Blockchain-based Smart Contracts Platform for Transparent and Efficient E-Ticketing

Cláudia Silva
School of Technology, Polytechnic Institute of Castelo Branco, Portugal, 6000-767 Castelo Branco, Portugal

Mónica Costa
School of Technology, Polytechnic Institute of Castelo Branco, Portugal, 6000-767 Castelo Branco, Portugal

Alexandre Fonte
School of Technology, Polytechnic Institute of Castelo Branco, Portugal, 6000-767 Castelo Branco, Portugal

ABSTRACT
Smart contracts, a key innovation of blockchain technology, are digital agreements that automatically execute predefined terms. These contracts are secure, decentralized, transparent, and immutable, making them suitable for a range of sectors, including event ticket sales, where they can streamline transactions and reduce fraud. This project explores using smart contracts for event ticket sales and marketing, aiming to reduce ticket speculation and paper redundancies. A prototype system was developed with Solidity on the Ethereum blockchain, tested locally, and integrated with a Vue.js frontend, a MySQL database, and a simulated cryptocurrency wallet via Ganache. The system is designed to serve stakeholders such as ticket sellers, buyers, and event organizers. The results demonstrate the feasibility of using blockchain for transparent and efficient ticketing processes. While currently limited to a local environment, it provides a foundation for future enhancements and broader adoption. This project highlights blockchain's potential to transform event ticketing, ensuring security, scalability, and user trust.

Keywords
E-ticket, Blockchain, Smart Contract, Ethereum, Cryptocurrency.

1. INTRODUCTION

The rise of cryptocurrencies has presented challenges like security concerns, lack of trust in peer-to-peer transactions, currency duplication, transaction valuations, and data storage. These issues, once managed by centralized systems, have led to the development of blockchain—a decentralized ledger storing transactional information via peer-to-peer network nodes [1]. Information is securely linked in blocks using cryptographic hashes (e.g., SHA-256 or SHA-3).

Blockchain technology's [2] evolution has brought about significant applications such as smart contracts, a digital agreements that automatically execute specific terms when conditions are met, removing the need for intermediaries. Stored on the blockchain, they ensure security and transparency. Key features of smart contracts include decentralization, transparency, and immutability. These traits make smart contracts suitable for sectors like marketing or event ticket sales, streamlining processes, reducing intermediaries, and ensuring secure transactions.

In response to these challenges, the primary objective of this project is to explore the use case of blockchain-based smart contracts (SC) for event ticketing, focusing on ticket marketing and sales to reduce speculation in ticket sales and minimize reliance on paper-based systems. The project integrates smart contracts to build agreements which automatically execute specific terms when certain conditions are met, eliminating the need for intermediaries, ensuring security and transparency. Additionally, the decentralized nature of blockchain allows all parties to view the terms simultaneously, while the immutability of smart contracts ensures that once deployed, they cannot be altered, fostering trust among stakeholders.

Given the complexity of implementing smart contracts and the methodology for developing software components (e.g., ICONIX), this project investigates methodologies and frameworks that guide the design, deployment, and debugging of smart contracts. To facilitate user interactions, a frontend interface was developed using JavaScript [3], TypeScript [4], and Vue.JS [5]. A local database, built with MySQL Workbench [6], recorded essential information required for interactions between the blockchain and the smart contracts. For transaction simulation, Ganache [7] was employed as a simulated cryptocurrency wallet, enabling the execution and local recording of blockchain transactions. The smart contracts were coded in Solidity [8] and deployed using the Truffle suite framework, with unit testing conducted via Visual Studio Code [9].

The main work contributions are summarized as follows:

- **Developing a Smart Contract-based Ticketing System**: A prototype was implemented using Solidity on Ethereum to streamline ticket sales, reduce speculation, and eliminate redundant processes;
- **Integrating Blockchain with Modern Software Tools**: The system combines a Vue.js frontend, a MySQL database, and a simulated cryptocurrency wallet (Ganache) to demonstrate feasibility;
- **Evaluation of Real-World Applications of Smart Contracts**: This project provides some insights into the performance and limitations of SC in ticketing scenarios;
- **Laying of a Foundation for Future Enhancements**: This work identifies areas for improvement, such as scalability and live blockchain integration, to support broader adoption and practical use.

