6 illustrate the flowchart for verifying NFT-based ticket ownership through a decentralized mechanism involving a user’s connected cryptocurrency wallet. This verification process is fundamental to ensuring the authenticity of ticket holders by validating the presence of a valid tokenized ticket (NFT) on the blockchain before access to event-related functionalities is granted.

The process is initiated when the user accesses the Ticket Verification Page, a user interface specifically designed for

the ownership validation procedure. Within this interface, the user is instructed to select the "Verify Ownership" button. This interaction serves as the entry point for the system to initiate a blockchain-based query. Subsequently, the system retrieves the currently connected wallet address via the user's Web3-enabled wallet. It then conducts a verification process by querying the deployed smart contract or blockchain ledger to ascertain whether any valid NFT-based tickets are associated with the provided address.

In the event that a valid ticket is detected, the system proceeds to confirm ownership by displaying the unique NFT Ticket ID along with the corresponding event name. This feedback mechanism serves as both a confirmation of authenticity and a means of associating the digital asset with a specific event, thus enabling further operations such as event access or digital check-in.

Conversely, if the verification process reveals that no valid ticket is linked to the wallet address, the system generates a notification stating that the wallet does not possess any valid ticket.

D. General System Overview

The development of a smart contract system for concert tickets has been designed and implemented according to needs. In its implementation, smart contracts are developed using the Solidity programming language and integrated through the Visual Studio Code application as a development environment.

This system uses JavaScript libraries, such as ethers.js or web3.js, to interact with smart contracts on the front end. These libraries support communication between web applications and blockchain networks. In addition, several other supporting libraries, such as react, hardhat, openzeppelin, and meta mask, are also used to facilitate the development, testing, and connection of user wallets.

Figure 7 illustrates the overall architecture of the blockchain-based concert ticketing platform that combines both off-chain and on-chain components. Users interact with a web application built using React.js to perform operations such as selling and purchasing concert tickets. The web app connects to the user's crypto wallet through the Web3.js library, enabling secure interactions with Ethereum smart contracts based on the ERC-721 token standard.

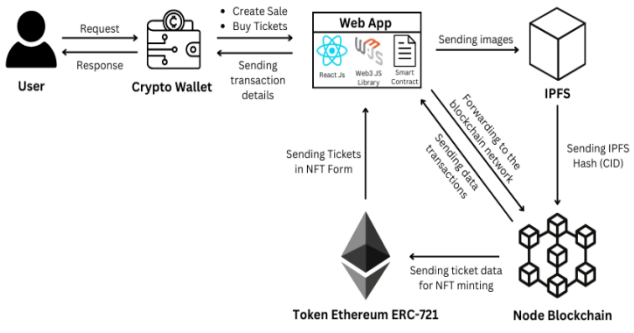

Figure 7. General Architecture Overview

When a user initiates ticket creation, the concert poster image is uploaded to the InterPlanetary File System (IPFS) via a third-party service such as Pinata, ensuring decentralized and persistent storage. This process generates a unique IPFS Hash (CID), which is used to reference the associated ticket metadata. The IPFS Hash (CID), along with other ticket details, is then forwarded from the web application to the blockchain network through a connected node. The node validates and processes the transaction, ultimately minting the NFT representing the concert ticket.

The minted NFT is transferred to the user's wallet as proof of ownership, ensuring authenticity, transparency, and immutability through blockchain technology. This architecture provides a robust and secure solution for managing digital concert tickets in a decentralized environment.

E. Transaction Fee

Transaction fee or transaction costs in the Ethereum blockchain, known as gas prices are outlined as a key factor in the transaction ecosystem. These fees are determined by two main elements, the amount of gas required to execute a transaction and the gas price set by the user. Gas itself is a unit of measurement for transaction complexity, the more complex the transaction, the higher the amount of gas required [17]. The calculation of transaction fees follows Equation 1.

$$\text{Transaction fee} = \text{gas used} \times \text{gas price} \quad (1)$$

Transaction fee is calculated by multiplying the amount of gas consumed (gas used) by the price per unit of gas (gas price). Gas measures the computational resources required for a transaction, while the gas price determines the cost per unit of gas, typically in Gwei. This fee compensates network validators for processing the transaction.

