BLOCKCHAIN-BASED SMART CONTRACTS E-TICKETING PLATFORM

Cláudia Marisa Canhoto da Silva

Orientadores

Alexandre José Pereira Duro da Fonte

Mónica Isabel Teixeira da Costa

Dissertação apresentada à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática - Área de Especialização em Desenvolvimento de Software e Sistemas Interativos, realizada sob a orientação científica do Professor Doutor Alexandre José Pereira Duro da Fonte e coorientação do Professor Doutor Mónica Isabel Teixeira da Costa, do Instituto Politécnico de Castelo Branco.

Janeiro 2025

Composição do júri

Presidente do júri

Doutor, José Carlos Meireles Monteiro Metrôlho

Vogais

Doutor, Alexandre José Pereira Duro da Fonte

Professor Adjunto, Instituto Politécnico de Castelo Branco

Doutor, Bruno Miguel Correia da Silva

Professor Associado, Universidade da Beira Interior

Doutor, Fernando Sérgio Rodrigues de Brito da Mota Barbosa

Professor Adjunto, Instituto Politécnico de Castelo Branco

Resumo

Os *Smart contracts* ou contratos inteligentes, são uma inovação fundamental da cadeia de blocos, comportam-se como acordos digitais que se executam, de forma autónoma, termos predefinidos. Estes contratos são seguros, descentralizados, transparentes e imutáveis, estando estes preparados para uma vasta aplicação e em diversos sectores, como a venda de bilhetes para eventos, agilizando transações e reduzir a fraude.

Nesta dissertação propõe-se um projeto com o objetivo de explorar a utilização de contratos inteligentes na comercialização de bilhetes eletrónicos para eventos e com isto reduzir a especulação na venda de bilhetes e as redundâncias de papel, bem como a aplicação prática desta solução. Esta solução obriga assim à conceção, execução e depuração de contratos inteligentes utilizando *Solidity* na cadeia de blocos (*blockchain*) *Ethereum*.

Para responder a estes desafios, foi especificamente desenvolvida uma aplicação funcional, tendo como alvo vendedores de bilhetes, compradores e organizadores de eventos. Os resultados deste desenvolvimento demonstraram a viabilidade e aplicabilidade dos *Smart contracts* para processos de emissão de bilhetes transparentes e eficientes, embora atualmente limitados a um ambiente local. Contudo, estes fornecem uma base sólida para melhorias futuras e uma adoção mais ampla.

Em resumo, este projeto destacou o potencial e a viabilidade das cadeias de blocos e dos *Smart contracts* para transformar a emissão de bilhetes para eventos, garantindo segurança, escalabilidade e confiança do utilizador.

Palavras-chave

E-ticket, Blockchain, Smart contract, Ethereum, Cripto moeda.

Abstract

Smart contracts, a fundamental 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 dissertation proposes a project aimed at exploring the use of smart contracts in the sale and marketing of event tickets. The main objective is to reduce ticket speculation and paper redundancies, covering the design, execution, and debugging of smart contracts using Solidity on the Ethereum blockchain.

To respond to these challenges, a functional application was specifically developed, targeting ticket sellers, buyers and event organizers. The results of this development demonstrated the feasibility and applicability of Smart contracts for transparent and efficient ticketing processes, although currently limited to a local environment. However, they provide a solid foundation for future improvements and wider adoption.

In summary, this project highlighted the potential and viability of blockchains and Smart contracts to transform event ticketing, ensuring security, scalability and user trust.

Keywords

E-ticket, Blockchain, Smart contract, Ethereum, Cryptocurrency.

General index

Index of figures

List of tables

List of abbreviations, acronyms, and acronyms

ABI – Application Binary Interface.

API – Application programming interface.

B2B – Business-to-business.

CLI – Command Line Interface.

CSS – Cascading Style Sheets.

CRUD – Create, Read, Update, Delete.

DAO – Decentralized autonomous organizations.

DNS – Domain Name System.

ENS – Ethereum Name Service.

ER – Entity-Relationship.

EVM – Ethereum Virtual Machine.

HTML – Hypertext Markup Language.

JS – JavaScript.

NFTs – non-fungible tokens.

NPM – Node Package Manager.

SCs – Smart contracts.

SC – Smart contract.

UML – Unified Modelling Language.

WEB – WWW or simply the Web.

1. Introduction

The development and application of cryptocurrencies have highlighted numerous challenges, such as security concerns, lack of trust in peer-to-peer transactions, currency duplication, transaction valuations, and the storage of transactional data.

These major challenges, which were previously managed in centralized systems, have led to the emergence of blockchain — a distributed ledger that enables decentralized storage of transactional information via nodes connected to a peer-to-peer network [1].

Records (or blocks) containing this information are securely linked in a chain using cryptographic hashes or hash functions (e.g., SHA-256 or SHA-3). As blockchain technology evolved [2], additional important applications like SCs [3] also emerged [4].

These Smart contracts are digital agreements that automatically execute specific terms when certain conditions are met, eliminating the need for intermediaries. Additionally, Smart contracts are recorded on the blockchain, ensuring security and transparency.

The key features of Smart contracts include decentralization, which enables them to operate on decentralized platforms like blockchain networks, and transparency, as all parties can view the terms and conditions simultaneously. Furthermore, Smart contracts are immutable, meaning once they are deployed, they cannot be altered, ensuring trust among all parties involved.

These characteristics make Smart Contracts (SCs) highly adaptable for various sectors, such as marketing or the sale and exchange of event tickets. Smart contracts have the potential to streamline and automate bureaucratic processes, reducing the number of intermediaries involved and ensuring the security of ticket transactions.

1.1. Objectives

The main objective of this project is to study the use case of Smart contracts built on blockchain technology for events, focusing on ticket marketing and sales to minimize speculation in ticket sales and reduce redundant paper/systems.

Given the implementation of Smart contracts is complex, along with the methodology for developing the software components (e.g., ICONIX) of the application, it is crucial to study methodologies or frameworks that guide the design, deployment, and debugging of Smart contracts.

The specific objectives of the dissertation include:

1. Defining the problem concisely to help identify the desired outcomes and strengthen the project. In this case, the issues to be addressed include "How to sell an E-Ticket?" by creating a platform-based solution, including a webpage and a database system connection, and "How to create a smart contract?" using the Solidity language and proper libraries [5].

2. Analysing the key stakeholders involved in the process and determining who will benefit from this approach (e.g., artists, organizers, ticket sellers or intermediaries, buyers). The project will target enterprises and individuals with crypto wallets and Ethers who wish to sell e-tickets.

3. Establishing and reviewing the essential requirements, such as complexity, scalability, security, and ease of use.

4. Creating a system model based on the identified requirements and needs, assisted by the development of a functional prototype.

5. Iteratively reviewing the model, evaluating the implemented prototype, and gathering feedback from key stakeholders involved in the process.

1.2. Dissertation Planning

To achieve the proposed objectives, 5 phases are foreseen during the development of the work (see Table 1):

1) State of the art or research for models/algorithms to design, execution and debugging a Smart contract. It's focuses on comparing the project's objectives with existing market solutions, analysing their characteristics and functionalities. This assessment is crucial for identifying opportunities for actual product differentiation and innovation.
2) Evaluation of results and comparison with results of approaches, included in the state of the art or PRISMA method.

3) Analysis of requirements for implementation, involves selecting the most suitable tools for both the development and implementation stages. This step ensures that the project is built using the best resources available.

4) Modelling, selection, and implementation of algorithms, where an in-depth study of various functionalities and interactions is conducted, with a particular focus on defining the different types of customers who will benefit from the system. This step is essential for tailoring the system to meet users' needs effectively.

5) After studies the development phase initialize and the entire system is brought to life, including the creation of the application itself. This is where all the design and planning take shape into a functional product.

6) Writing of the report.

7) Finally, the "Testing and reporting" phase ensures the system is reliable and free from errors. Repeated verifications and validations help identify and resolve any potential issues. The reporting process documents all significant findings and observations, providing a comprehensive overview of the system's performance and areas for improvement.

Table 1 – Project schedule

	November 2023	December 2023	January 2024	February 2024	March 2024	April 2024	May 2024	June 2024	July 2024	August 2024	September 2024	October 2024
1												
2												
3												
4												
5												
6												
7												

1.3. Dissertation Outline

Chapter 2 thoroughly examines the research topic's background, highlighting its importance and potential contributions. It provides essential context, explores relevant literature, and outlines critical issues, illustrating the study's significance and the innovative solutions it may offer. This discussion engages the reader and underscores the timeliness and necessity of the research in advancing knowledge in the field.

Chapter 3 will analyse scientific articles on the future of smart contracts to identify key insights and trends relevant to the E-Ticketing project. By summarizing relevant papers, this chapter aims to generate innovative ideas and practical solutions, exploring the applications, challenges, and benefits of smart contracts. This analysis will strengthen the project's conceptual framework and keep it aligned with emerging technological advancements.

Chapter 4 will examine the engineering and computer science principles essential for designing a robust system. It will define the architecture, components, modules, interfaces, and data structures needed to meet requirements effectively.

A systematic approach will ensure that each element works cohesively within the system. The chapter will also analyse the database structure for efficient information management, focusing on data storage, retrieval mechanisms, and integrity. This exploration will establish a framework that addresses immediate project needs while anticipating future enhancements and scalability.

Chapter 5 focuses on Development and Implementation, detailing the foundational aspects of the proposed web application. It will outline the initial development stages, emphasizing strategic planning before coding begins.

A clear implementation roadmap will guide the project and ensure key milestones are met. This chapter will stress the importance of defining project requirements and architecture to enhance the application’s quality and performance. It will explore design patterns and methodologies to streamline development and promote team collaboration.

This careful planning will facilitate a smoother coding process and create a robust, user-friendly web application that meets project goals. Ultimately, Chapter 5 will highlight the critical relationship between design and implementation for a functional and scalable final product.

Chapter 6 will present the study's results, showcasing the outcomes of the development and implementation processes. It will highlight final solutions and provide a comprehensive analysis of the methodologies used. By examining collected data and performance metrics, the chapter will demonstrate how the proposed solutions address earlier challenges

A thorough discussion will compare results against benchmarks, illustrating the practical implications of the findings. Additionally, it will reflect on the iterative processes during development, emphasizing the role of feedback loops and continuous testing in refining the solution. Ultimately, Chapter 6 aims to highlight the significance of the results in advancing knowledge in the field and inform future research and applications.

2. Background

This chapter highlights the importance of the research topic and its potential contributions to the field, setting the stage for the study and helping the reader understand its significance.

2.1. Webs and Progress

Before starting the development of the project, it is necessary to perform a retrospective of the past of the Webs to understand how it has evolved up to the present day and thus have a better perception of what the future holds for us.

2.1.1. Web 1.0

The era of Web 1.0, spanning from 1990 to 2004, marked the inception of the World Wide Web. During this period, web pages were static and presented in a "Read-only" format [6].

These pages interconnected through hyperlinks, creating a network of information that users could navigate. However, there was no interaction between the user and the content; users were passive consumers of information. The primary focus was on delivering content, and the user experience limited to viewing and reading.

2.1.2. Web 2.0

The advent of Web 2.0, beginning in 2004 and continuing to the present day, revolutionized the way we interact with the internet. This generation of the web introduced "Read/Write" capabilities [6], allowing users to not only consume content but also create and share it. Web 2.0 is characterized by dynamic and interactive pages designed to operate seamlessly across various devices, from desktops to mobile phones.

In this phase, user interaction became a central feature. Users could engage with content, participate in discussions, and contribute to the web ecosystem. The rise of social media platforms, blogs, and wikis exemplifies this shift. Additionally, user data became an asset. Companies like Google and Amazon began collecting and storing personal information to enhance online sales and tailor user experiences. This practice effectively transformed users into "products," as their data was monetized to drive targeted advertising and personalized services.

A significant development in Web 2.0 is the integration of single sign-on (SSO) systems, allowing users to log into multiple platforms and services with a single account. This convenience further interconnected the web, making it easier for users to navigate and interact with various online services.

2.1.3. Web 3.0

The Semantic Web, a concept under study since 2014, promises to bring profound changes to our relationship with and understanding of the internet. This new stage of the internet aims to transform the way we interact with digital content and data.

With Web 3.0, payments will be made differently; they will have to be processed with virtual currency. In the case of Ethereum, this currency is called Ether, however, there are many other virtual currencies available in the market, such as Bitcoin [7] or Cardano [8].

Currently, economic, and financial transactions are carried out through intermediaries, such as banks; in the future, there is a possibility that this third party will be excluded, giving the possibility of carrying out transactions directly via tokens such as ETH (Ethereum) to send money directly from a browser, for example [9].

In the era of the Semantic Web, users will transition from being mere products to becoming true "owners" of their digital assets. This shift is made possible by Non-Fungible Tokens (NFTs), which empower users to own and control specific pieces of digital information or assets. These NFTs provide a heightened level of security and personal control, as the assets they represent cannot be taken or altered without the user's explicit consent. This marks a significant change, giving individuals far more autonomy over their digital presence.

The advancements in user authentication are paving the way for the Semantic Web to allow users to access various platforms with a single account. Unlike traditional data collection methods, this new approach leverages digital identities linked to Ethereum addresses and ENS (Ethereum Name Service) profiles [10].

Like DNS (Domain Name System) but with enhanced privacy features, ENS enables users to manage their domains anonymously. This system improves security by minimizing errors when inputting recipient addresses, ultimately enhancing the overall user experience and ensuring safer transactions [11].

With the decentralization of databases, the traditional model of centralized data storage gives way to decentralized or distributed systems. This shift means that major companies, which have long held control over our data, will lose that dominance. As a result, a new landscape for information exchange had to be designed, leading to the emergence of blocks and chains [12].

Though blocks and chains serve different purposes, they are closely intertwined, giving rise to the concept of blockchain, which refers to a sequence of transactions linked together. Each block in this system contains specific information that, once validated, becomes immutable, meaning it cannot be altered. After validation, the block is added to a chain of other blocks, forming a secure, unchangeable sequence of data. This chain is then copied and distributed across multiple machines in various locations, ensuring both its security and reliability [13].

Blockchain technology was first conceptualized by Satoshi Nakamoto in 2008 as the underlying technology for Bitcoin, a decentralized digital currency [14], is a distributed ledger that records transactions across multiple computers, ensuring that the record cannot be altered retroactively without altering all subsequent blocks and the consensus of the network.

Smart contracts, proposed by Nick Szabo in 1994, are self-executing contracts with the terms of the agreement directly written into code [15]. These contracts automatically execute and enforce the terms when predefined conditions are met.

Research on blockchain has expanded beyond cryptocurrencies to include applications in supply chain management, healthcare, and finance. Studies highlight its potential for enhancing transparency, security, and efficiency. For instance, a study by Zheng et al. (2017) provides a comprehensive overview of blockchain technology, its architecture, and its applications [16].

Similarly, Smart contracts have been explored for their ability to automate transactions and agreements in various sectors such as decentralized finance (DeFi), real estate, and insurance. They are praised for reducing the need for intermediaries and increasing trust and transparency. A notable paper by Atzei et al. (2017) discusses the security vulnerabilities in Smart contracts and proposes solutions to mitigate these risks [17].

Despite the promising applications, challenges related to scalability, security, and regulatory acceptance of blockchain, and Smart contracts remain. For example, scalability issues are discussed in the work of Croman et al. (2016), which examines the limitations of blockchain scalability and potential solutions [18]. Your research could address these gaps by exploring innovative solutions or new applications.

The concept of Decentralized Autonomous Organizations (DAOs) represents a revolutionary approach to organizational management. Unlike traditional hierarchical structures, where the founder sits at the top and employees are at the bottom, DAOs operate on a decentralized model. This structure eliminates the pyramid, allowing for more democratic and transparent decision-making processes [19].

Decentralized Autonomous Organizations (DAOs) are characterized by several key features. Firstly, decentralization ensures there is no central authority; instead, power is distributed among all members who hold governance tokens, allowing them to vote on proposals and decisions democratically [19].