The rest of this paper is organized as follows. Section 2 provides a review of related work. Section 3 outlines the system architecture and introduces interface prototyping. Section 4 covers the development and implementation phase, while Section 5 presents and discusses the results. Finally, Section 6 concludes the paper, highlighting the key findings.

2. STATE OF ART OR LITERATURE REVIEW

This review was conducted using the PRISMA methodology PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses), which is a standardized approach reporting systematic reviews and meta-analyses [10].

In brief, the IEEE Xplore, Springer Link, Research Gate, ACM DL and Science Direct databases were considered the most suitable sources for extracting and identify relevant articles between 2017 and 2024. Among a total of 120 articles, only 4 articles were identified and screened for inclusion in this review. Further methodological details are going to be provided in a separate publication by the authors.

First, the study *Blockchain and smart contracts* [11] primarily focus on the general technical aspects and challenges of blockchain and smart contracts rather than their specific applications in e-ticketing. However, it identifies potential avenues where blockchain technology could address key challenges in the ticketing industry, such as fraud prevention, transparency, and efficiency.

In *Smart Contract Designs on Blockchain Applications* [12], the authors evaluate smart contract architectures for efficient blockchain data indexing in ridesharing scenarios. They compare Sparse and Catalog designs, concluding that while the Catalog design is more complex, it offers significantly improved query efficiency over Sparse.

In turn, the article *A Blockchain-Based Ticket Sales Platform* [13] presents a decentralized ticket sales app using NFTs on Ethereum and Avalanche. The platform aims to enhance enhancing transparency, security, and cost-efficiency in transactions. Avalanche is favoured for its lower costs and scalability.

Finally, *Assessment of E-Ticketing Technology in Concert Websites: A Review of Benefits, Profits, and Customer Satisfaction* [14] assesses the impact of e-ticketing on concert websites. The article highlights benefits for companies, such as increased operational efficiency, and improved customer satisfaction through streamlined ticket purchasing processes.

3. SYSTEM DESIGN

System design is a critical aspect of engineering and computer science, involving the definition of its architecture, components, modules, interfaces, and data to satisfy specified requirements.

In subsection 3.1, the essential components of the system, are reviewed providing a comprehensive overview of its architecture and functionality. Afterwards, in subsection 3.2, interface prototyping, is introduced to refine the user experience. Various modeling tools are employed to create prototypes that visually demonstrate the user interface and interactions.

3.1 System Component

System components are the essential parts that make up a larger system, such as a computer, network, or software application. These components work together to ensure the system functions correctly and efficiently, typically including hardware, software, data, people, and processes.

These elements interact to collect, process, store, and distribute information, thereby supporting decision-making and control within an organization.

Figure 1 presents a simple explanation of the system's behaviour. the user interacts with the web page (a front-end component) and the front-end system interacts with the back-end system, providing the information for the system to function correctly.

Fig 1: Explanation of system behaviour.

3.2 Interface Prototyping

In this study, the interface prototype focuses on a customer interface designed to facilitate the online purchase of e-tickets. Figure 2 outlines the steps involved in each customer action throughout the purchase process.

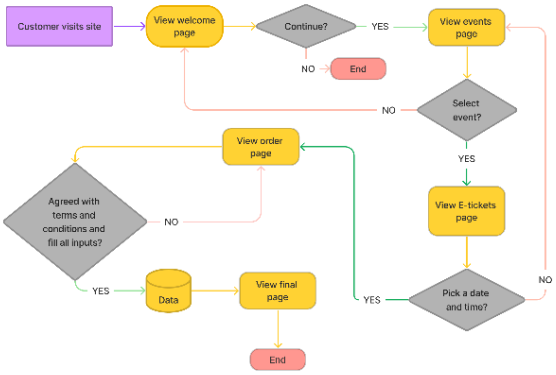


Fig 2: Flow customer interactions with interface.

Description: The customer visits the website or webpage of the E-Ticketing platform and sees the welcome page. If the user wants to continue, then will see the event page, which presents the events taking place.

The user can stop and go to the welcome page, but if the user selects an event, the user can process the webpage to choose a date and time for that event.