F. ERC Token Comparison

ERC (Ethereum Request for Comments) tokens are a technique used in the Ethereum network to create and manage digital tokens that run on the Ethereum blockchain. There are many types of ERC tokens, and each has its function. The following are some types of ERC tokens on the Ethereum network whose functionality is a comparison [18].

TABLE I
ERC-721 AND ERC-1155 TOKEN COMPARISON

Token	Function	Use Case	Gas Estimation	System Compliance
ERC-721	NFT	Tickets, Certificates and Artwork	Average	Yes
ERC-1155	Fungible + NFT	Lots of Tickets, Game Item and Voucher	Average	Yes, for mass production

Table I presents a comparison between several ERC token standards that enable functionalities relevant to this research. The analysis is conducted across multiple aspects, including core functions, commonly implemented use cases, gas efficiency in transactions, and the overall suitability of each standard with the system proposed in this study. Based on a comprehensive evaluation, the ERC-721 token is identified as the most appropriate standard to be implemented in a digital concert ticketing system. This is due to its non-fungible nature, which allows each ticket to be uniquely represented, making it non-interchangeable and resistant to counterfeiting. This characteristic ensures the authenticity and exclusivity of each issued ticket.

However, while ERC-721 provides a unique digital identity for assets such as tickets, the standard still allows NFTs to be transferred to other wallet addresses by default. This introduces potential risks of misuse, such as unauthorized resale or ticket usage by individuals other than the original owner. Therefore, it is essential to implement a reliable ownership verification mechanism—either through wallet address matching at the point of use or through the integration of additional authentication systems. With such ownership verification in place, the system can not only ensure that tickets are used solely by their rightful owners but also enhance the overall security and user trust in the platform.

III. Result and Discussion

This section presents the results of the research, including outcomes from the development process as well as the application of previously discussed theoretical concepts. The findings are analyzed to evaluate the system's performance, and the relationship between the implemented solutions and the theoretical foundation is also discussed in this chapter.

A. Black Box Testing

Black Box testing is performed independently, focusing on evaluating the functionality of the system without knowledge of the internal workings of the smart contract. This approach involves testing several key features of the ticketing system to ensure that the contract functions as intended in various scenarios. The goal is to verify the correctness of the external behavior of the contract, ensuring that it aligns with the intended design and business logic.

TABLE II
SYSTEM TESTING USING THE BLACK BOX METHOD

Testing	Scenario	Expected Output	Actor	Results
Create Sale	User creates a music concert ticket sale by filling in all the required form fields.	Ticket appears in the ticket list with correct metadata.	Seller	Success
Buy Ticket	User purchases a ticket listed on the ticket sales page with sufficient funds in their crypto wallet.	Ticket is marked as sold; ownership is recorded; funds are held by the escrow smart contract.	Buyer	Success
Download Ticket	User downloads a music concert ticket after purchasing it.	NFT is minted; ticket download is enabled; funds are released from escrow to the seller.	Buyer	Success
Verify Ticket	User presses the verify ownership button with the concert ticket in their crypto wallet.	Ticket ID and wallet address match the current user; event name is displayed.	Buyer	Success

Table II presents the results of system testing performed using the Black Box method, which evaluates the system's external behavior without consideration of its internal implementation. This testing approach simulates real user interactions to ensure that the core functionalities operate according to specified requirements.

The first scenario involves the Create Sale function, where the seller completes all required form fields to initiate a music concert ticket sale. The system successfully adds the ticket to the listing with accurate metadata, indicating correct processing of user input and data rendering.

In the second test, the Buy Ticket function is evaluated. A buyer with sufficient cryptocurrency funds initiates a purchase of an available ticket. The system correctly updates the ticket status to "sold," records ownership on the blockchain, and transfers the payment to an escrow smart contract. This verifies the correct operation of transactional logic and fund handling mechanisms.

The third test assesses the Download Ticket feature, which is triggered after a successful purchase. The system mints the ticket as a non-fungible token (NFT), enables the download functionality, and releases the previously held funds from escrow to the seller. This confirms proper execution of the NFT minting process and financial settlement.

The final scenario tests the Verify Ticket feature, in which a user who possesses a valid ticket initiates ownership verification. The system successfully matches the wallet address and ticket ID, and subsequently displays the event name, validating the integrity of the ticket verification mechanism.