The backbone of a DAO is its Smart contract, which defines the organization's rules and holds its treasury. Once deployed on the blockchain, these rules cannot be changed without a member [19].

Transparency is another crucial aspect, as all activities and decisions within a DAO are recorded on the blockchain, making every transaction and vote publicly accessible. Finally, DAOs operate with autonomy based on the rules encoded in their Smart

contracts, reducing the need for intermediaries and minimizing the risk of human error or manipulation [20].

2.2. Chapter Conclusion

In summary, if developed as previously outlined, Web 3.0 has the potential to fundamentally transform the economic and financial landscape, as well as the autonomy afforded to consumers. The decentralized nature of Web 3.0 technologies promises to facilitate faster transactions, enhance data security, and reduce the vulnerability of information to cyber-attacks. This shift toward a more decentralized internet could empower users by giving them greater control over their data and financial interactions.

However, it is essential to acknowledge that this envisioned scenario is not without its challenges. For instance, there may be significant constraints on the distribution of information, as decentralized systems can introduce complexities in regulatory compliance and data governance. Moreover, the scalability of processes and transactions could become a critical issue; if systems are not designed to handle increased loads effectively, they may become overwhelmed, leading to delays and inefficiencies.

Additionally, while Web 3.0 aims to mitigate certain cyber threats, it may inadvertently create opportunities for more sophisticated forms of cybercrime. As systems evolve, malicious actors may adopt more aggressive and targeted strategies, necessitating robust security measures to protect users and their data. Thus, while the prospects of Web 3.0 are promising, a thorough understanding of its limitations and potential risks is crucial for realizing its benefits while safeguarding against its inherent challenges.

3. Systematic Review or Literature Review

In this case, the aim of this chapter is to analyse in detail scientific articles that contain information related to Smart contracts and to understand their evolution. The analysis will focus on the review of relevant articles related to this topic. This chapter is organized as follows: first the review methodology, search questions, exclusion criteria, sources, and data extraction are presented; following this an analysis and a summary of articles' studies will be presented.

3.1. Review Methodology

This review was carried by employing the novel methodology PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) of reporting systematic reviews and meta-analyses [20], conducted the present investigation following the core stages (planning, conducting, and reporting) presented in Figure 1.

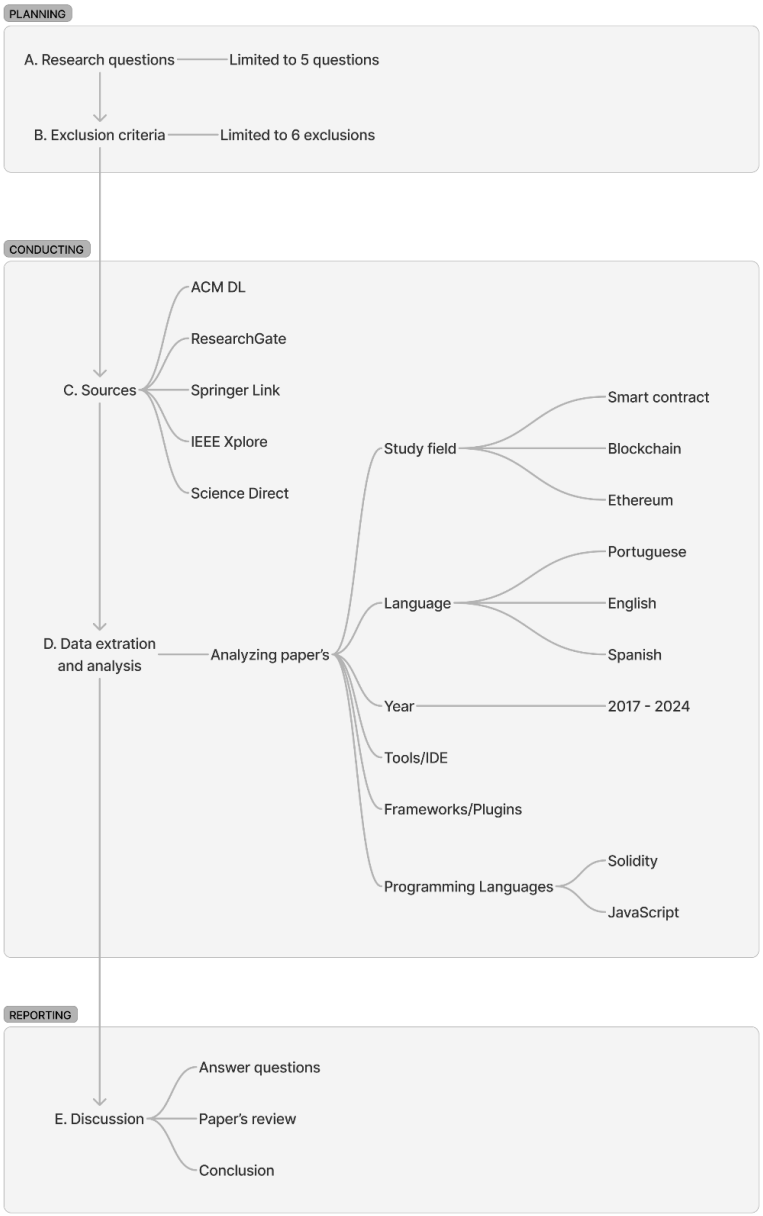

Figure 1 – Structural schema of literature review flow.

A. Search questions

This topic discusses the methods and criteria of the research carried out, in accordance with the proposed objectives.

1. **Research Question 1 (RQ1):** What programming languages might be available for development?

2. **Research Question 2 (RQ2):** Are there frameworks to support development?

3. **Research Question 3 (RQ3):** What tools should you use for development?

4. **Research Question 4 (RQ4):** Are there any developed blockchain based Smart contract solutions?

5. **Research Question 5 (RQ5):** Are there any studies on blockchain based Smart contracts and E- Ticketing?

B. Exclusion criteria

1. **Criteria 1:** The studies prior to 2017 are excluded, to know the most recent technologies in blockchain and Smart contracts fields.

2. **Criteria 2:** Articles not published on non-credible websites for more accuracy about the study's Smart contracts and E- Ticketing.

3. **Criteria 3:** Studies based on non-integrative reviews [1] to discover what tools and frameworks are available to development.

4. **Criteria 4:** Studies in languages other than English or Portuguese or Spanish for better understanding.

5. **Criteria 5:** Open and opinion polls, such as blogs, forums, etc to exclude non-scientific evidence.

6. **Criteria 6:** Search in books with editions prior to 2017 are excluded, to know the most recent technologies in blockchain and Smart contracts fields.

[1] Alternative for reviewing and combining studies with different methodologies while maintaining the methodological rigor of systematic reviews.

C. Sources

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. The search was carried out between January and February 2024, and the set of individual or combined search terms were:

- "Smart contracts".
- "Blockchain".
- "Blockchain – Smart contracts".
- "Frameworks Smart contract".
- "Ethereum Smart contracts".
- "Smart contract E-ticking".

Among a total of 125 articles, 9 articles were identified and screened for inclusion in the review, as shown in Table 2.

Table 2 – Flowchart of the research phases

Identification	**Selection**	**Eligibility**	**Included**
Studies identified in: **Science Direct = 40** **IEEE Xplore = 55** **ACM DL = 20** **Research Gate = 10** **N = 125**	Studies analysed (by title and abstract). N = 90	Studied assessed for inclusion (full analysis). N = 75	Studies/books included for review. N = 9
Studies removed before screening: **Duplicate or similar studies.** **N = 10** **Studies in a language another then EN or PT or ES.** **N = 10** **Studies older than 2017.** **N = 15**	Records removed: Directed at education. N = 10 Economic. N = 5	Exclusion after full-text analysis: Content not very revealing N = 23 Content without Smart contract specification. N = 43	

D. Data extraction and analysis

Data was extracted from all identified studies using a pre-defined format: Study; Year of Publication; Authors; Source and ISBN / ISSN, this study is present in the Table 3.

Table 3 - Scientific articles or books analysed.

Ref	Research	Year published	Authors	Source	ISBN, ISSN or DOI
[21]	A Smart Contract - Based Mobile Ticketing System with Multi-Signature and Blockchain	2019	Lin, Keng-Pei. Chang, Yi-Wei. Wei, Zhi-Hong. Shen, Chih-Ya. Chang, Ming-Yi.	IEEE Xplore	978-1-7281-3575-5
[22]	Blockchain and Smart contracts	2019	Abdelhmaid, Manar. Hassan, Ghada.	ACM DL	9781450361057
[23]	Smart contract designs on blockchain applications	2020	A, Abuhashim. C C Tan.	IEEE Xplore	978-1-7281-8086-1
[24]	Immutable Smart contracts on Blockchain Technology: Its Benefits and Barriers	2021	Kumar Kaushal, Rajesh. Kukreja, Vinay. Kumar, Naveen. Narayan Panda, Surya.	IEEE Xplore	10.1109/ICRITO51393.2021.9596538
[25]	SmartCon: A blockchain-based framework for Smart contracts and transactions management	2021	Muneeb, Muhammad. Raza, Zeeshan. Haq, Irfan UI. Shafiq, Omair.	IEEE Xplore	2169-3536
[26]	Blockchain 2.0: A Smart contract	2022	Saini, Kavit. Roy, Abhishek. Raj Chelliah, Pethuru. Patel, Tanisha.	IEEE Xplore	978-1-6654-3656-4
[27]	A Blockchain-Based Ticket Sales Platform	2023	Sombat, Patcharaporn. Ratanaworachan, Paruj.	IEEE Xplore	979-8-3503-4210-9
[28]	Applying Smart contract in Blockchain Technology to	2023	Nguyen, Huu-Quang. Huynh, Huynh Tuong.	IEEE Xplore	979-8-3503-2405-1

	Manage the Ticketing Issuance and Ticketing Traceability		Pham, Thi-Thiet.		
[29]	Smart contracts and blockchain: An analytical approach	2023	Sayal, Anu. Vasundhara, Chetlur. Gupta, Veethika. Gupta, Ashulekha. Maheshawri, Himani. Memoria, Minakshi.	IEEE Xplore	979-8-3503-0448-0

3.2. Summarized Paper's Studies

The characteristics of the included studies were summarized, by topic, presenting the analysis described below.

a) Blockchain and Smart contracts by Manar Abdelhamid and Ghada Hassan

Blockchain is a distributed ledger maintained by network nodes that records all transactions across a network. These transactions are executed on multiple nodes, ensuring both transparency and immutability. A key application of blockchain is smart contracts, which are self-executing digital agreements that automatically enforce terms once specific conditions are met.

Ethereum is one of the most widely used platforms for smart contracts, known for its support of programming languages and advanced features. Smart contracts provide several benefits, including autonomy, trust, data backup, and accuracy, which collectively enhance data protection. This review aims to identify topics related to blockchain-based smart contracts and address the current challenges they face, while also highlighting gaps for future research, particularly from a technical standpoint.

Because blockchain operates in a decentralized manner, it eliminates the need for intermediaries, boosting efficiency, security, and reducing costs. Applications of blockchain extend across various sectors, including government, finance, commerce, the Internet of Things (IoT), and education. Smart contracts are essentially code stored on the blockchain, executing agreed-upon conditions between parties. Their execution across all participants' databases ensures consistency and reliability.

However, several challenges arise in the development of smart contracts, particularly in areas such as coding, security, privacy, and performance. Despite these issues, smart contracts are finding practical applications in fields like IoT, music rights management, and e-commerce. They facilitate digital property access, music rights tracking, and enable secure transactions between untrusted parties.

Prominent blockchain platforms for smart contracts include Bitcoin, NXT, and Ethereum, with Ethereum standing out due to its Turing-complete programming language, which supports the development of more complex smart contracts.

b) Smart contract Designs on Blockchain Applications by Alkhansaa Abuhashim and Chiu C. Tan

This paper delves into the development of Smart contract designs aimed at enhancing the efficiency of indexing and querying ride-sharing data stored in blockchain ledgers. The primary focus is on addressing the challenges of retrieving complex customer data and transactions in a decentralized manner, without relying on traditional relational database management systems (DBMS).

The study uses a bicycle-sharing system in smart cities as its main use case, specifically leveraging the Chicago Divvy Bicycle Sharing dataset from 2013 to 2017. It discusses the implementation of a sophisticated data structure to represent customer transactional data, including ride details, personal information, and order specifics.

Two Smart contract designs are introduced: Sparse Scenarios and Catalog Scenario. The paper examines various methods for transacting the blockchain, comparing simple individual transactions with more complex cases that utilize Smart contracts. Sparse and catalog designs are explored for updating blockchain states, emphasizing the importance of maintaining a sequence of actions and ensuring efficient retrieval of related transactions.

A key part of the research involves evaluating the gas consumption of mined transactions for both the Catalogue and Sparse Smart contract designs. Gas consumption experiments, conducted on the Rinkeby network using the Chicago Divvy dataset, demonstrate the differences in gas usage between the two designs.

The paper also discusses the limitations of querying blockchain data and presents experimental results comparing the efficiency of querying Sparse and Catalog transactions. The performance gap in retrieving data between the two designs is highlighted, showcasing the significant differences in efficiency.

Lastly, the paper acknowledges the current state of blockchain querying, referencing popular blockchain explorers like *Etherscan* and other research tailored to specific applications. It emphasizes the novelty of its contribution, offering distinct Smart contract designs to more effectively index and query blockchain transactions.

c) Immutable Smart contracts on Blockchain Technology: Its Benefits and Barriers by Rajesh Kumar Kaushal, Vinay Kukreja, Naveen Kumar and Surya Narayan Panda

SHA256 is considered safe for use in blockchain systems due to its irreversible nature, making it a secure choice for hashing. The most widely adopted type of blockchain is the public version, though it is more frequently targeted by attacks due to its open and decentralized structure. In blockchain ecosystems, there are three main types of participants: buyers, sellers, and miners.

Unverified transactions are stored in the MemPool, a block of memory available on each node in the blockchain network, until they are processed. The mining process, known as Proof of Work (PoW), is fundamental to verifying transactions and securing the network.

Smart contracts offer significant advantages over traditional systems, particularly by reducing corruption and ambiguity through automated and transparent execution. Various tools are available for developing smart contracts, including the MetaMask Plugin, Solidity, JavaScript/Node.js, Ethereum Wallet, Blockchain, Truffle, Web3.js, and Remix IDE.

However, there are challenges in implementing smart contracts, such as high energy consumption, slow transaction speeds, complex architecture, and a lack of awareness and expertise in the field. Common barriers to their widespread adoption include legal issues, high costs, and a shortage of experienced professionals.

d) SmartCon: A Blockchain-Based Framework for Smart contracts and Transaction Management by Muhammad Muneed, Zeeshan Raza, Irfan Ul Haq and Omair Shafiq

The proposed project focuses on developing a new transaction and data storage system that enhances transparency and security, enabling businesses to engage with untrusted parties while effectively reducing fraud. This innovative system is designed to identify any corruption within the blockchain when changes occur, thereby ensuring integrity.

Smart contracts play a crucial role in automating business-to-business (B2B) activities. They can be triggered by various inputs, including IoT devices, data streams, or other programs that create specific events. The article distinguishes between two types of smart contract management systems: one based on blockchain technology and another that relies on fundamental rules to automate contracts between parties.

Specifically, the proposed framework introduces a blockchain-based smart contract management system that utilizes two separate blockchains: SBlockchain and TBlockchain. SBlockchain is dedicated to storing smart contracts, while TBlockchain serves as the repository for all data generated by these contracts.

The implementation of this management system is executed in JavaScript, facilitating efficient smart contract management and interaction within the proposed architecture.

e) Blockchain 2.0: A Smart contract by Kavita Saini, Abhishek Roy, Pethuru Raj Chelliah, Tanisha Patel

The study delves into the evolution and implementation of blockchain technology, examining its transformative impact on both technological landscapes and society. It categorizes smart contracts into two main types: simple and complex.