When a date and time is picked, the user can proceed to the order page, agree with the conditions, submit the Ethereum account and fill in the form to pay the e-ticket.

If all is well, the system sends the data to the server and records the information, the user just sees a thank you page and receives an email with the electronic ticket. This aspect will be next illustrated in Figure 3.

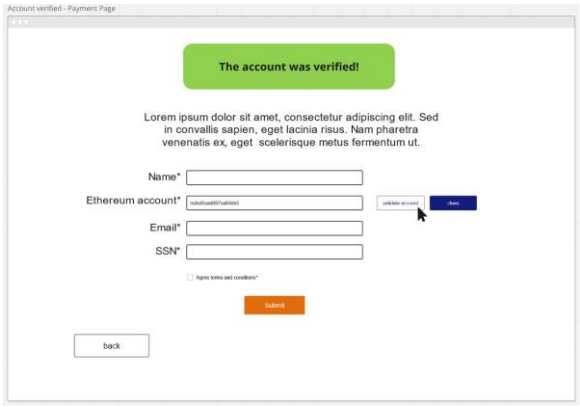


Fig 3: Sample of buy ticket interface.

4. DEVELOPMENT AND IMPLEMENTATION

After the modelling phase, the project moves on to the development and implementation phase. Therefore, this section first details the development of the proposed web application for this project. However, before diving into coding, it is necessary to carry out some additional planning. Afterwards, it starts the implementation and coding rises.

4.1 Digital architecture of a smart contract

Digital architecture refers to the design, structure, and organization of digital systems, including software, hardware, networks, and data. It encompasses the principles, techniques, and methodologies used to create and manage digital environments, such as computer systems, websites, applications, and other technology-driven platforms

In this scenario, the digital architecture of a SC (smart contract) process can be summarized in a process of two key stages, as illustrated in Figure 4:

- **Development**: This stage involves the creation, updating, or interruption of the Smart contract in case of process failure. In this case, the development starts with coding the Smart contract and when finished the developer deploys in the network.
- **Process automation**: This refers to the technology used to execute recurring tasks or processes within a business or organization without requiring human intervention. After the deploy the system will requires time to populate the Smart contract in the network, when finished that process the information in about the Smart contract will be recorded into a block. After that the block will be replicated to the network in many machines logged and terminated the entire process.

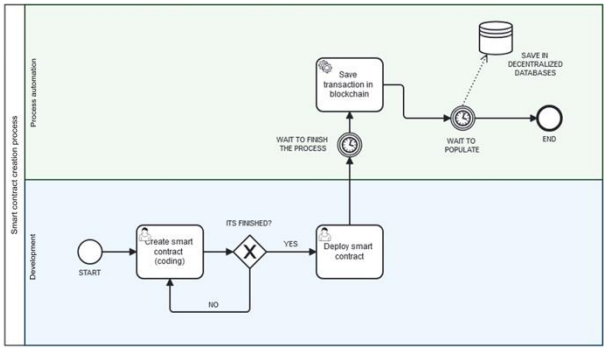


Fig 4: Digital architecture of SC creation process.

4.2 Tools used to development

For front-end development, several tools were utilized, such as Visual Studio Code served as the primary editor for writing, editing code and identify and rectify errors.

Programming languages and frameworks such as JavaScript with TypeScript and Vue facilitated the development process.

JavaScript is utilized within the webpages, which are developed using TypeScript. The framework Vue.js is employed to establish an HTTP localhost server, facilitating real-time site previews and updates.

For back-end development, various tools were employed, such as MySQL Server functioned as the database manager, while MySQL Workbench provided the interface.

The front-end and back-end developments are integrated within the same project but require independent initiation. The back end is responsible for initializing the database, which underpins the data displayed on the website. This database stores Ethereum public accounts submitted by customers upon form completion and validation. Each record in the database associates an order ID with an Ethereum ID.

4.3 Implementation

First, the development of HTML involves creating webpages, CSS, and connections to Vue, along with the integration of add-ons and images. Once this phase is completed, testing operations must commence.

Next is the database development phase, which includes creating the database using Workbench with SQL Server, followed by testing operations. The project required a database with test data to be evaluated step by step across the web pages.