TABLE III
VALIDATION TESTING USING THE BLACK BOX METHOD

Testing	Description	Expected Output	Actor	Results
Create Sale	User submits a form with incomplete fields.	System shows validation error; form not submitted.	Seller	Success
Create Sale	User sets a sale end date after the event date.	System shows validation error; sale end date must be before event date.	Seller	Success
Buy Ticket	User tries to buy a ticket with insufficient funds.	Transaction is rejected; error message about insufficient balance shown.	Buyer	Success
Download Ticket	User cancels the ticket download after purchase.	NFT is not minted; download is not triggered; funds remain held in the smart contract (escrow).	Buyer	Success
Verify Ticket	User tries to verify a ticket without owning any ticket in the wallet.	No ticket ID or event name shown; verification fails with appropriate feedback.	Buyer	Success

Table III presents the results of validation testing using the Black Box method, focusing on scenarios in which users provide invalid or inappropriate input. This type of testing ensures that the system responds appropriately to user errors without requiring access to or knowledge of the internal structure of the source code.

The first and second scenarios involve testing the Create Sale feature, where the user submits an incomplete form and sets a sale end date that exceeds the event date. In both cases, the system successfully displays validation messages and prevents the process from continuing, indicating that input validation mechanisms function as expected.

The third scenario tests the purchase of a ticket with insufficient funds in the user's crypto wallet. The system correctly rejects the transaction and displays a warning message about the insufficient balance, demonstrating proper transaction validation.

In the fourth scenario, the user cancels the ticket download process after purchasing. The results show that the system does not mint the NFT, the download is not triggered, and the funds remain held by the escrow smart contract. This indicates that the system is capable of safely handling process interruptions.

The fifth and final scenario tests ticket ownership verification when the user does not possess a ticket in their wallet. The system accurately fails the verification and does not display the ticket ID or event name, proving that ownership validation is correctly enforced.

Overall, the validation testing results show that the system is capable of handling various types of invalid input effectively, maintaining data integrity, and providing appropriate user feedback.

B. Transaction Fee of Smart Contract Deploy

Every process of deploying a smart contract to a blockchain network requires a transaction fee. The amount of the transaction fee depends on the complexity of the contract and the network traffic conditions when the transaction is made. The calculation of the amount of the transaction fee can be formulated as in Equation 1, which is the basis for calculating the cost based on gas consumption and gas price at the time of the transaction.

TABLE IV
TRANSACTION FEE OF SMART CONTRACT DEPLOY

Transaction Detail	
Transaction Fee	0.37667261451875897 ETH
Gas Used	2,831,126 gas unit
Gas Price	0.000000133046927095 ETH

Table IV is detailed information from the execution of a transaction to buy a music concert ticket. Based on the transaction fee calculation formula that has been described in Equation 1, the transaction fee that must be paid when deploying to the sepolia testnet network is 0.37667261451875897 ETH. This value is obtained based on the gas used value of 2,831,126 gas units and the gas price value of 0.000000133046927095 ETH. All detailed transaction information is obtained from etherscan.

The system was tested on the Ethereum Sepolia Testnet. In a real-world deployment on the Ethereum Mainnet, gas fees are expected to be significantly higher due to the inherently high gas usage per transaction and fluctuating gas prices driven by network congestion. Additionally, using external libraries such as OpenZeppelin—while beneficial for security and reliability—can increase contract complexity and bytecode size, leading to additional gas costs. For instance, a typical ticket purchase is estimated to consume between 50,000 and 100,000 gas units, equivalent to approximately $2 to $5, depending on current gas prices. Creating five tickets in one transaction may consume around 300,000 gas units, or about 0.015 ETH (approximately $30 at 50 gwei). To mitigate such costs and improve scalability, future implementations could consider deploying on Layer-2 networks like Polygon, Arbitrum, or Optimism.

C. Smart Contract with Slither Security Analysis

Smart contract security analysis was performed using Slither based on common security principles used in blockchain application development. The evaluation focused on key aspects such as reentrancy, overflow/underflow, potential front-running, access restrictions on critical public functions, input validation, and gas efficiency. The analysis

was performed on Solidity version 0.8.20 code, which includes overflow and underflow protection by default.