Simple smart contracts facilitate digital value exchanges, such as applications requiring timestamping for important documents like birth certificates and educational degrees. In contrast, encompass decentralized autonomous organizations, decentralized autonomous governments, and decentralized autonomous societies, including business enterprises.

The execution of a smart contract involves several key steps. First, the contract is written as code and stored on a blockchain. Once the conditions are met, the execution of the code is triggered, resulting in the release of assets to the necessary parties. Finally, regulators can examine the immutable transaction record, which provides a transparent account of all activities that have occurred, ensuring accountability and understanding of the process.

f) Smart contracts and Blockchain: An Analytical Approach by Anu Sayal, Chetlur Vasundhara, Veethika Gupta, Ashulekha Gupta, Himani Maheshawri and Minakshi Memoria

The main objective of this article is to explore the necessity of blockchain technology in smart contracts and to discuss future directions for this field. It highlights several key areas where blockchain and smart contracts are making significant impacts, including Decentralized Finance (DeFi), Supply Chain Management, Digital Identity, Real Estate, and Healthcare.

In the realm of digital identity, the article emphasizes the role of smart contracts in securely and decentralizing the acquisition of digital identities. This innovation not only helps mitigate the risk of identity theft but also empowers customers with greater control over their personal information.

When it comes to real estate, the article proposes the use of smart contracts to automate the processes involved in buying and selling properties. This automation can significantly reduce the time and costs associated with these transactions while eliminating the need for intermediaries.

In the healthcare sector, the authors suggest that smart contracts can enhance the management of health records, providing greater security and facilitating information sharing among healthcare providers. This advancement has the potential to improve the quality of care and reduce costs within the healthcare system.

g) A Blockchain-Based Ticket Sales Platform by Patcharaporn Sombat and Paruj Ratanaworachan

The challenges associated with centralized ticket sales are significant, often resulting in system crashes, bot-controlled purchasing, VIP privileges, and counterfeit tickets in secondary markets. These problems stem from a lack of transparency and the inherent single point of failure present in centralized systems.

To address these issues, the paper introduces a decentralized application (DApp) that leverages blockchain technology and smart contracts for ticket sales. By utilizing

Non-Fungible Tokens (NFTs) as tickets, this solution ensures transparency, security, and immutability in all transactions.

The platform is deployed on two blockchains: Ethereum and Avalanche. While Ethereum is renowned for its popularity in smart contract deployment, it also suffers from high costs and inefficiencies. In contrast, Avalanche offers a more cost-effective solution with faster transaction speeds.

Tickets are created using the ERC-721 standard, which guarantees that each ticket is unique and non-divisible. This feature renders tickets tamper-proof and easily auditable on the blockchain.

The platform boasts several key features: it allows developers to easily extend its capabilities, is open-source and accessible to organizations of any size, and enables ticket resale on a secondary marketplace, thereby maintaining transparency and security throughout the ticketing process. Additionally, smart contracts enforce important rules regarding ticket quotas, sale timings, and resale pricing.

The paper also provides a comparison of deployment costs and transaction fees between Ethereum and Avalanche, highlighting that Avalanche is significantly cheaper and more scalable, making it the preferred choice for such platforms.

To facilitate user interaction, a web interface allows event organizers to manage events, create tickets, and set prices. Users engage with the platform through their MetaMask accounts, which are linked to their verified identities, ensuring a secure and seamless experience.

h) A Smart Contract - Based Mobile Ticketing System with Multi-Signature and Blockchain by Keng-Pei Lin, Yi-Wei Chang, Zhi-Hong Wei, Chih-Ya Shen and Ming-Yi Chang

The traditional mobile ticketing systems that rely on static QR codes face significant challenges, including counterfeiting, unauthorized reselling, and the potential for double issuance of tickets. To address these issues, this paper proposes a solution that leverages blockchain technology, smart contracts, and multi-signature mechanisms to ensure the authenticity and security of tickets.

The proposed system employs blockchain technology for a decentralized approach, enhancing security and transparency. Smart contracts play a crucial role by enabling automatic, self-enforcing transactions between buyers and sellers. The immutability of blockchain provides a transparent and traceable ledger for all transactions, ensuring accountability throughout the ticketing process.

To further enhance security, a multi-signature mechanism is implemented. In this setup, both the event host and the ticket buyer must sign the QR code ticket for it to be activated. This approach ensures that ownership is authenticated, effectively preventing ticket scalping and counterfeiting. Additionally, the multi-signature mechanism facilitates the handling of payments related to ticket sales.

The system architecture consists of several key components. First, the event host creates and sells tickets through smart contracts on the blockchain. Tickets are issued as QR codes, but require the buyer's signature for validation, making them both secure and verifiable. At the ticket gate, the smart contract is triggered when the buyer scans their QR code, finalizing the payment and confirming the ticket's validity.

For implementation, the system is built on EOSIO, a blockchain platform optimized for decentralized applications (DApps). EOSIO utilizes a delegated proof-of-stake consensus mechanism, which reduces transaction latency and increases throughput, making it well-suited for a high-transaction environment like ticketing.

The paper also discusses the costs associated with deploying smart contracts and operating the system on the EOSIO platform. Although the RAM needed for storing state information can be costly, the overall system remains efficient and low-cost for both event organizers and ticket buyers.

The advantages of the proposed system include its ability to prevent counterfeiting and unauthorized reselling, ensuring trust and transparency through the blockchain's immutable ledger. Additionally, it automates transactions with smart contracts, thereby reducing the need for intermediaries, and provides system stability by avoiding a single point of failure.

i) Applying Smart contract in Blockchain Technology to Manage the Ticketing Issuance and Ticketing Traceability by Huu-Quang Nguyen, Huynh Tuong Nguyen and Thi-Thiet Pham

The Industrial University of Ho Chi Minh City hosts annual events where students receive free tickets. However, challenges have emerged regarding fake tickets and unauthorized ticket sales, resulting in problems with ticket distribution and validity.

To address these issues, the authors propose leveraging blockchain technology and smart contracts on the Ethereum platform for efficient ticket management, including issuance, transfer, and traceability.

By utilizing blockchain, data immutability is ensured, which prevents unauthorized modifications and ticket fraud. Furthermore, smart contracts automate the ticketing process and transactions without the need for third-party intervention.

The system is designed for three types of users: students, collaborators, and administrators. Key functionalities include ticket creation and issuance, ticket transfer, and ticket verification. Tickets are issued to students based on their eligibility, and the entire ticketing process is recorded on the blockchain, ensuring traceability.

In terms of ticket transfer and verification, students can request ticket transfers, which require approval from collaborators. When entering events, students present their student IDs, and the system verifies ticket ownership using information stored on the blockchain.

Technologically, the system is built on the Ethereum blockchain with smart contracts written in Solidity. The front-end application utilizes Java and Web3.js, while Firebase manages real-time database operations. The application was implemented on the Android platform.

For system testing, real student data was used to manage 100 tickets for a university event. The results indicated that the system performed effectively, ensuring ticket validity, preventing fraud, and successfully managing ticket transfers.

3.2. Discussion

(RQ1): What programming languages might be available for development?

In the reviewed papers, languages such as Java, JavaScript, and Python were frequently mentioned. However, the most popular languages for Smart contract development are JavaScript and Solidity. Therefore, the chosen languages for this project will be a combination of JavaScript and Solidity.

(RQ2): Are there frameworks or plugins to support development?

Yes, there are several frameworks that support the development of Smart contracts. According to the analysed papers, the most popular frameworks are Truffle [30], Hardhat [31] and Ethereum. In terms of plugins, MetaMask is a preferred option for creating and managing ETH wallets.

For this project, Truffle and Ethereum were selected as the frameworks, while Ganache, a local EVM, will be used to simulate ETH transactions between accounts. Ganache also provides insights into mined blocks and the creation of Smart contracts.

(RQ3): What tools should you use for development?

The reviewed works indicate that development is commonly done using Visual Studio Code [32], which will be the chosen coding environment for this project.

(RQ4): Are there any developed blockchain based Smart contract solutions?

The studies examined were conducted in controlled environments rather than on the actual Ethereum network, primarily due to the excessive costs (or gas fees) associated with using the live network. This limited the scope of testing and results.

(RQ5): Are there any studies on blockchain based Smart contracts and E-Ticketing?

Making a comparison with the resumes above and the keys blockchain, Smart contract and e-ticketing the results are (picking the letter of paper analysed):

a) The provided paper context does not seem to contain information specifically about blockchain and Smart contracts in the context of e-ticketing. The summary focuses more on the general technical aspects and challenges of Smart contracts and blockchain technology.

b) This study focuses on the design and evaluation of different Smart contract architectures for efficiently indexing and querying blockchain data, particularly in the context of a ride-sharing application.
   The researchers explored two Smart contract designs - a "Sparse" contract that categorizes ride transactions based on stations, and a more sophisticated "Catalog" contract that manages a catalog of station-specific contracts.
   Through experiments on the Rinkeby Ethereum testnet, the study compared the complexity (measured by gas consumption) and retrieval performance of these two Smart contract designs. The results showed that the Catalog design had higher complexity but significantly improved the efficiency of querying the blockchain data compared to the Sparse design.
   This trade-off between Smart contract complexity and query performance is an important consideration for blockchain-based applications, such as e-ticketing systems, which require efficient data retrieval from the distributed ledger.

c) The text discusses the barriers in adopting blockchain technology and Smart contracts, such as high-power consumption, slow transaction rate, complex architecture, lack of awareness and expertise, legal issues, and high transaction costs.
   There is no information about e-ticketing or a specific study paper on that topic.
   The paper has many examples of frameworks and languages it can used in the project, like NodeJS, Solidity, JavaScript, etc.

d) In the paper discusses a blockchain-based Smart contract management system called SmartCon that was developed in 2015. It is free for open-source projects and uses two separate blockchains - one to store the Smart contracts (SBlockchain) and one to store the transactions (TBlockchain).
   SmartCon supports multi-party Smart contracts and can perform over 10,000 transactions per second. It is a private/permissioned blockchain and its usage spans across various industries.
   The consensus engine used is Raft and the block time and transactions per block are configurable. The information in the context was gathered from official websites, whitepapers, forums, and articles related to Smart contract management systems.
   The comparative analysis focused on specific characteristics that can help organizations choose the right technology for their needs.

Overall, the SmartCon framework provides a unified architecture that supports both Decentralized Autonomous Organizations (DAO) and organizational-level blockchain-based Smart contract execution.

The two-blockchain approach aims to improve scalability and efficiency compared to single-blockchain designs.

e) This study paper primarily focuses on Smart contracts and their implementation within the blockchain framework.

It delves into the evolution and technical aspects of Smart contracts, emphasizing their digital nature and the use of platforms like Ethereum and Solidity programming language for deployment.

While the paper does not explicitly mention e-ticketing, it provides a comprehensive overview of Smart contracts, their types, working principles, and the Ethereum Virtual Machine's role in executing Smart contract code.

f) The study paper primarily focuses on blockchain and Smart contracts, discussing the concept, applications, working principles, limitations, and future scope of Smart contracts.

While the paper does not specifically address e-ticketing, it provides a comprehensive analysis of Smart contracts and their potential impact across various domains, including decentralized finance, supply chain management, digital identity, real estate, and healthcare.

The paper outlines the key benefits of using Smart contracts, such as autonomy, cost savings, security, speed, accuracy, and the ability to enable multi-signature accounts. It explains the working of Smart contracts, which use simple logic like "IF-THEN" conditions to trigger actions when certain predetermined conditions are met, such as transferring digital assets upon completion of a task.

The paper also discusses the limitations of Smart contracts, such as lack of regulation, complexity in writing them, immutability, and potential scalability issues. However, the paper concludes that the future scope of Smart contracts is promising, as they offer several advantages over traditional contractual agreements, and their use is likely to expand as blockchain technology becomes more widely adopted.

g) This study paper focuses on the use of blockchain technology, Smart contracts, and electronic ticketing systems. The blockchain-based ticket sales system offers several significant benefits compared to traditional systems.

Firstly, by utilizing Smart contracts and non-fungible tokens (NFT), the system ensures greater transparency and security in ticket distribution and resale, thus reducing the risk of fraud and the presence of counterfeit tickets in the secondary market.

Additionally, being blockchain-based, the system eliminates centralization, meaning there is no single point of failure and avoids censorship or manipulation by a central authority. Another key benefit is efficiency and cost reduction.

The implementation on the Avalanche blockchain is much more economical than on the Ethereum blockchain, enabling more cost-effective transactions and more efficient deployment.

Furthermore, by offering a secondary market for ticket resale, the system provides a decentralized and open platform for buyers and sellers, eliminating the issues associated with purchasing tickets from unauthorized resellers.

h) This the study paper is centred around the development of a blockchain-based mobile ticketing system that leverages Smart contracts and multi-signature mechanisms to enforce and authorize ticket transactions.

The system is implemented on the EOSIO blockchain platform, which utilizes a delegated proof-of-stake (DPoS) consensus mechanism. Smart contracts are used to establish binding agreements between sellers and buyers, ensuring the integrity of transactions related to ticket sales, purchases, and usage.

These Smart contracts are accompanied by Ricardian contracts for dispute resolution. Multi-signature is employed to guarantee the authenticity and ownership of tickets, preventing issues such as counterfeiting and ticket scalping.

The blockchain's immutable ledger records all ticket-related transactions, providing traceability and irreversibility, which aids in dispute resolution. Furthermore, the decentralized nature of the blockchain-based system prevents single points of failure and monopolistic control.

The experimental results demonstrate that the proposed system is efficient and has a nominal cost of operation on the EOSIO platform.

i) The document describes a system for managing ticket issuance and traceability using blockchain technology and Smart contracts. The system is designed to address issues with fake tickets and unauthorized ticket distribution at events organized by the Industrial University of Ho Chi Minh City. The system uses blockchain technology to store and manage ticket information, ensuring the integrity and traceability of ticket data.

The system includes functionalities for ticket issuance, ticket transfer, ticket verification, and ticket traceability. The system was tested with real student data from the Faculty of Information Technology at the Industrial University of Ho Chi Minh City, and the results show that the system performs all the required functions effectively.

3.3. Review Paper's Conclusion

Unfortunately, there is a limited number of articles specifically addressing Smart contracts and E-ticking. Consequently, the research was extended to include studies on Smart contracts and blockchain technology more broadly.

The available research indicates that blockchain technology and Smart contracts hold significant potential for enhancing e-ticketing systems. By leveraging the

immutable and decentralized nature of blockchain, these solutions can address longstanding issues like ticket fraud, unauthorized resale, and lack of transparency.

Smart contracts enable the automated enforcement of rules and policies around ticket issuance, transfer, and verification. This helps streamline the ticketing process and reduces the potential for disputes.

Furthermore, integrating blockchain and Smart contracts allows e-ticketing systems to incorporate additional features like secondary ticket markets and multi-signature mechanisms. These functionalities further enhance security, user experience, and overall system integrity.

However, the adoption of these blockchain-based e-ticketing solutions remains limited so far. The studies referenced in the context are relatively few, suggesting that more research and real-world implementation may be needed to fully understand the trade-offs and best practices for deploying such systems at scale. Nonetheless, the available evidence points to the promising future of blockchain and Smart contracts in transforming the e-ticketing landscape.

The analysis of the literature indicates that research on this topic spans from 1981 to 2024, with a noticeable increase in interest beginning in 2019. The volume of published articles has grown each year.