The next phase involves integrating the HTML with the database to enable CRUD operations, after which exhaustive testing must be carried out. The backend folder and its code files are designed to interface with the database software and support frontend development.

To achieve this, Node.js [15] must be installed in the backend folder. Node.js provides a server-side JavaScript runtime environment, facilitating the connection between the backend and frontend by managing their dependencies and interactions.

In the ultimate phase, the development of smart contracts occurs, focusing on creating the contracts and managing transitions between the client side and operations side.

Ganache is a personal blockchain used for rapid development of Ethereum and File coin distributed applications [16]. It functions like an EVM and provides Ethereum public and private key accounts, along with gas and simulated Ethereum balances. Ganache allows testing the transactions and smart contracts in a local environment.

Unfortunately, testing must be conducted within a local ecosystem or off-chain blockchain storage [17], as performing these tests on a real Ethereum network or on-chain blockchain storage [17] would be prohibitively expensive due to the high cost of Ethereum transactions.

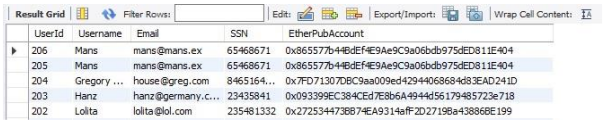
5. RESULTS

In this section the main project results after implementation and testing are presented. Subsection 5.1 provides an overview of the records stored in the local database during frontend testing. Subsection 5.2 details the generation of blocks and blockchain transactions observed during frontend testing. Following that, subsection 5.3 covers the testing of the smart contract after deployment, while subsection 5.4 describes the process of deploying and creating the smart contract.

For simplicity and demonstration purposes, there is no login system, the order and user will share the same number ID. However, it's important to note that these IDs are distinct in the database: one represents the user ID, and the other represents the order ID.

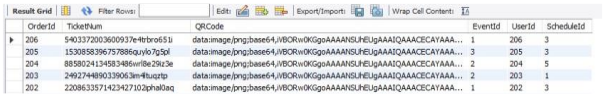
5.1 Local database

In the local database, the user and orders are created, like demonstrated in Figure 5 and Figure 6.



UserId	Username	Email	SSN	EtherPubAccount
206	Mans	mans@mans.ex	65468671	0x865577b44BdEf4E9Ae9C9a06bdb975dED811E404
205	Mans	mans@mans.ex	65468671	0x865577b44BdEf4E9Ae9C9a06bdb975dED811E404
204	Gregory ...	house@greg.com	8465164...	0x7FD71307DBC9aa009ed42944068684d83EAD241D
203	Hanz	hanz@germany.c...	23435841	0x093399EC384CEd7E8b6A4944d56179485723e718
202	Lolita	lolita@lol.com	235481332	0x272534473B874EA9314affF2D2719Ba43886BE199

Fig 5: Illustration of user's data from workbench.



OrderId	TicketNum	QRCode	EventId	UserId	ScheduleId
206	5403372003600937e4tbro651i	...	1	206	3
205	153085839675786guylo7g5pl	...	3	205	3
204	8858024134583486wrl8e29iz3e	...	2	204	5
203	2492744890339063m4tuqztp	...	2	203	1
202	2208633571423427102phal0aq	...	1	202	3

Fig 6: Illustration of orders data from workbench.

5.2 Block and blockchain

The generation of the smart contract is potentially more complex. For example, consider tracking block 36, which is related to user ID 206, for a better understanding. First, a transaction for the e-ticket is created within the block (Figures 7 and 8).

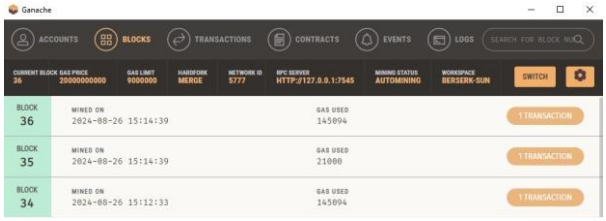


Fig 7: Illustration of a mined block in Ganache.

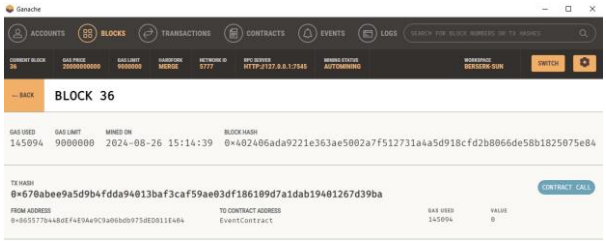


Fig 8: Scope and detail of block 36.