TABLE V
SLITHER SECURITY ANALYSIS – CREATEMULTIPLETICKETS FUNCTION

Function createMultipleTicket	
Aspect	**Status**
Reentrancy	Safe
Overflow	Safe
Front-running	Safe
Access Control	Public
Input Validation	Checked
Gas Efficiency	Efficient

Table V presents a Slither security analysis of the createMultipleTickets function. The function is confirmed to be safe against reentrancy as no external contract calls (e.g., call, transfer) are made before important state changes. The use of Solidity version 0.8.20 ensures arithmetic safety, preventing overflow and underflow without additional libraries. No front-running risks are identified since ticket creation does not involve competitive behavior or public auction mechanisms. The function is publicly accessible by design to allow sellers to issue tickets, and all input parameters (such as ticket price, expiration date, and poster CID) are properly validated. Additionally, the function executes gas efficiently, creating a fixed number of tickets in a controlled loop.

TABLE VI
SLITHER SECURITY ANALYSIS – BUYTICKET FUNCTION

Function buyTicket	
Aspect	**Status**
Reentrancy	Safe
Overflow	Safe
Front-running	Minor
Access Control	Public
Input Validation	Checked
Gas Efficiency	Efficient

Table VI presents a Slither security analysis of the buyTicket function within the smart contract. While automated auditing tools such as Slither and Mythril could not be utilized due to technical constraints in the local development environment, this Slither evaluation addresses key vulnerability patterns commonly identified in Ethereum smart contracts. The analysis focuses on well-established security aspects, including reentrancy protection, arithmetic safety (overflow/underflow), front-running susceptibility, input validation, gas efficiency, and access control enforcement. Each element has been carefully reviewed to ensure the function adheres to best practices and mitigates known risks. In particular, the function's public accessibility is acknowledged while confirming that adequate input checks and gas-efficient operations are in place.

TABLE VII
SLITHER SECURITY ANALYSIS – MARKASDOWNLOAD FUNCTION

Function markAsDownload	
Aspect	**Status**
Reentrancy	Acceptable
Overflow	Safe
Front-running	Safe
Access Control	Buyer-only
Input Validation	Checked
Gas Efficiency	Efficient

Table VII presents the results of the Slither security analysis of the markAsDownloaded function. The function's reentrancy status is marked as acceptable. Although it performs an external call (seller.call{value: ...}) to transfer funds, this call is made after all relevant state changes, which reduces the risk of reentrant behavior. Nevertheless, in critical production systems, it is recommended to move such calls to the final line of the function to fully adhere to the checks-effects-interactions pattern. The function benefits from Solidity's built-in overflow protection and does not include arithmetic operations that pose overflow/underflow risks. It does not expose any front-running vulnerabilities, as the action can only be executed by the verified ticket buyer. The access control is appropriately restricted to the ticket buyer, and all relevant input checks—including ownership, download status, and escrow balance—are validated before execution. Lastly, the function is considered gas efficient, performing a direct value transfer and NFT minting with minimal computational complexity.

TABLE VIII
SLITHER SECURITY ANALYSIS – UPDATEEXPIREDTICKETS FUNCTION

Function updateExpiredTickets	
Aspect	**Status**
Reentrancy	Safe
Overflow	Safe
Front-running	Safe
Access Control	Public
Input Validation	Checked
Gas Efficiency	Inefficient

Table VIII outlines the results of the Slither security analysis of the updateExpiredTickets function. The function is considered safe from reentrancy, as it does not perform any external contract calls before updating internal state, and any external value transfers (such as refunds) occur after all state transitions. The use of Solidity 0.8.20 ensures protection from arithmetic overflow or underflow issues. There are no front-running concerns, as the function logic is not susceptible to transaction ordering manipulation and does not involve competitive conditions.

This function is publicly accessible, which is intentional to allow anyone to trigger the expiration update process. All input conditions—such as verifying whether a ticket is expired, sold, or downloaded—are properly validated before executing a refund or marking the ticket inactive. However, the primary concern identified by Slither lies in its gas efficiency: the function iterates over an unbounded list of issued tickets, which can become problematic as the number of tickets increases. This looping pattern may lead to block

gas limit issues and failed on-chain execution. To address this, a more scalable approach is recommended, such as indexing tickets by event, implementing batched updates, or adopting an event-driven expiration mechanism.

TABLE IX
SLITHER SECURITY ANALYSIS – SETADMIN FUNCTION

Function setAdmin	
Aspect	**Status**
Reentrancy	Safe
Overflow	Safe
Front-running	Safe
Access Control	onlyOwner
Input Validation	Not Validate
Gas Efficiency	Efficient

Table IX summarizes the Slither security analysis of the setAdmin function. The function is classified as safe from reentrancy, as it does not involve any external calls or complex state transitions that could be exploited through reentrant behavior. It is also secure from overflow and underflow vulnerabilities, given the absence of arithmetic operations. No front-running risks are present, as the function only updates the admin address and does not rely on competition or timing-sensitive logic.