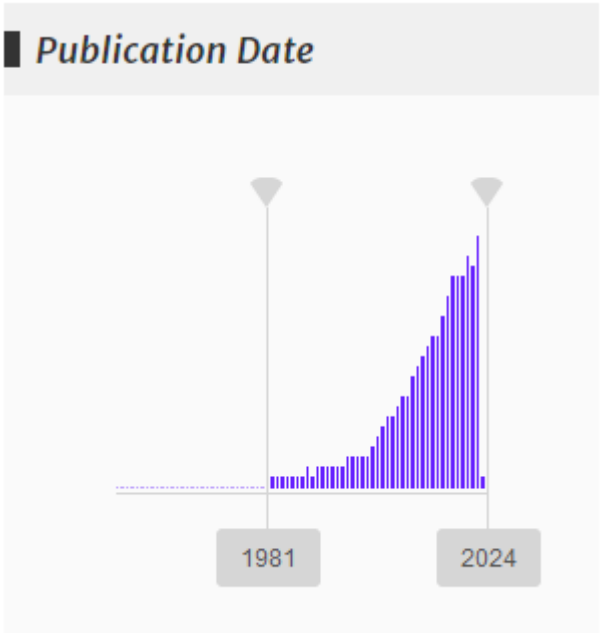

Figure 2 – Evolution of the number of articles per year on the IEEE Xplore website for the search term “blockchain Smart contracts”.

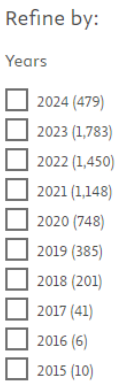

Figure 3 – Evolution of the number of articles per year in Science Direct for the search term "blockchain Smart contracts".

This project has an advantage compared to the papers analysed, will be implemented and the results will be objective and visible. With this information can potential the best way to improve systems like this, and the researchers or programmers can see the effects at the moment.

3.4. Chapter Conclusion

Blockchain technology began to attract academic and practical interest several years ago, but it wasn't until 2016 that its popularity surged. This increase in interest can largely be attributed to the advent of Bitcoin in 2008.

Bitcoin's launch exposed various flaws in its underlying system, prompting the development of solutions that have made blockchain technology more secure and effective over time.

In this context, Smart contracts emerged. Utilizing blockchain to store these contracts allows them to function like traditional physical contracts but with enhanced efficiency.

Smart contracts offer faster execution, reduced costs, and greater transparency. Their decentralized nature prevents fraud and tampering, as they are distributed across a network rather than being controlled by a single entity.

After examining the papers with PRISMA methodology, the best way to structure a E-Ticketing project is implementing languages such as JavaScript, Solidity, or Java, the most used. Additionally, needs to aggregate frameworks and platforms like Ethereum, Hard Hat, Ganache or Truffle are play crucial roles in the development and deployment of blockchain applications.

In conclusion, blockchain technology has gained significant traction since the launch of Bitcoin in 2008, particularly following its surge in popularity in 2016. This evolution has led to the identification and rectification of various flaws within its initial framework, enhancing the security and effectiveness of blockchain systems.

Smart contracts have emerged as a pivotal innovation, leveraging blockchain's capabilities to provide efficient, cost-effective, and transparent contractual solutions. Their decentralized nature mitigates risks of fraud and tampering, further solidifying their utility.

Through a thorough analysis using the PRISMA methodology, this study highlights that the most effective approach for structuring an E-Ticketing project involves the use of programming languages such as JavaScript, Solidity, and Java, along with essential frameworks and platforms like Ethereum, Hard Hat, Ganache, and Truffle, which are vital for the development and deployment of blockchain applications.

4. System Design

System design is a critical aspect of engineering and computer science that involves defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements.

The background, located in system design in this dissertation, would typically include an overview of the principles and methodologies used in system design, such as object-oriented design, service-oriented architecture, and model-driven engineering. It would also discuss the importance of system design in ensuring the scalability, reliability, and maintainability of complex systems.

By providing a comprehensive analysis of system design, this dissertation aims to contribute to the field by offering insights into effective design strategies and identifying areas for future research. This study is significant as it not only enhances our understanding of system design but also provides practical guidelines for designing robust and efficient systems in various domains.

Therefore, the section 4.1, delves into the essential components of the system, providing a comprehensive overview of its architecture and functionality. The system comprises several key components that work in tandem to ensure efficient operation and user interaction. To facilitate understanding,

In section 4.2, UML modelling techniques are employed, including user/customer cases that outline the various interactions between users and the system. Robustness diagrams are used to illustrate the system's response to various scenarios, ensuring that all potential use cases are considered. Additionally, class diagrams provide a structured representation of the system's entities, their attributes, and relationships, while sequence diagrams detail the order of operations and interactions over time.

Data modelling, presented in 4.3, is another critical aspect of system design, involving the creation of an Entity-Relationship (ER) model that visually represents the data and its relationships within the system. Normalization processes are then applied to ensure that the database structure is efficient, reducing redundancy and improving data integrity.

Finally, interface prototyping, described in 4.4, is introduced to refine the user experience. Various tools are utilized for modelling, allowing for the creation of prototypes that visually demonstrate the user interface and interactions. This approach not only enhances the design process but also enables stakeholders to visualize the final product, leading to informed decision-making throughout the development lifecycle.

4.1. System Components

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 4 shows 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.

Figure 4 – Explanation of system behaviour.

4.2. UML Modelling

UML is a standardized visual modelling language used for specifying, visualizing, constructing, and documenting the artifacts of a software system. It provides a set of graphical notations to represent different aspects of a system, including its structure, behaviour, interactions, and architecture.

In this project are used some key UML diagrams:

- **Use Case Diagrams:** These capture functional requirements by depicting actors, use cases, and their relationships.
- **Robustness diagrams:** The goal is to capture, for each use case, the main objects and their communication relationships.
- **Sequence Diagrams:** These illustrate interactions between objects over time, emphasizing the order of messages exchanged during a specific use case.
- **Class Diagrams:** These depict the static structure of the system, showing classes, their attributes, methods, and relationships.

4.2.1. User/Customer cases

A use case typically describes a specific scenario or task that a customer or actor wants to accomplish with the system. It outlines the steps involved, the interactions between the customer and the system, and any potential alternative paths or variations, like demonstrated in Figure 5 for present system.

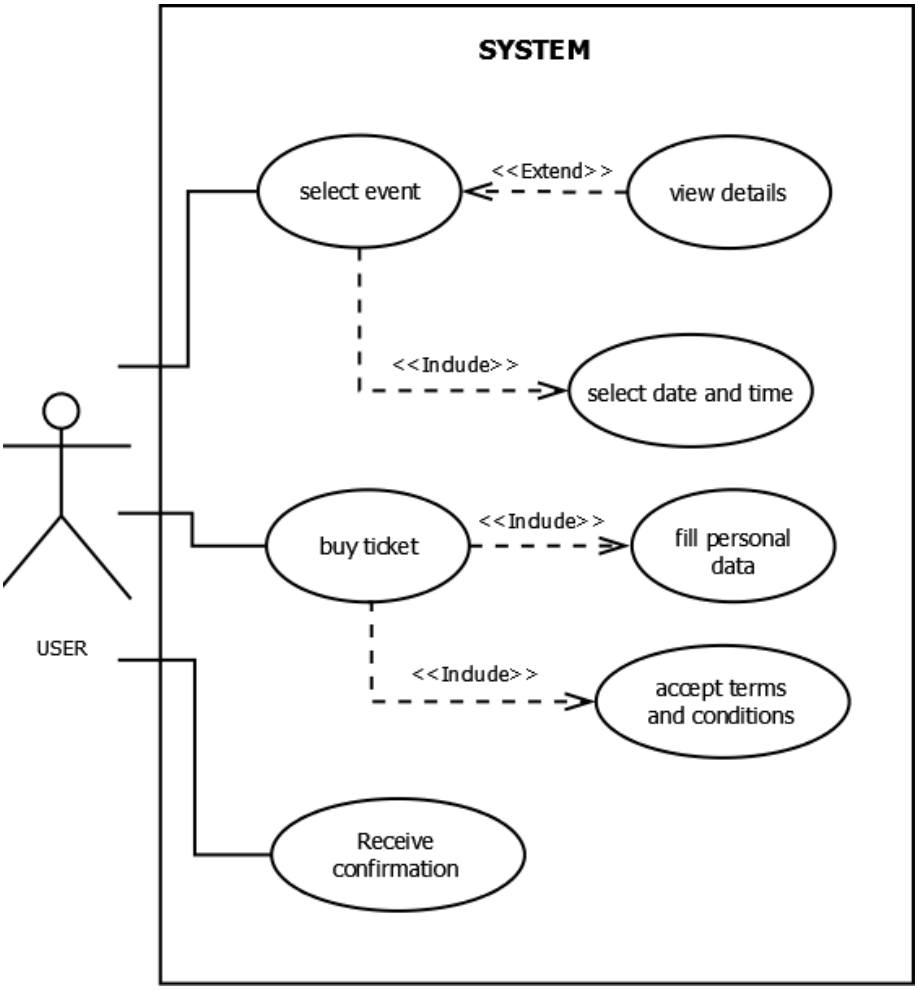

Figure 5 – Customer case for interface.

Description: When the user selects an event and clicks on "Details," a new interface opens displaying the available dates and times for that event. If the user agrees with the displayed price for the chosen date and time, they can click "Buy" to proceed to the payment webpage.

On this page, the user will need to enter their personal information, accept the terms and conditions, and complete the payment process. Upon successful completion, the user will receive a confirmation of their purchase.

4.2.2. Robustness diagrams

In robustness diagrams, the goal is to capture, for each use case, the main objects, and their communication relationships. It includes the following elements, as shown in Figure 6:

- **Boundary Objects/Interfaces:** These allow communication between actors and the system.
- **Entity Objects:** These typically correspond to the objects identified in the domain model.
- **Control Objects:** These integrate boundary objects and entity objects.

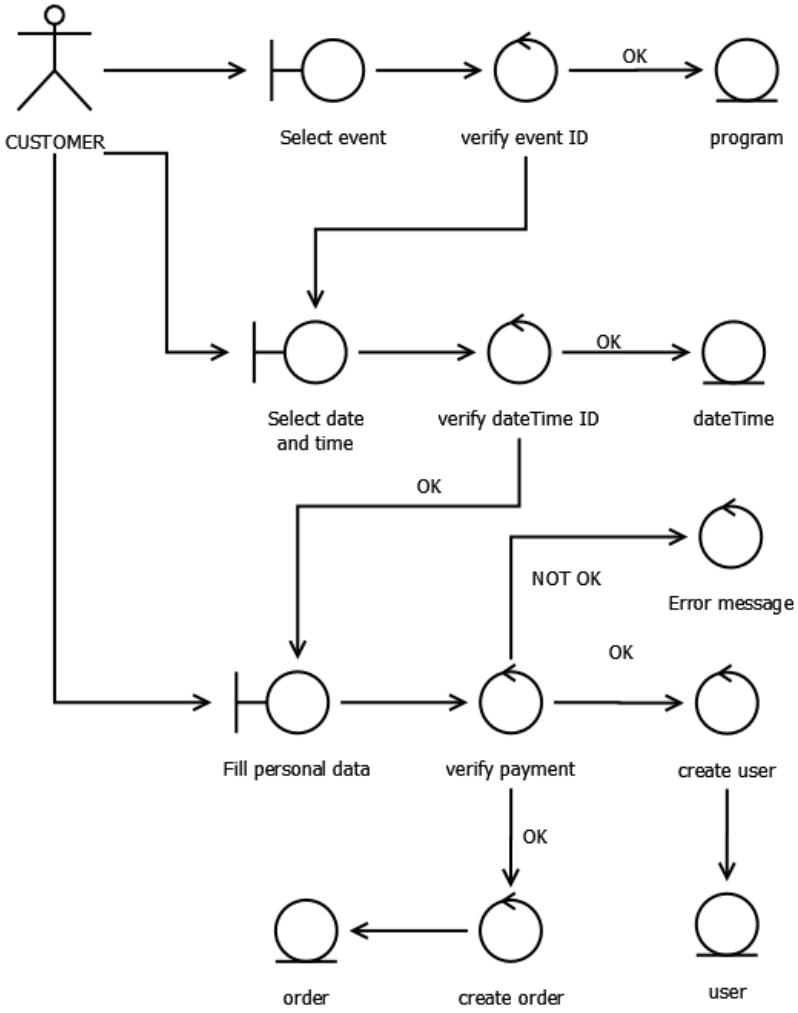

Figure 6 - System procedure to select, buy, and pay a ticket.

Description: The customer selects an event on the webpage and chooses a specific date and time.

The system verifies and validates this information. If is correct, the event price will be displayed on the webpage.

Finally, when the customer is ready to purchase the ticket, they fill in their personal details on the webpage and click "Finish." The system then verifies and validates the payment, creates an order, and registers the customer.

If the payment cannot be processed, an error message will be displayed to the customer. The customer selects an event in the webpage, the system verifies that event, if not exists the customer does not see it.

After that, the customer selects a date and time for that specific program, one more time the system verifies and validate it, if everything is ok, shows the price of the event in the webpage, if cannot do it, the date time will not show up and the customer cannot proceed.

Finally, the customer wants to pay the ticket, in the webpage fills personal data inputs, and clicks “Finish”, the system verifies and validate the payment, creating an order and a customer, if cannot do it, sends a message error to customer.

4.2.3. Sequence diagrams

Sequence diagrams follow a temporal order and are facilitated by messages, demonstrated in Figure 7. These diagrams are constructed based on the use case diagram but represent the execution and timeline for each specific use case. First, the system’s role is defined, and then it describes how the software will fulfil that role.

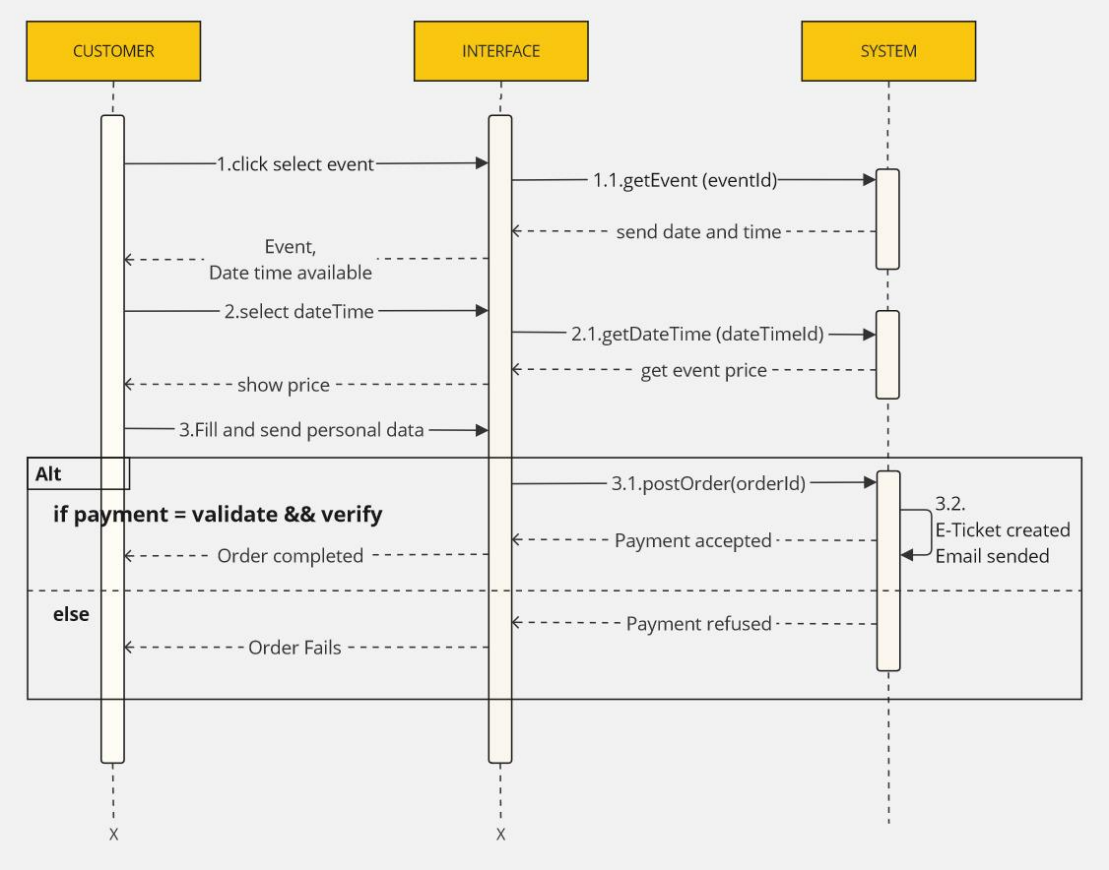

Figure 7 – Sequence diagram for E-ticketing.