By clicking on the block, more detailed information about it is showed. This includes data such as gas usage, block hash, and, if a smart contract is involved and already deployed, details about the contract.

When the block is processed locally, this process is very fast. If the contract hasn't been deployed yet, Ganache will only display basic information, such as gas usage, date, and block hash.

5.3 Testing smart contract

To do tests in the smart contracts, the Truffle framework **Error! Reference source not found.** was used in the terminal of the Visual Studio Code. The Truffle test command is used to run automated tests for smart contracts written using Truffle. In the present case, as shown in Figure 9, a unit test was created to validate the core functionality of the buy function within the SC. This function aggregates the necessary elements in the array required to interact with the contract and confirms its intended behaviour.

The Deployment Test is successfully deployed, as indicated by the green checkmark beside "should deploy correctly.", the same for Buy Function Test, was performed correctly during the test, as seen in the green checkmark next to "should perform the buy function correctly" with a runtime of 60 milliseconds, because is executed locally.

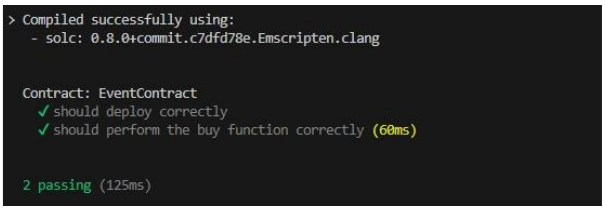


Fig 9: Unit test for validation of the core functionality.

These passing results in Ganache, as in Figure 10, confirm that the contract's key functions, including `buy`, behave as expected under the test conditions. The quick execution times (125ms overall, because is executed locally) indicate efficient interactions with the blockchain simulator, supporting the integrity and responsiveness of the contract functions.

Fig 10: Results in Ganache.

The test suite for *EventContract* (see Figure 11), shows that the contract deploys correctly, but the buy function test fails due to an invalid address error, so the deployment was not completed. Specifically, the error message tells us that the _seller argument was passed an empty string ("") instead of a valid Ethereum address. In Ethereum, addresses must be in a specific format and an empty string doesn't meet these requirements.

The error originates in the test file Test.js, on line 20. This line is where the buy function is called, and it appears that _seller is either not initialized or is given an invalid value. The underlying problem here is that the test setup doesn't provide a valid address for seller.

Fig 11: Illustration of a failure result.

Based in test results in Figure 12, the *EventContract* successfully deploys, but the test for the buy function fails. In successful deployment, the contract deploys without issues, as indicated by the checkmark next to "should deploy correctly", but is failure in buy Function Test.

The error suggests that the contract has a restriction on who can execute the buy function, likely enforced with a require statement such as require. This condition checks that only a specific address (the buyer) is allowed to call the buy function.

Cause of the Failure: The test appears to be attempting to call buy from an address that isn't recognized as the "buyer" according to the contract's logic. As a result, the transaction is reverted with the error message "Only buyer can buy."

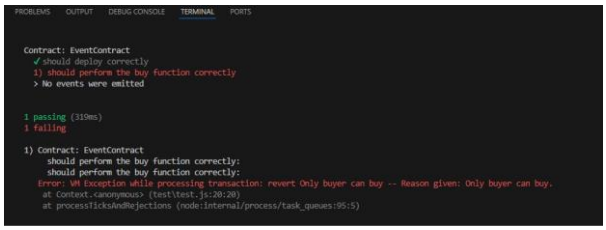


Fig 12: Illustration of buyer failure.

5.4 Deploy smart contract

After testing transactions and mining blocks, the Smart Contract (SC) needs to be deployed. In this project, deployment is performed manually via the Visual Studio Code terminal with Truffle library. Once deployed, Ganache will display the following information presented in Figure 13.