Access control is properly enforced through the onlyOwner modifier, ensuring that only the contract owner can execute this administrative operation. However, Slither identifies a lack of input validation, as the function does not check whether the new admin address is valid (e.g., not a zero address). Despite this, the function is considered highly gas-efficient, executing only a single storage write with no loops or heavy computation, making it well-suited for on-chain deployment.

D. Purchase and Download Restriction on Wallet Address

The smart contract enforces a purchase limitation policy, allowing each wallet address to acquire only one ticket per event. This restriction is implemented using a two-dimensional mapping structure that associates wallet addresses with specific event identifiers, tracking both the purchase and download history. Before executing a ticket purchase or download, the contract performs a conditional check to ensure the requesting address has not previously completed a transaction for the same event. If the condition is not met, the transaction is programmatically rejected on-chain. This preventive measure effectively mitigates the risk of ticket hoarding, resale abuse, and unauthorized bulk acquisition, thereby promoting fairness and transparency in the ticket distribution process.

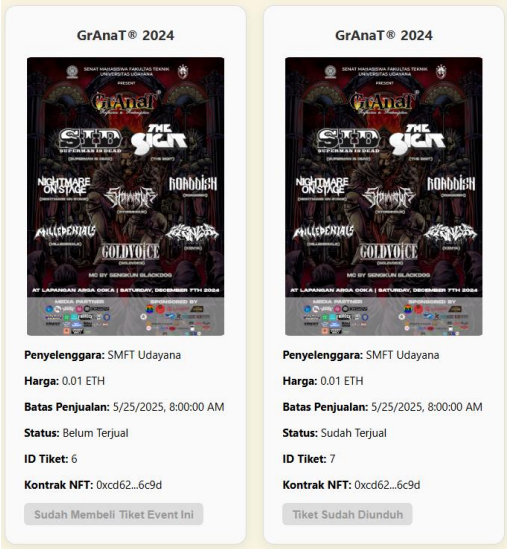

Figure 8. Purchase and Download Restriction on Wallet Address

Figure 8 illustrates the implementation of a purchase and download restriction mechanism in the smart contract, which limits each wallet address to a single ticket purchase and download per event. This restriction is strategically enforced to prevent bulk ticket purchases and unauthorized resale or ticket scalping activities. By recording the purchase and download history of each wallet address against specific event identifiers, the smart contract ensures that any subsequent attempts to purchase or download additional tickets for the same event from the same address are programmatically rejected. This approach contributes significantly to maintaining fair ticket distribution, enhancing system transparency, and reducing the risk of market manipulation by automated bots or scalpers.

E. Transaction Transparency

To ensure trust between users and the system, transaction transparency is an important aspect in implementing blockchain-based tickets. This section discusses how the system displays and ensures the visibility of every transaction related to ticket creation and purchase through an immutable blockchain log displayed in the system.

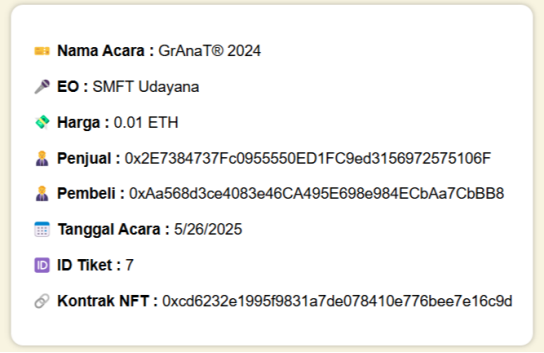

Figure 9. Transaction Transparency

Figure 9 illustrates a transaction detail interface between the ticket seller and buyer as part of the blockchain-based

ticketing system. This interface serves as tangible evidence of transactional transparency, where essential data resulting from seller and buyer—such as wallet addresses, ticket price, and contract information—are displayed clearly to users.

By presenting this data openly, the system reinforces its commitment to transparency and trustworthiness. Every transaction is recorded immutably on the blockchain and reflected in the user interface without the possibility of alteration or deletion. This design not only ensures accountability for both parties involved but also demonstrates how blockchain technology can be effectively leveraged to prevent fraud and promote verifiable digital ownership within decentralized ticketing platforms.