Description: The customer opens the event webpage and select an event (1), the information is response to the system processes (1.1). Selects the date and the time of the program/event (2), the system processes the response (2.1), and the price is showed up.

The customer finishes the process filling and send the data form, (3), the system response and try to create an order (3.1), the payment will be verified and validate for the system:

If the payment is successful, the system creates a e-ticket, post the data to the blockchain, and send an email with the ticket information's (3.2).

If the payment fails, the system sends to customer an error message.

4.2.4. Class diagram

A class diagram defines all the classes that the system requires for constructing communication, sequence, and state diagrams. It serves as a bridge between the system and the database.

This study allows for designing the necessary database to connect the different entities required for the system's operation. It's possible to see all these elements in Figure 8.

The structure of a class diagram includes:

- Associations (lines and arrows).
- Classes (boxes).
- Attributes (class characteristics, including visibility: private, public, protected, and package).
- Operations (functions required by an abstract object) and composition (relationships between classes).

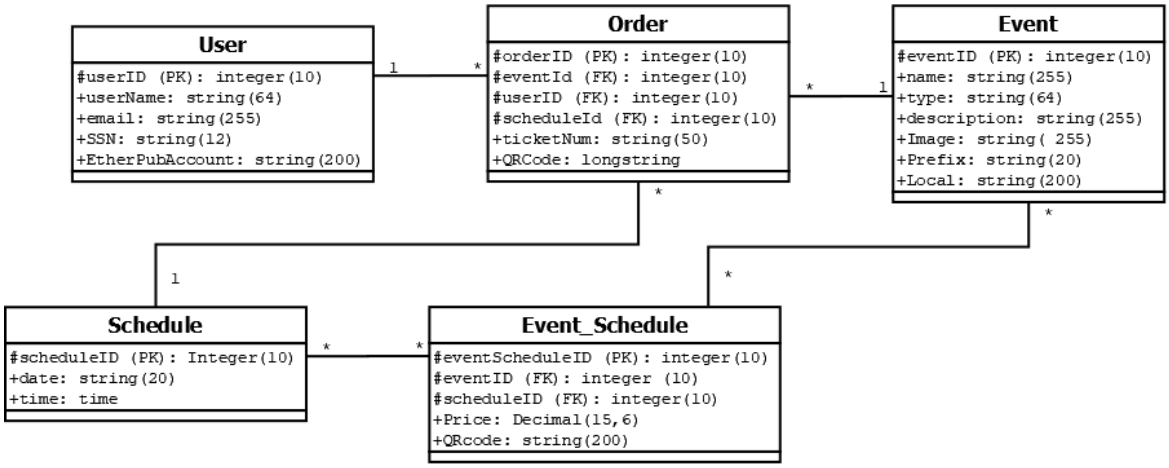

Figure 8 – Class diagram for E-Ticketing data base.

Description: Schematic composition of the interventions and interconnections necessary to establish communication between the different structures.

The tables Event, Schedule and Event_schedule is filled previously, all that information must be in the database to ensure the functionality. The tables user and orders will be generated and recorded when necessary.

4.3. Data Modelling

Data Modelling one of the key features of File Management Systems is that each application is responsible for creating and managing the internal structure of its own files, avoiding redundancies and data inconsistencies. Data storage and organization management involve analysing the Entity-Relationship (ER) model, ER model details, normalization, the relational model, and specifics related to this model.

4.3.1. ER model

The Figure 9, is an Entity-Relationship (ER) model which helps identify all the relevant entities and relationships for the situation being analysed. It also assists in identifying the relevant attributes of each entity and associating them with their respective relationships.

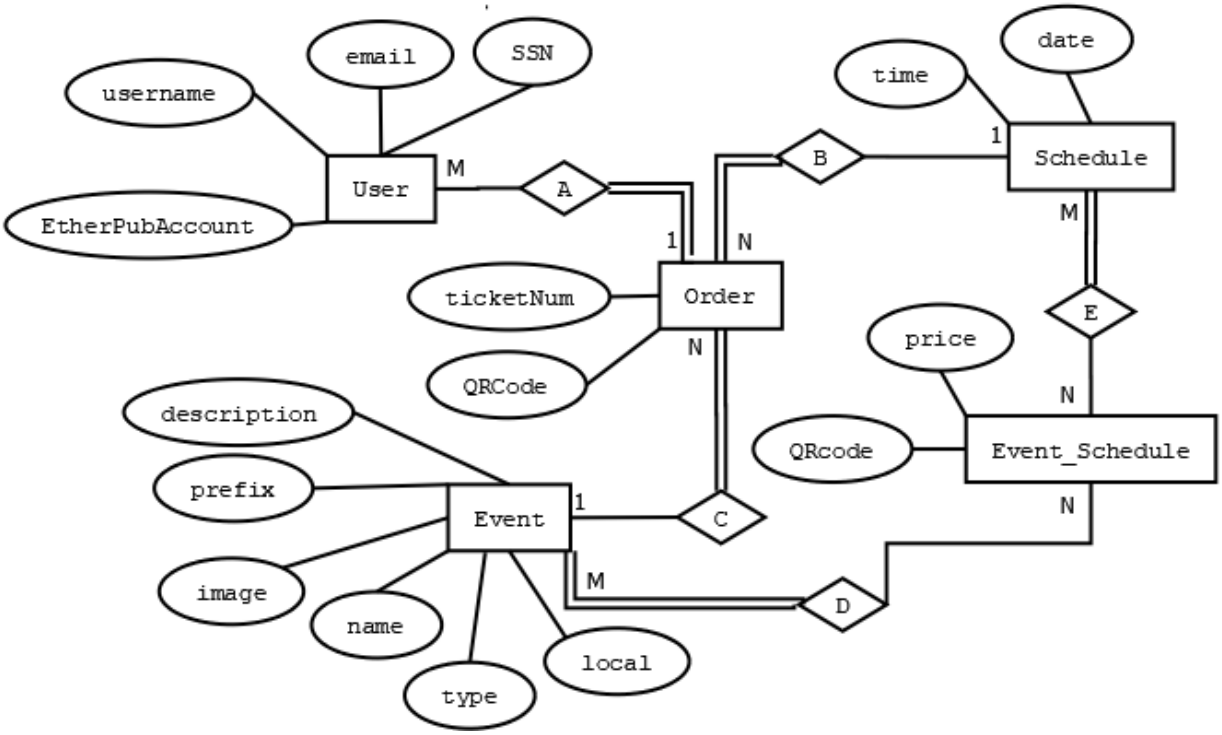

Figure 9 – ER model for E-Ticketing.

Description: The *Order* is the principal entity, grouping the other entities around, this entity has connections with *Event*, *Customer* and *Schedule* entities. The order must have a date and time creation. Schedule has connection with Event, a relation many to many origins a third entity *Event_Schedule*, this entity groups the foreign keys of the 2 entities and have the entity "price".

- *Customers* have the attributes: EtherPubAccount, username, email and SSN.
- *Schedule* have the attributes date and time.
- *Events* have the attributes name, type, description, prefix, local and image.
- *Event_Schedule* have the attribute price and Qrcode.
- *Order* have the attribute ticketNum and QRCode.

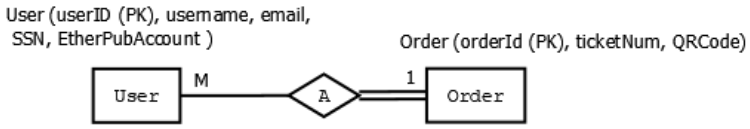

Figure 10 – Relation between Customer and Order.

Description: the "A" represents Create, Figure 10.
An order creates a customer, and an order is created if a customer is created.
Only exists customers if exists orders.

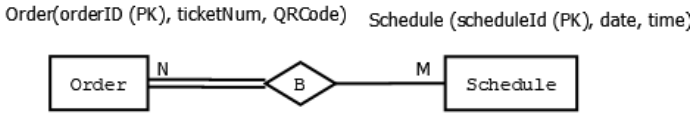

Figure 11 – Relation between order and schedule.

Description: the "B" represents Has, Figure 11.
An order has a schedule, and an order only exists if exists a schedule.

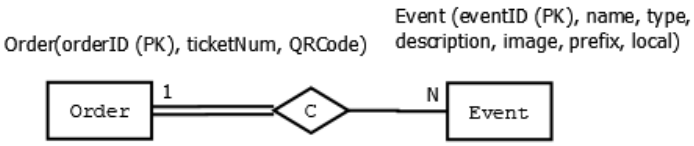

Figure 12 – Relation between order and event.

Description: the "C" represents Has, Figure 12.
An order has an event, and an order only exists if exists an event.

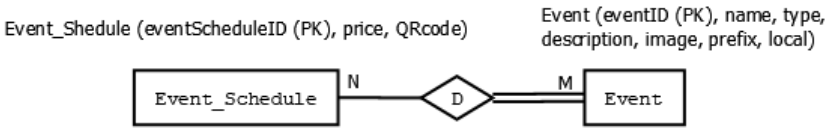

Figure 13 – Relation between Event_Schedule and Event.

Description: the "D" represents Contain, Figure 13.
An Event_Schedule must contain an event.

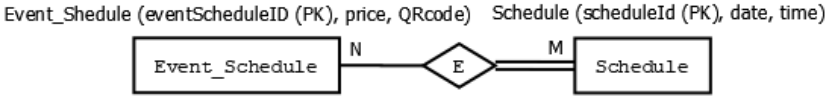

Figure 14 – Relation between Event-Schedule and Schedule.

Description: the "E" represents Contain, Figure 14.
An Event_Schedule must contain a schedule.

4.3.2. Normalization

Normalization is the process of defining the logical format for data structures identified in the logical design of a system. Its goal is to minimize the space used by data and ensure the integrity and reliability of information.

The objectives of normalization include reducing or eliminating data redundancies, improving system performance, and enabling referential integrity mechanisms between entities.

- User (userID (PK), etherPubAccount, email, name, SSN).
- Event (eventID (PK), description, prefix, image, name, type, local).
- Schedule (scheduleID(PK), date, time).
- Order (orderID (PK), ticketNum, QRCode, eventID(FK), customerID(FK), scheduleD(FK)) -> A, B, C.
- Event_Schedule(event_ScheduleID(PK), price, QRcode, scheduleID (FK), eventide(FK)) -> D, E.

4.3.3. Relational model

The relational model, Figure 15, is constructed based on the complete ER model and is represented by tables. It relies on the concept of relations, where a relation is a special type of table. In this model, there exists a set of relations or tables.

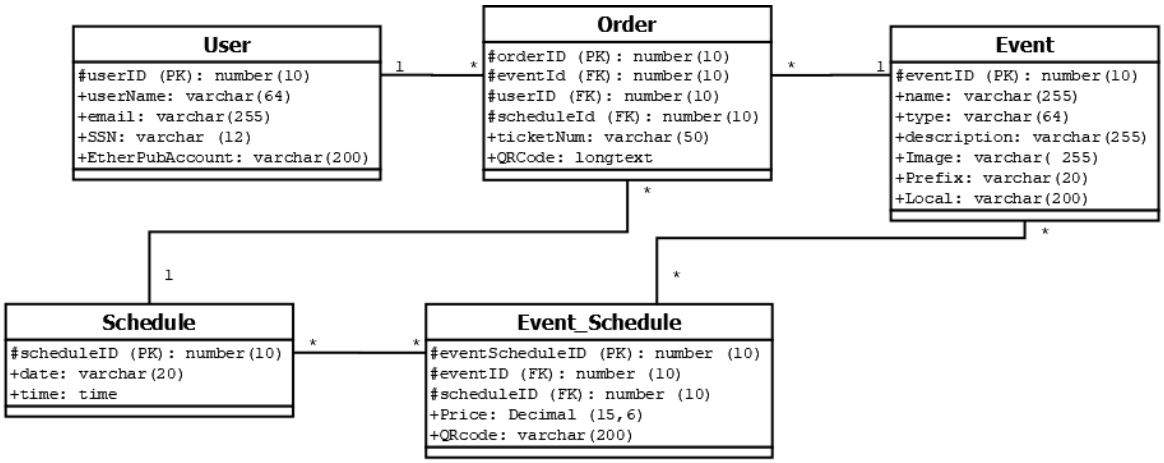

Figure 15 – Relational Model for E-ticketing.

Description: the customer can have multiple orders, but an order have only one customer.

The schedule is present in many orders, but an order only have one schedule associated at the time, the same happens to event and the order. This connecting exists to supply the information needed when a ticket is created.

An *event_schedule* have multiples schedules and events, and schedule is present in multiples *event_schedule*, the same happens to event and *event_schedule*.

Table 4 – Customer: Keep track of customer registrations.

Field	Data type	Description	Observations
userID	INT (Auto-increment, 10)	Unique	Primary Key, not null
EtherPubAccount	Varchar (200)	Public account	Not null
Email	Varchar (255)	Customer email	Not null
Name	Varchar (64)	Customer name	Not null
SSN	Varchar (12)	tax number	Not null

Table 5 – Order: Keep track of order registrations.

Field	Data type	Description	Observations
orderID	INT (Auto-increment, 10)	Unique	Primary Key, not null
ticketNum	Varchar (50)	Random generation	Null
QRCode	longtext	Qrcode codified	Null

Table 6 – Event: keep track of program registrations.

Fields	Data type	Description	Observations
eventID	INT (Auto-increment, 10)	Unique	Primary Key, not null
Name	Varchar (255)	Event name	Not null
Type	Varchar (64)	Event type	Not null
Description	Varchar (255)	Event description	Not null
Image	Varchar (255)	Event image	Not null
Prefix	Varchar (20)	Event prefix	Not null
local	Varchar (200)	Event local	Not null

Table 7 – Schedule: keep track of schedule registrations.

Fields	Data type	Description	Observations
scheduleID	INT(Auto-increment, 10)	Unique	Primary Key, not null
Date	Varchar (20)	Event date	Not null
Time	Time	Event time	Not null

Table 8 – Event_Schedule: keep track of event_schedule registrations.

Fields	Data type	Description	Observations
event_scheduleID	INT(Auto-increment, 10)	Unique	Primary Key, not null
Price	Decimal	Event ETH price	Not null
QRcode	Varchar (200)	QRcode image	Null

4.4. Interface Prototyping

An interface prototype is an early version or mock-up of a user interface for a software application, website, or other interactive system. It visually represents how the final interface will look and function, allowing designers, developers, and stakeholders to evaluate its usability, functionality, and aesthetics before committing to full-scale development.

In this study, the interface prototype focuses on a customer interface designed to facilitate the online purchase of e-tickets. Figure 16 outlines the steps involved in each customer action throughout the purchase process.