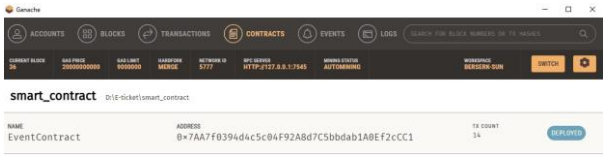


Fig 13: Illustration of the SC once deployed in Ganache.

Ganache displays the smart contract's address, which need to copy and paste into the project to interact with the mined blocks and the deployed contract. If the contract hasn't been deployed, it will only show "Not Deployed."

By clicking on the contract, as shown in Figure 14, Ganache provides detailed information such as storage, transactions, and events.

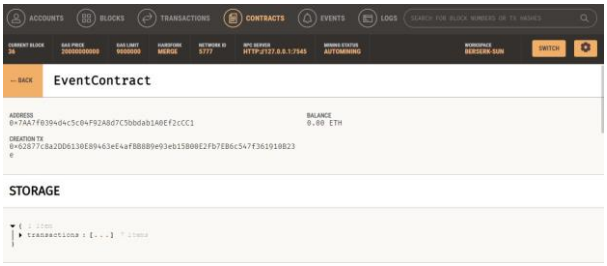


Fig 14: Details of the SC in Ganache.

Ganache has three important keys that allow users to view information and analyse what is happening during deployment and afterward, they are: Storage, transactions and events.

Each smart contract has its own storage, demonstrated in Figure 15, space where it can save variables and state information:

Fig 15: Print screen inside storage.

Inside storage, it shows the information codified in smart contract project, in first look, the information will be generic or empty, case the contract was not interactions. After that, the ganache records the information in the storage, demonstrated in Figure 16.

Fig 16: Illustration of transactions and events in the SC.

If everything is correct, the Ganache will record the transaction in the mined block. The subscription will then be realized, visible in the Events tab within Ganache, and sent to the blockchain, as shown in Figure 17.

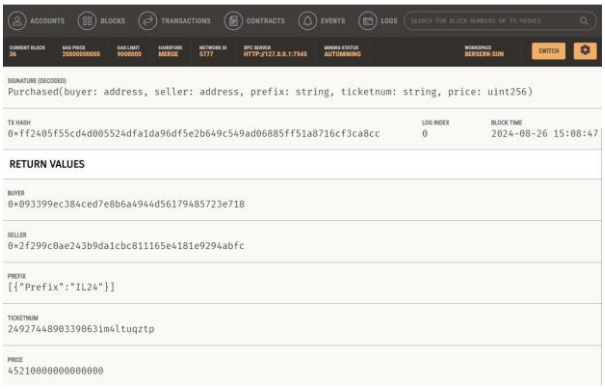


Fig 17: Event details about the smart contract deployed.

5.5 Security Analysis

Due to the project's limited scope and its local development context, the implementation of a comprehensive security system was deemed unnecessary. However, a thorough examination of potential security measures was conducted and documented.

Blockchain-based smart contract offers several security advantages for e-ticketing platforms, like the decentralization, in the traditional ticketing systems rely on centralized databases, which can be a single point of failure. Blockchain's decentralized nature distributes data across multiple nodes, reducing the risk of data breaches [19].

The immutability of the smart contracts blocks fraud. Once a ticket is issued on the blockchain, it cannot be altered or duplicated. This prevents counterfeit tickets and ensures the authenticity of each ticket.

Blockchain allows for transparent tracking of ticket ownership and transactions. This makes it easier to verify the legitimacy of tickets and reduces fraud.

Smart contracts are self-executing contracts with terms directly written into code. They automate the ticketing process, ensuring that tickets are only transferred under predefined conditions, such as payment confirmation.

But smart contracts have disadvantages too, like complexity, the multi-phase system can be complicated for users, especially those who are not tech-savvy. This complexity might deter some users from adopting the wallet.

The risk of loss the 12-phase code (or more like Trezor [20]) is potential and there is no recovery option, they permanently lose access to their funds. This can be a significant risk compared to

wallets that offer recovery options. The need to input two random phases for every transaction can be cumbersome and time-consuming, potentially leading to a less smooth user experience. Advanced security features often come at a higher cost, whether it's a hardware wallet or a premium software service.