F. NFT Ownership Transferability and Ticket Verification Security

Although ERC-721 tokens can be transferred by default, the implemented smart contract overrides the transfer function to emulate the behavior of Soulbound Tokens (SBT). Smart contract records and tags the ownership of the ticket with the first buyer's address, ensuring that ownership remains tied to the original buyer's wallet and preventing unauthorized resale. This restriction supports secure access control by permanently binding the ticket to the first owner.

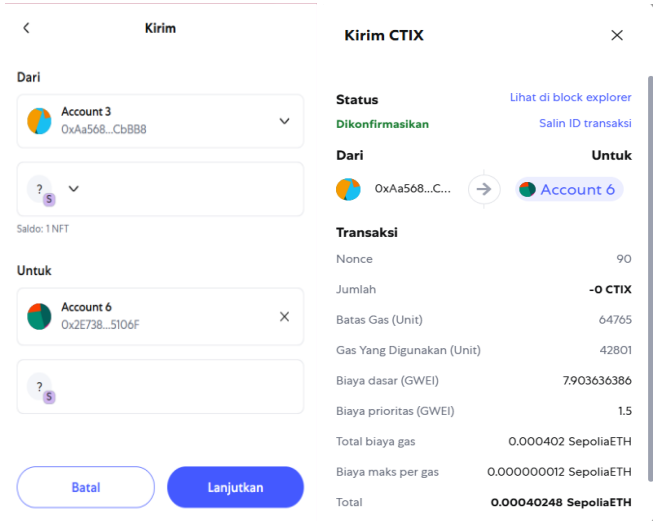

Figure 10. Transfer Ticket

Figure 10 illustrates the transfer of an ERC-721 NFT ticket, where the ticket is successfully transferred to another user who did not originally purchase it. This transfer highlights a potential vulnerability of ERC-721 tokens, where ownership can be changed after the initial purchase, allowing for unauthorized redistribution of the ticket. The figure demonstrates how the default behavior of the smart contract, with account 0x2E7384737Fc0955550ED1FC9ed3156972575106F receiving the NFT ticket sent by account 0xAa568d3ce4083e46CA495E698e984ECbAa7CbBB8, which allows token transfer, can be exploited to circumvent the intended restriction of limiting ticket ownership to the original buyer. This could potentially facilitate unauthorized resale or secondary market activities.

During the verification process, the system cross-references the ticket's token ID with the owner's wallet address using on-chain queries. The ticket is treated as valid only when ownership is confirmed and the token has not been marked as used. This approach ensures that each NFT ticket is verifiable, unique, and cannot be reused fraudulently.