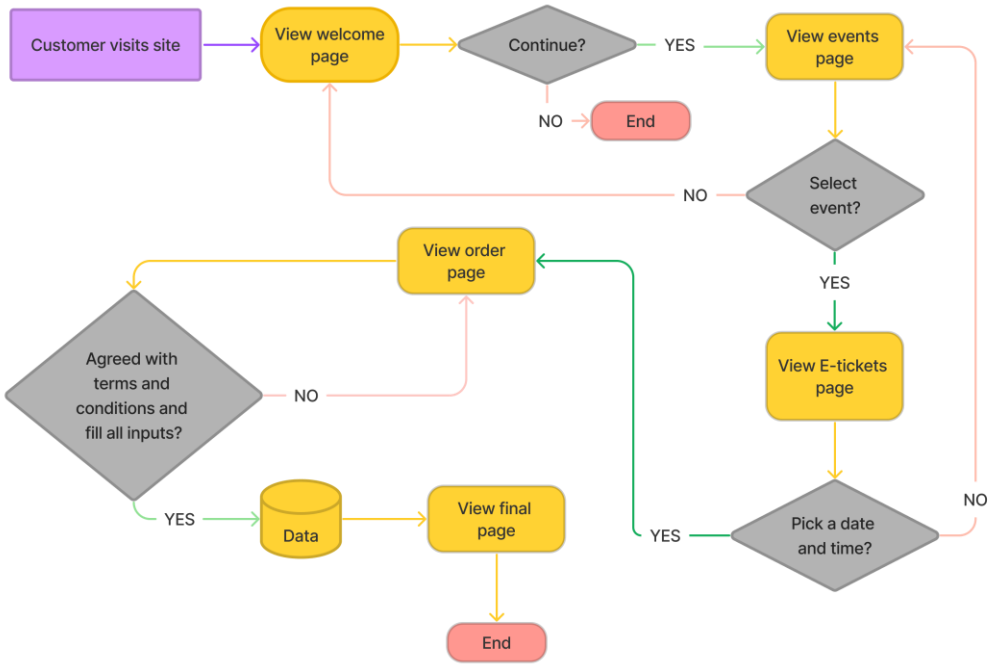

Figure 16 - Flow customer interactions with interface.

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 order page, agree with the conditions, submit the Ethereum account and fill 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.

The next sequence of figures, Figure 17 to Figure 27, presents the several interface screens available to the user, some potential errors and fixes are demonstrated and studied in each stage.

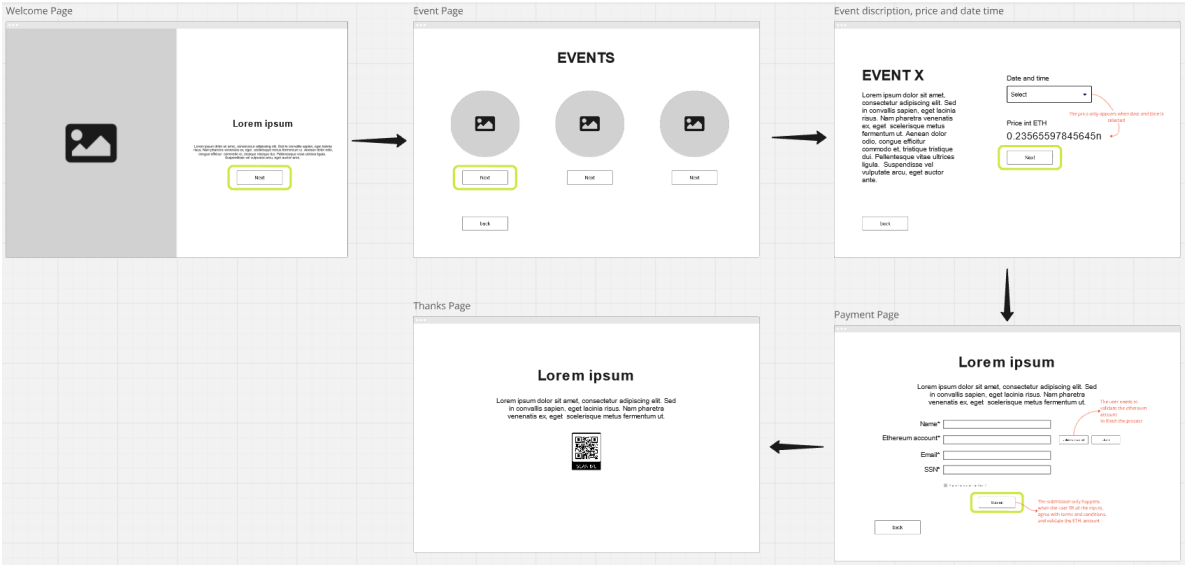

Figure 17 – Customer interface transitions of our system prototype.

Description: Customer costumer interface graphic flow, from the beginning to the end of the process without errors.

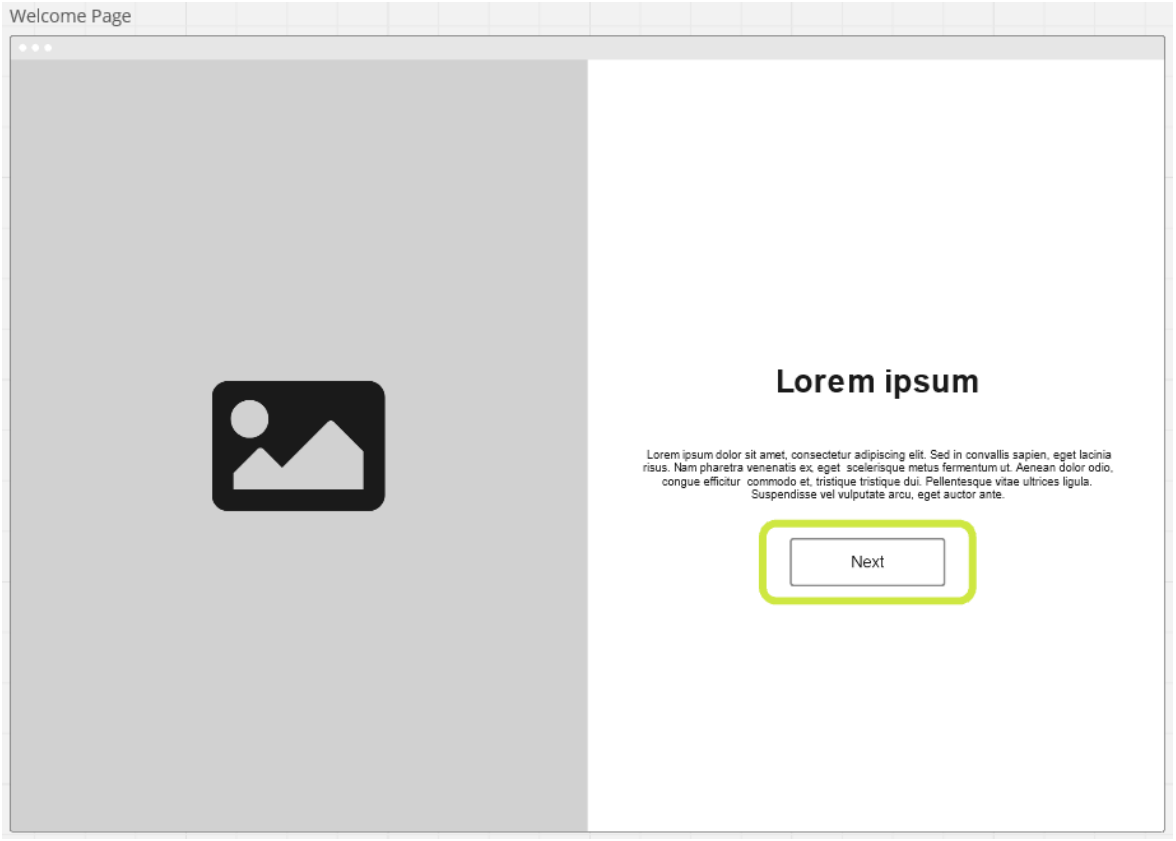

Figure 18 – Welcome screen.

Description: Welcome screen with one button (green highlight), when clicked the customer goes to the event list screen.

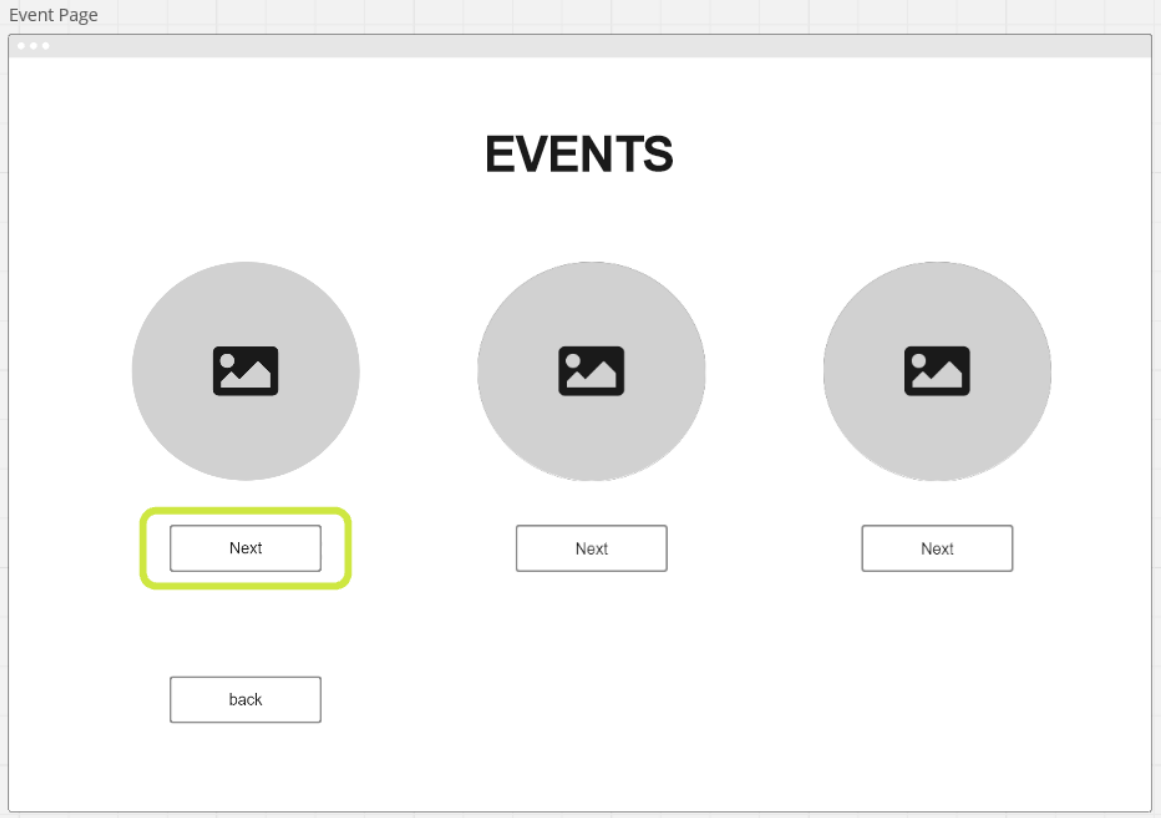

Figure 19 - List screen.

Description: The customer looks the options and select the button with green highlight in this example.

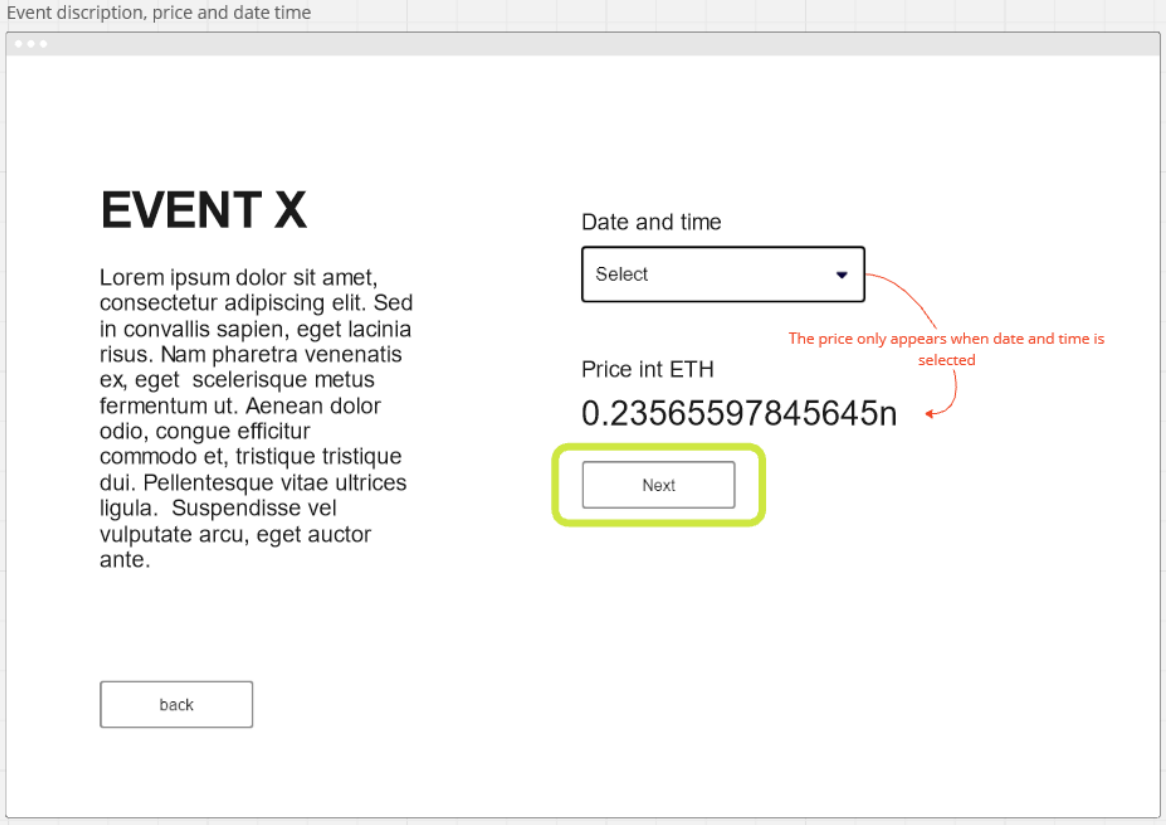

Figure 20 – Description and time sample screen.

Description: The customer can see a description, schedule, and price of the event selected in the previous screen and clicks “buy or next” (green highlight) to proceed to the next step.

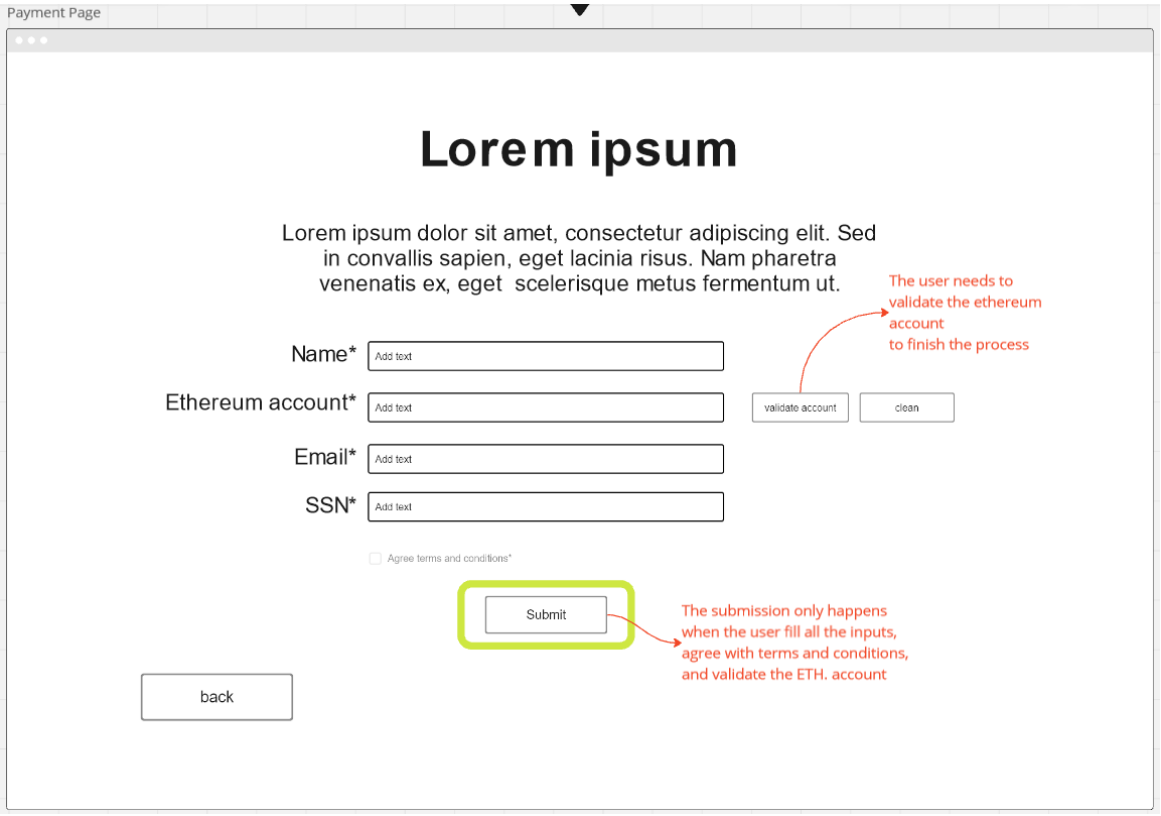

Figure 21 - Payment screen.