6. CONCLUSION

The development of smart contracts on blockchain technology marks a significant shift in digital transactions and agreements, enhancing transparency, security, and decentralization. This technology effectively addresses issues such as fraud and inefficiencies in ticketing by eliminating intermediaries. The ambitious project described creates a blockchain-powered e-ticketing platform using smart contracts to ensure secure, transparent, and tamper-proof ticketing transactions.

By leveraging the decentralized nature of blockchain, this platform aims to eliminate fraud, streamline ticket verification, and enhance user experience for both organizers and attendees. The smart contract architecture automates critical processes from ticket issuance to resale, making the e-ticketing process more efficient and trustworthy.

The project phases encompass a comprehensive review of smart contracts, blockchain, and e-ticketing technologies, followed by system design, and development and implementation of the platform using technologies like JavaScript, Solidity, Truffle, and Ganache.

In conclusion, the project highlights the transformative potential of blockchain and smart contracts in the event ticketing industry, while also recognizing the need for further enhancements and real-world testing to ensure scalability and adoption.

7. REFERENCES

[1] Singhal, Bikramaditya, Et Al. "How Blockchain Works." Beginning Blockchain: A Beginner's Guide To Building Blockchain Solutions (2018): 31-148.

[2] Di Pierro, Massimo. "What Is The Blockchain?" Computing In Science & Engineering 19.5 (2017): 92-95.

[3] JavaScript Docs: https://developer.mozilla.org/en-US/docs/Web/JavaScript, accessed in November 2024.

[4] TypeScript homepage: https://www.typescriptlang.org/, accessed November 2024.

[5] VueJS homepage: https://vuejs.org/, accessed in November 2024.

[6] MySQL Workbench homepage: https://www.mysql.com/products/workbench/, Acceded in November 2024.

[7] Ganache homepage: https://archive.trufflesuite.com/ganache/, accessed in November 2024.

[8] Solidity homepage: https://soliditylang.org/, accessed in November 2024.

[9] Visual Studio Code homepage: https://code.visualstudio.com/, accessed November 2024.

[10] Bmj, "Prisma 2020 Explanation And Elaboration: Updated Guidance And Exemplars For Reporting Systematic Reviews Bmj," 2021. Available: Https://Www.Bmj.Com/Content/372/Bmj.N160, Accessed in november 2023.

[11] Manar Abdelhamid and Ghada Hassan. 2019. Blockchain and Smart Contracts. In Proceedings of the 8th International Conference on Software and Information Engineering (ICSIE '19). Association for Computing Machinery, New York, NY, USA, 91–95. https://doi.org/10.1145/3328833.3328857.

[12] A. Abuhashim and C. C. Tan, "Smart Contract Designs on Blockchain Applications," 2020 IEEE Symposium on Computers and Communications (ISCC), Rennes, France, 2020, pp. 1-4, doi: 10.1109/ISCC50000.2020.9219622.

[13] P. Sombat and P. Ratanaworachan, "A Blockchain-Based Ticket Sales Platform," 2023 27th International Computer Science and Engineering Conference (ICSEC), Samui Island, Thailand, 2023, pp. 226-230, doi: 10.1109/ICSEC59635.2023.10329682.

[14] Noerlina, A. Khairunnisa and Meiryani, "Assessment of E-Ticketing Technology in Concert Website: A Review of Benefits, Profits, and Customer Satisfaction," 2023 International Conference on Information Management and Technology (ICIMTech), Malang, Indonesia, 2023, pp. 1-5, doi: 10.1109/ICIMTech59029.2023.10277708.

[15] NodeJs homepage: https://nodejs.org/en, accessed in November 2024.

[16] Ganache Docs: Https://Archive.Trufflesuite.Com/Docs/Ganache/, accessed in May 2024.

[17] Reijers, Wessel. Et Al. "Now the code runs itself: On-chain and off-chain governance of blockchain technologies". Topoi 40 (2021): 821-831, accessed in June 2024.

[18] Truffle Documentation: https://Archive.Trufflesuite.Com/Docs/Truffle/, accessed in August 2024.

[19] Open Access Publisher Empowering Researchers, Https://Journals.Plos.Org/Plosone/Article?Id=10.1371/Journal.Pone.0284166, Accessed in August 2024.

[20] Crypto Security Trezor: https://Trezor.Io/, accessed in August 2024.