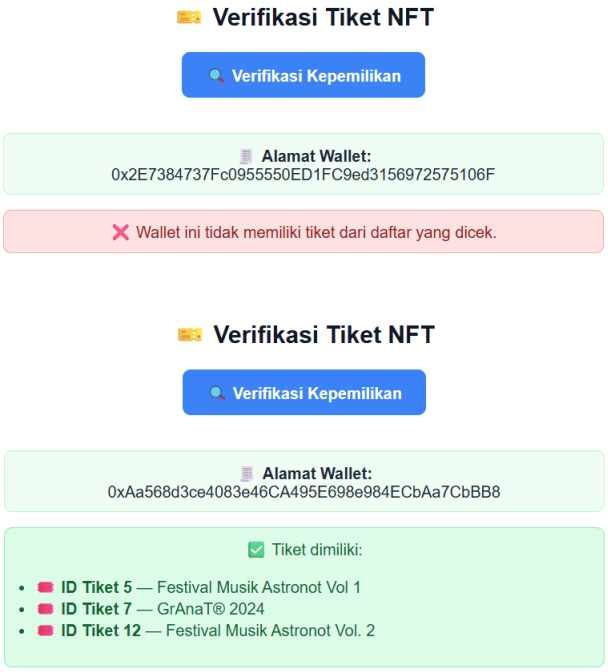

Figure 11. Ticket Verification Security

Figure 11 illustrates the implementation of ticket verification security, where account 0x2E7384737Fc0955550ED1FC9ed3156972575106F is verified as not owning the NFT ticket, while account 0xAa568d3ce4083e46CA495E698e984ECbAa7CbBB8 is confirmed as the rightful owner of the NFT ticket. This verification process is a key component in reducing the risk of unauthorized ticket resale at inflated prices, which often occurs in the secondary market. By leveraging on-chain data to cross-reference ticket ownership with the original purchaser's wallet, the system ensures that only the original purchaser of the NFT ticket has legitimate ownership. Furthermore, the system prevents subsequent attempts to transfer or resell the ticket, thereby strengthening the security and fairness of the ticketing system. The integration of such a robust verification mechanism not only prevents fraudulent activities but also increases transparency and trust in the distribution process, ensuring that tickets remain tied to their rightful owners and cannot be exploited for profit by unauthorized third parties.

G. On-Chain and Off-Chain Ticket Metadata

The ticket metadata is managed using a hybrid storage approach that combines both on-chain and off-chain mechanisms to balance data integrity, scalability, and cost-efficiency.

TABLE X
TICKET METADATA

Field	Type	Example Value	Description
ticketId	uint256	7	Unique ID of the ticket
eventName	string	GrAnaT® 2024	Event name
eventOrganizer	string	SMFT Udayana	Organizer of the event
posterUrl	string	QmXdpg7...ar6	IPFS hash of the event poster
ticketImageUrl	string	QmRKkNN...TTL	IPFS hash of the ticket image
price	uint128	10000000000000000	Ticket price in Wei
eventDate	uint64	1748131200	UNIX timestamp of the event date
saleEndDate	uint64	1748217600	Ticket sale expiration timestamp
seller	address	0x2E7384...	Wallet address of the ticket creator

Table X shows the results of storing ticket metadata well through On-Chain and Off-Chain approaches. Important ticket-related information, such as ticketId, price, eventName, purchaseDate, and buyer address, is stored directly on-chain in the smart contract. By permanently recording this data on the blockchain, the system ensures transparency, verifiability, and resilience to unauthorized changes or tampering.

However, due to storage limitations and the high cost of storing large data on the blockchain, some metadata components, especially image files such as event posters and digital ticket images, are stored off-chain using the InterPlanetary File System (IPFS). Off-chain storage via the InterPlanetary File System (IPFS) produces a unique content identifier (CID or IPFS hash), which is then stored on-chain to ensure reliable retrieval and data integrity.

IV. CONSLUSION

Conclusions that can be drawn from the development process of the Implementation of Blockchain Smart Contract for Online Concert Ticket Transactions Based on NFTs system. The blockchain smart contract for the purpose of music concert ticket transaction activities in this study was successfully built using the Solidity programming language and can work well according to the needs of music concert ticket transaction activities and has been broadcast publicly on the Ethereum Sepolia testnet with the smart contract address 0xCD6232e1995F9831a7De078410e776BeE7E16c9D [19] for the TicketSale.sol smart contract and the smart contract source code can also be accessed via the GitHub repository [20].

The use of NFTs ensures that each ticket is uniquely identified and verifiable on-chain. Ownership is tied directly to the buyer's wallet address, and only this address can access download and verification functions. This prevents ticket duplication and eliminates the possibility of forgery or resale without authorization.

The transaction detail feature on the blockchain smart contract system has been successfully implemented and creates security and transparency for users in NFT-based Online Music Concert Ticket Buying and Selling Transactions. Where the transaction details are displayed on the system.

The ticket verification feature on the music concert ticket buying and selling transaction system has been successfully implemented and functions as it should in ensuring the authenticity of the ticket. With this ticket verification feature, only valid and legally owned tickets are displayed on the system.

REFERENCES

[1] J. Margaretha and A. Voutama, "Perancangan Sistem Informasi Pemesanan Tiket Konser Musik Berbasis Web Menggunakan Unified Modeling Language (UML)," *JOINS (Journal Inf. Syst., vol. 8, no. 1, pp. 20–31, 2023, doi: 10.33633/joins. v8i1. 7107*, 2023.

[2] A. Rozak and T. Srihadiati, "Tinjauan Kriminologis Terhadap Praktik Penipuan Calo Tiket Konser di Indonesia," *INNOVATIVE: Journal Of Social Science Research*, vol. 4, no. 1, pp. 6707–6717, 2024.