Description: The customer fills the inputs present in the webpage, the Ethereum account have a special treatment, the customer needs to validate the account first to unlock the button submit or finish (green highlight). Must agree with the terms and condition too.

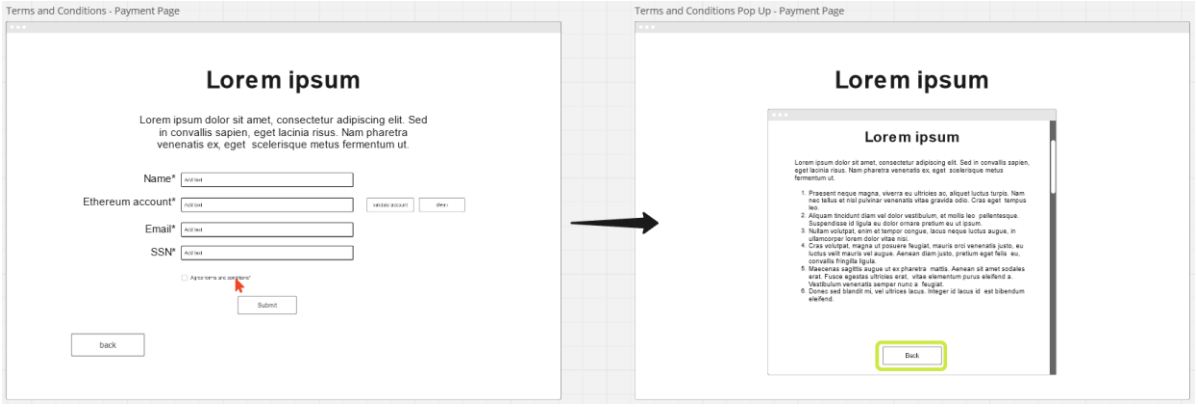

Figure 22 - Agreement cycle.

Description: The customer must agree with the terms and conditions, if wants to read the terms must click in the link of terms and conditions (mouse pointer with red colour), and a new webpage is showed up. When the customer stops reading the terms and want to return in the previous page, clicks in back (green highlight).

Figure 23 – Thanks screen.

Description: When customer submits the order, a validation message screen is visible for customer and one QRcode appears with an event location information. If clicks in back button, the customer goes to the welcome screen page.

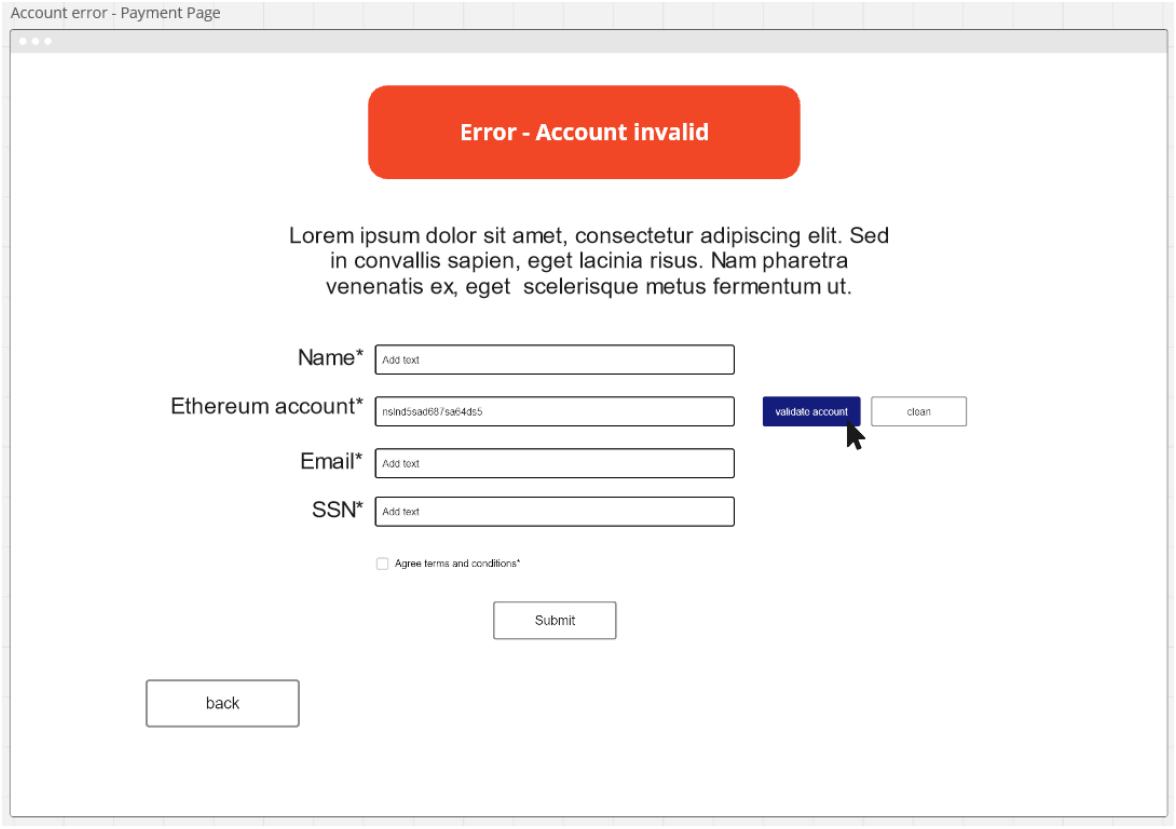

Figure 24 - Validation account.

Description: When customer clicks on validate account, the system will verify and validate the customer Ethereum account, if the account does not exist, shows an error message.

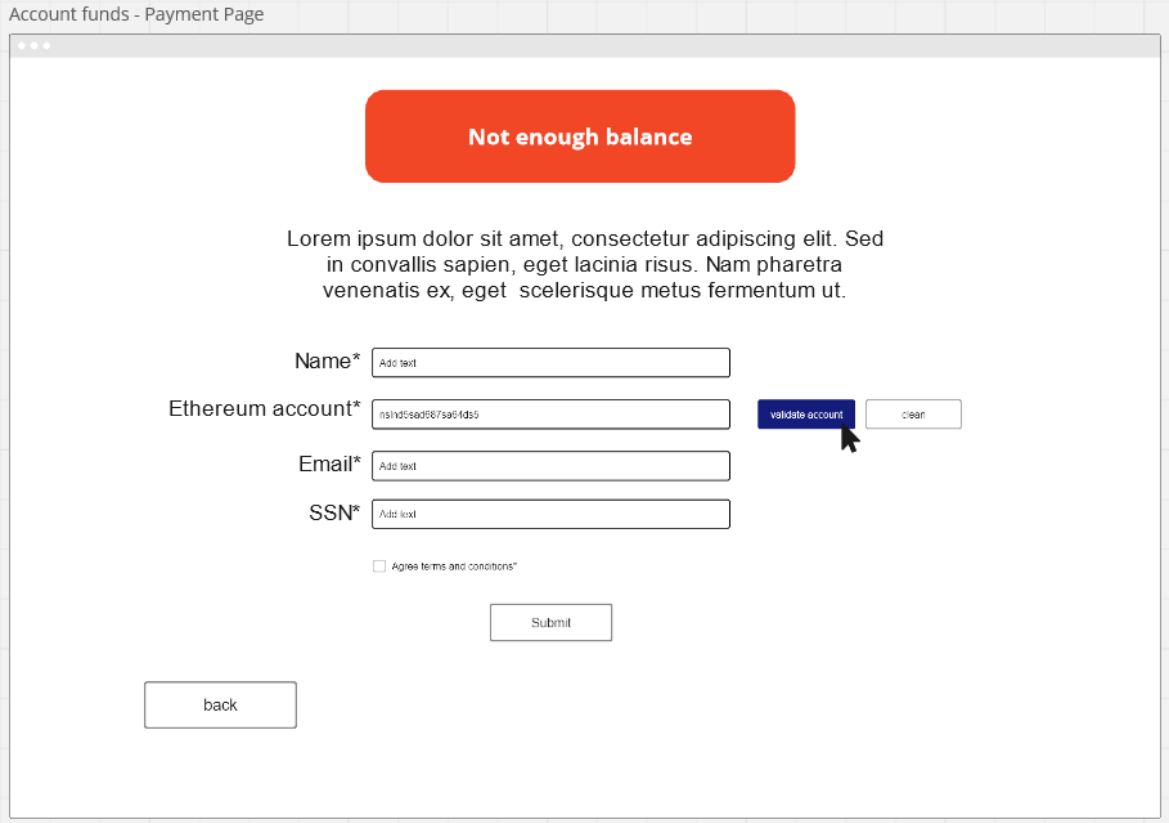

Figure 25 – Verify funds account.

Description: If the Ethereum account does not have enough funds an error message is showed up.

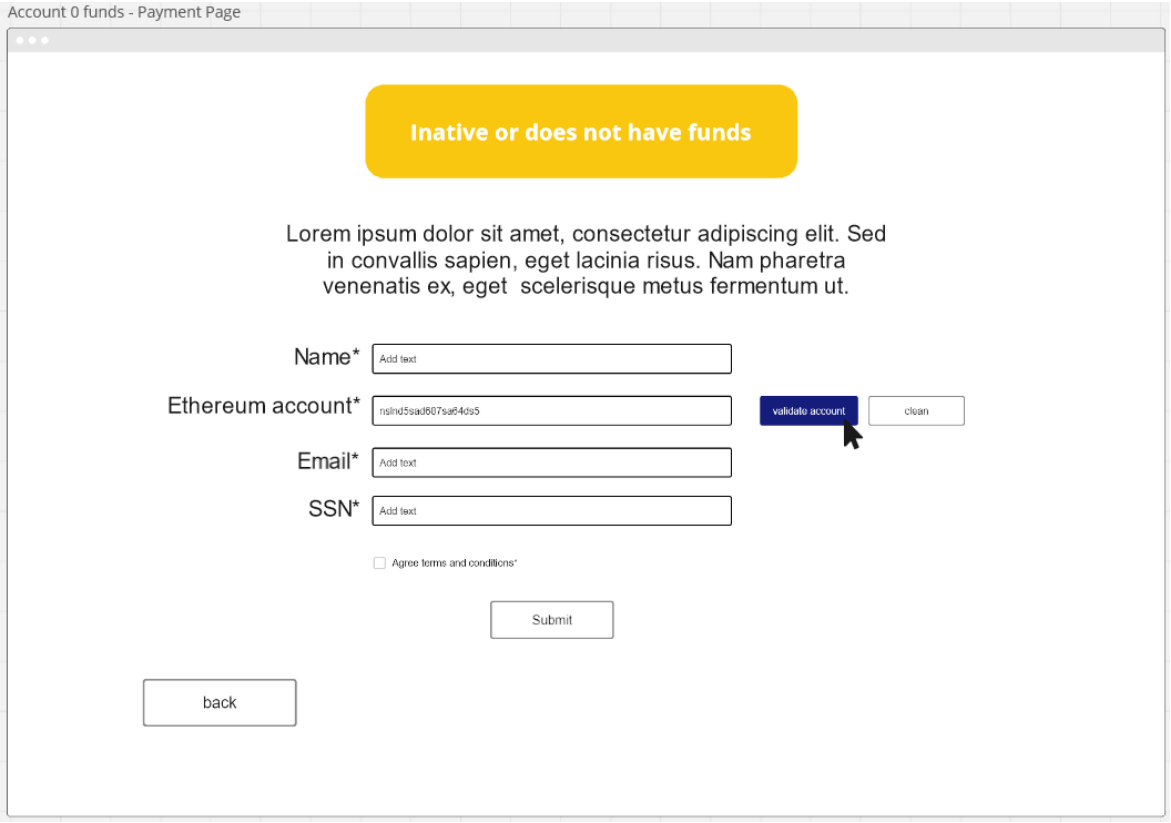

Figure 26 – Verify funds account, in this case with 0 funds.

Description: If the Ethereum account is inactive or does not have funds a warning message is showed up.

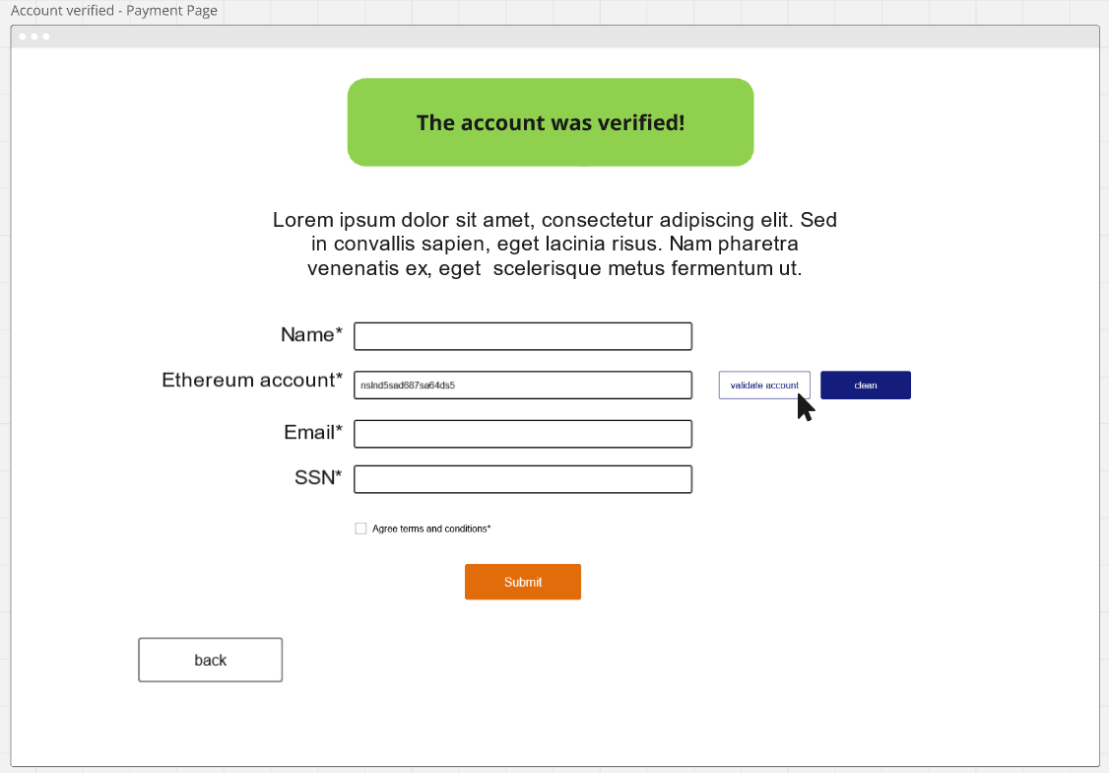

Figure 27 - Verified account.

Description: If the Ethereum account is validated and verified a valid message is showed up, the clean and submit buttons are activated.

4.4.1. Colour study

The colours select for the layout are [33]:

RGB (252, 208, 172)
#fcd0ac
hsl(27, 93%, 83%)

The colour transmits serenity, interest, and optimism.

Gradient
RGB (99, 130, 158)
#63829e
HSL (208, 23%, 50%)

RGB (255, 183, 183)
#ffb7b7
HSL (0, 100%, 86%)

The joined colours transmit excitement, high energy (blues) and slow breaths (pinks).

Font family in Figure 28: 'Rubik', sans-serif from google fonts [34].