[3] M. F. Sinaga, I. A. Putri, W. P. Ningrum, B. R. Setiadji, and M. J. Shofa, "Efektivitas Sistem Penukaran Tiket Konser: Perbandingan Metode Online dan On The Spot," *Jurnal Teknik Industri*, vol. 3, no. 1, pp. 7–13.

[4] Y. D. Nugroho and I. M. Suartana, "Penerapan Teknologi Blockchain pada Sistem Transaksi Aplikasi Point of Sale Berbasis Web," *Journal of Informatics and Computer Science (JINACS)*, pp. 179–188, 2024.

[5] M. Gysel, B. Ford, and L.-H. Merino, "Blockchain-based Event Ticketing," 2023.

[6] S. Rafati Niya, S. Bachmann, C. Brasser, M. Bucher, N. Spielmann, and B. Stiller, "DeTi: A Decentralized Ticketing Management Platform," *Journal of Network and Systems Management*, vol. 30, no. 4, p. 62, 2022.

[7] T. T. Harmanda *et al.*, "Systematic Literature Review of the Use of Blockchain as a Secure Technology in E-Ticketing Systems," in *2024 International Conference on Electrical Engineering and Computer Science (ICECOS)*, 2024, pp. 308–313. doi: 10.1109/ICECOS63900.2024.10791105.

[8] Y. Khan, P. Indoriya, and P. Sharma, "Blockchain enabled Smart Ticketing Solution," *Blockchain enabled Smart Ticketing Solution*.

[9] D. Ghelani, "What is Non-fungible token (NFT)? A short discussion about NFT Terms used in NFT," *Authorea Preprints*, 2022.

[10] P. Cholke, Y. Munde, M. Sayyed, A. Vidhale, and N. Zanwar, "Secure Event Ticketing System Using NFT ERC721 Tokens," in *2024 4th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS)*, 2024, pp. 1401–1407. doi: 10.1109/ICUIS64676.2024.10866537.

[11] A. Guayasamín, W. Fuertes, N. Carrera, L. Tello-Oquendo, and V. Suango, "Blockchain-Enhanced E-Ticket Distribution System to Effective Transactions, Validation, and Audits," in *2024 8th Cyber Security in Networking Conference (CSNet)*, 2024, pp. 52–59. doi: 10.1109/CSNet64211.2024.10851765.

[12] W. Metcalfe, "Ethereum, smart contracts, DApps," *Blockchain and Crypt Currency*, vol. 77, pp. 77–93, 2020.

[13] S. D. S. Saian, I. Sembiring, and D. H. F. Manongga, "A Prototype of Decentralized Applications (DApps) Population Management System Based on Blockchain and Smart Contract," *JOIV: International Journal on Informatics Visualization*, vol. 8, no. 2, pp. 845–853, 2024.

[14] J. Gjorgjev, N. Sejfuli-Ramadani, V. Angelkoska, P. Latkoski, and A. Risteski, "Use Cases and Comparative Analysis of Blockchain Networks and Layers for DApp Development," in *2024 13th Mediterranean Conference on Embedded Computing (MECO)*, 2024, pp. 1–5. doi: 10.1109/MECO62516.2024.10577885.

[15] S. Pargaonkar, "A comprehensive research analysis of software development life cycle (SDLC) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering," *International Journal of Scientific and Research Publications (IJSRP)*, vol. 13, no. 08, pp. 345–358, 2023.

[16] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Business & information systems engineering*, vol. 59, pp. 183–187, 2017.

[17] A. Donmez and A. Karaivanov, "Transaction fee economics in the Ethereum blockchain," *Econ Inq*, vol. 60, no. 1, pp. 265–292, 2022.

[18] Y. Tan, Z. Wu, J. Liu, J. Wu, T. Chen, and K. Lin, "Bubble or Not: An Analysis of Ethereum ERC721 and ERC1155 Non-fungible Token Ecosystem," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024, pp. 1–5. doi: 10.1109/ISCAS58744.2024.10558166.

[19] G. Lingga, 'Smart Contract TicketSale.sol Sepolia Testnet', *Etherscan*.[Online]. Available: https://sepolia.etherscan.io/address/0xCD6232e1995F9831a7De078410e776BeE7E16c9D. [Accessed: 01-May-2025].

[20] G. Lingga, 'TicketingBlockchain: Smart Contract Source Code', *GitHub*.[Online]. Available: https://github.com/Gunglingga/TicketingBlockchain. [Accessed: 30-April-2025].