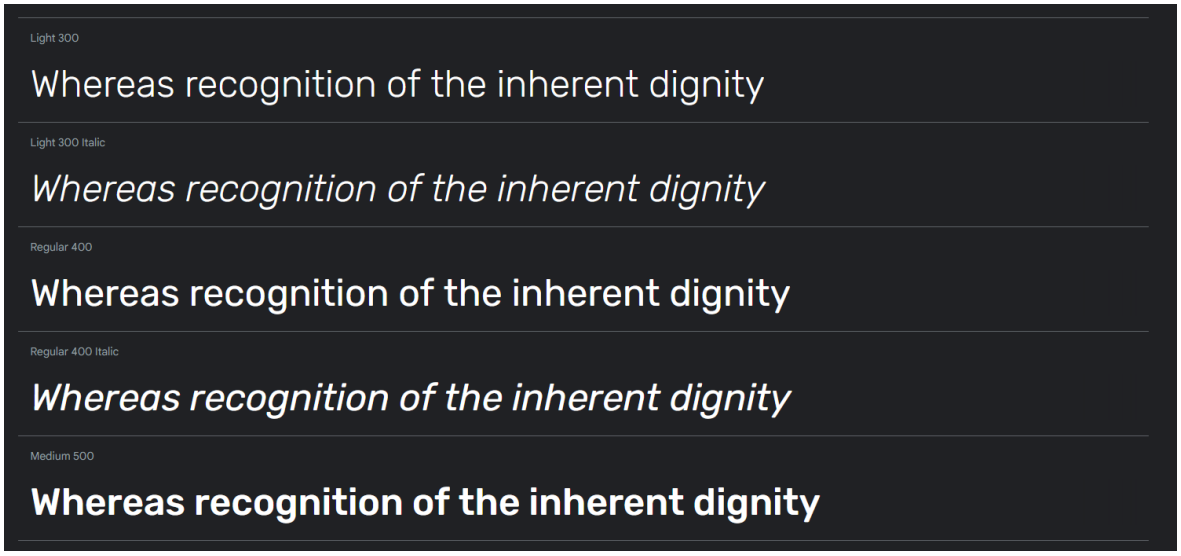

Figure 28 – Rubik font samples from google fonts.

Buttons colours:

Alarm, important, adventure, sociability

Peace, trust, loyalty

Message colours:

Accepted, free

Attention, awake awareness

Alarm, important

4.4.2. Tools used for modelling

For the preparation and execution of the project modelling study was used Dia Software [35], referenced in Figure 29, for developing UML diagrams and the Miro online [36], referenced in Figure 30, collaboration platform for graphic collaboration. The project aimed to create a comprehensive model of our system's architecture, focusing on key components and their interactions.

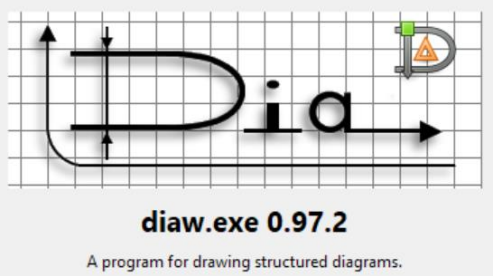

Figure 29 - Dia Diagram Editor v0.97.2

Figure 30 – Miro, online prototype.

5. Development and Implementation

After the modelling phase, the project moves on to the development and implementation phase. Therefore, this chapter 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.

5.1. Digital Architecture of a SC

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 [37].

In this scenario, the digital architecture of a SC process can be summarized in a process of two key stages, as illustrated in Figure 31:

- **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.

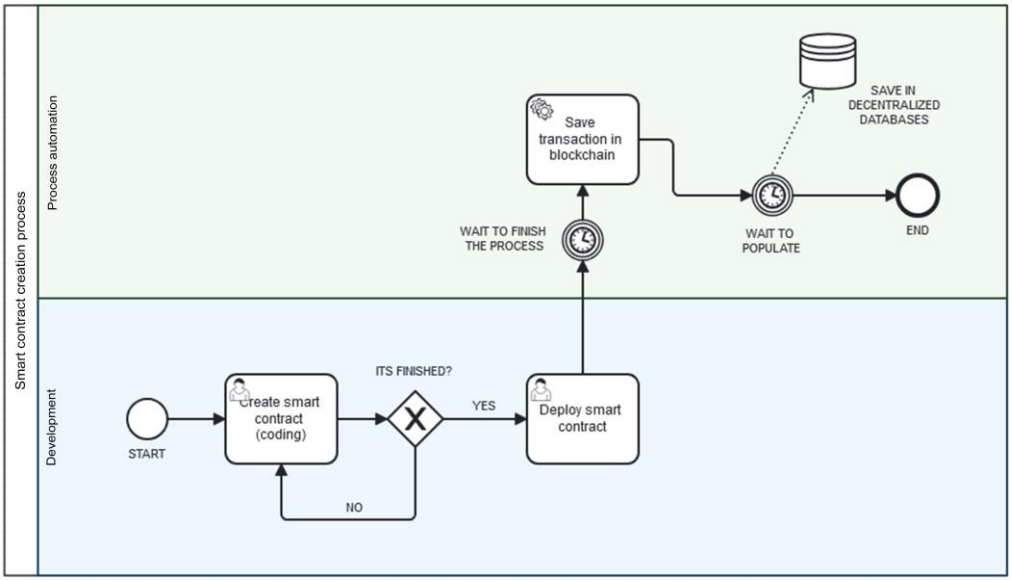

Figure 31 – Digital architecture of SC creation process.

5.2 Tools Used to Development

Several key tools and technologies were used to develop the project, which focused on smart contracts for event ticket sales, several essential tools and technologies were employed. Solidity was the primary programming language used to create secure and immutable smart contracts on the blockchain.

A development framework facilitated the compilation, testing, and deployment of these contracts, while a local blockchain simulator provided a controlled environment for testing.

To enable interaction between the front-end and the blockchain, a JavaScript library was used, ensuring a smooth user experience. A JavaScript framework helped build an interactive user interface, while a decentralized storage solution was implemented for managing ticket data securely.

These tools collectively supported the creation of a user-friendly and secure ticket sales platform based on smart contracts, aiming to innovate the event ticketing industry.

Following, this subsection presents the main the tools employed in the project development across each step of the three main stages: front-end, back-end and Smart contract.

For working with Smart contracts, the following tools and technologies are demonstrated in the Figure 32:

- **Visual Studio Code**: Editor for writing and editing code.
- **Solidity**: Programming language for writing Smart contracts.
- **GitHub** [38]**:** Platform used for repository management, including version control.
- **Truffle suite:** For deploy and unit tests.

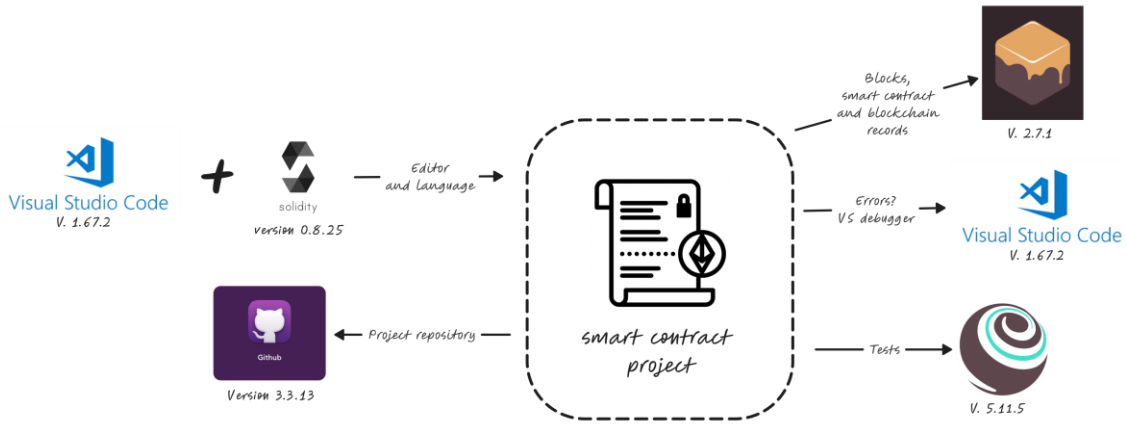

Figure 32 –Tools for Smart contracts development and implementation.

To see the output and identify errors in the Smart contract, the terminal in Visual Studio Code is used. The Truffle framework is employed to test the deployment and to run unit tests. In this case, a unit test was written for the buy function, which aggregates the necessary parameters required to interact with the Smart contract.

The Smart contract functions as a payment bridge between the customer and the system. It is activated once the customer completes the e-ticket purchase process and performs the following steps:

- **Verifies the Ethereum Account:** Confirms the authenticity of the customer's Ethereum account.
- **Checks Gas and ETH Balance:** Ensures that the customer's account has sufficient gas and ETH for the transaction.
- **Validates the Transaction:** Processes and finalizes the payment operation if all conditions are met.

For front-end development, the following tools are demonstrated in the Figure 33:

- **Visual Studio Code**: Editor for writing and editing code.
- **JavaScript with TypeScript** [39] and **Vue** [40]: Programming languages and frameworks.
- **GitHub**: Used for repository management.
- **Microsoft Edge**: Browser for viewing the output.
- **Visual Studio Debugger**: For identifying and fixing errors.

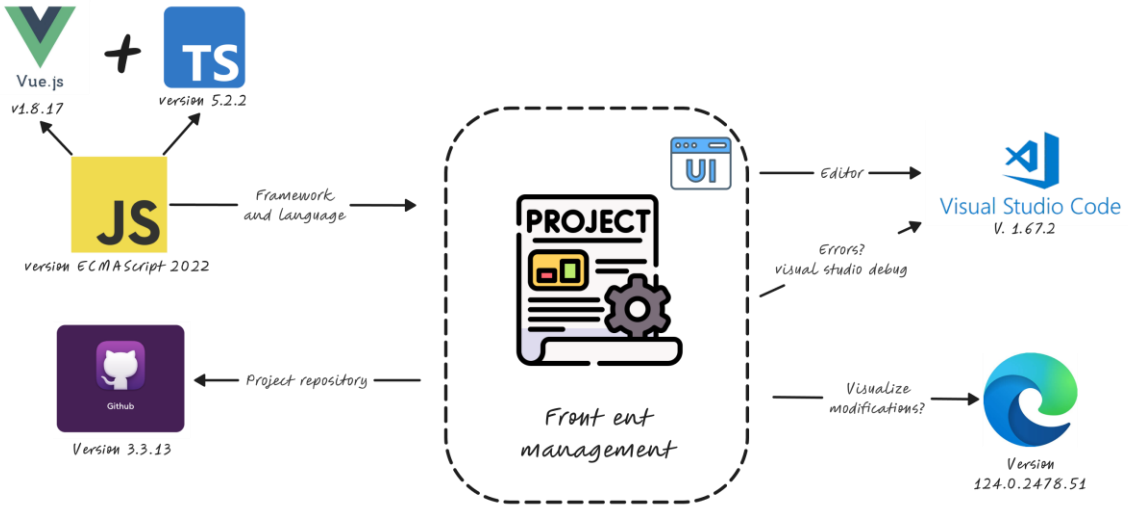

Figure 33 - Tools for front end development and implementation.

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 via Microsoft Edge. This project is hosted on GitHub.

For back-end development tools, the following tools are demonstrated in the Figure 34:

- **MySQL Server:** Used as the database manager, with MySQL Workbench [41] the interface.
- **TypeScript/JavaScript and Vue:** Languages and frameworks used for coding.
- **GitHub:** Used for version control and repository management.
- **Mail trap** [42]**:** Connected to the database for sending email confirmations to customers.

Figure 34 - Tools for back-end development and implementation.

The integration of JavaScript, enhanced with TypeScript and Vue.js, will play a pivotal role in the project's front-end development. This framework will enable seamless interactions between customers and the e-ticket purchasing system. Specifically, it will allow users to select and buy e-tickets through an intuitive interface.

Moreover, the system will facilitate robust communication with the backend database. This includes the ability to retrieve pertinent information, such as ticket availability and pricing, and to transmit user data securely. The use of TypeScript ensures type safety and reduces the likelihood of runtime errors, while Vue.js provides a reactive and component-based architecture that enhances the user experience.

Integration Details: 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.

5.3. Implementation

After the planning tools and system integration plan, starts the implementation and coding rises. Some adjustments are realized in this stage to adapt to the real needs. The implementation phase of the project must be carried out through the following five phases:

1. **HTML development:** webpages, CSS, connections to Vue, add-ons, images, etc. After that, the testing operation must start.
2. **Database development:** Creating the database in workbench with SQL server. After that, the testing operation must start.
3. **HTML + database:** integration and connection for CRUD operations. After that, exhaustive tests must start.
4. **Smart contract development:** creation of a Smart contract and transitions between client side and operations side.
5. **Mail trap:** integration with the application Mail trap to send email to customer with information's but the order.

Next, the delve into each of these phases in detail, much like an artist carefully revealing the layers of their masterpiece in the context of our project.

Phase 1: HTML development

HTML provides the basic structure of a webpage, while CSS enhances its appearance, making it more readable and functional. CSS helps in styling the page, including forms and useful elements like buttons. In this context, HTML often requires NPM [43] to manage and incorporate various packages and app code.

In Figure 35 it's possible to see the welcome webpage supported in HTML structure. The same is replicated to all webpages in the entire project.

NPM includes a CLI that can start a server, in this case with Vue, enabling you to view the webpage graphically. These elements are typically managed and manipulated using an editor like Visual Studio Code.

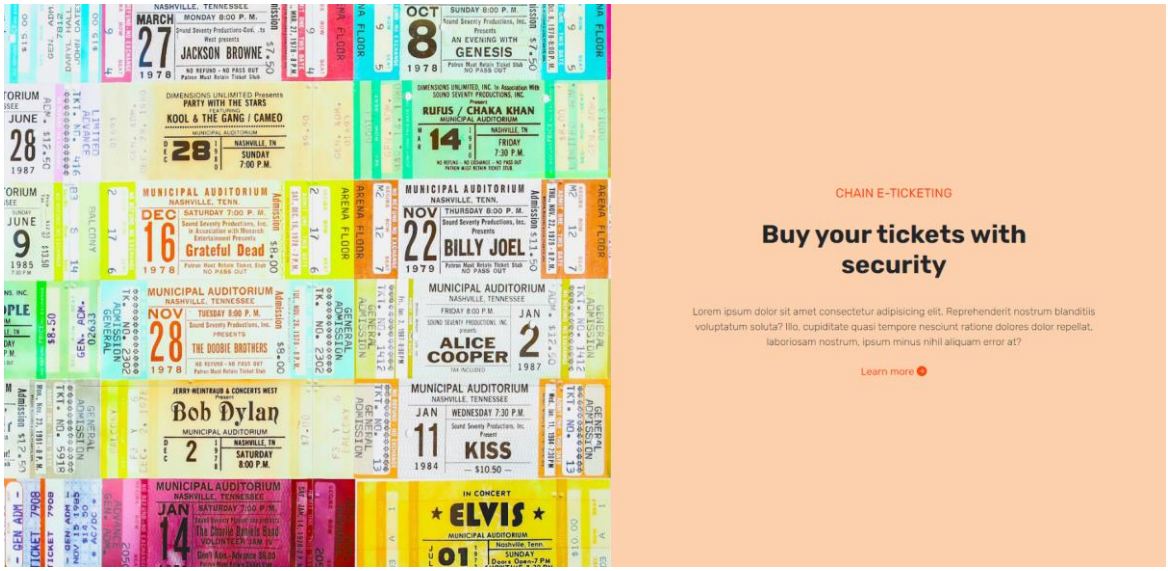

Figure 35 –E-Ticketing system welcome webpage.

Phase 2: Database development

Upon completion of Phase 1, the project will require a database with test data to be evaluated step by step across the web pages. Using the MySQL Workbench interface, a database was created, represented in the Figure 36, including tables and attributes. Subsequently, some of these attributes were populated with information directly within the software.

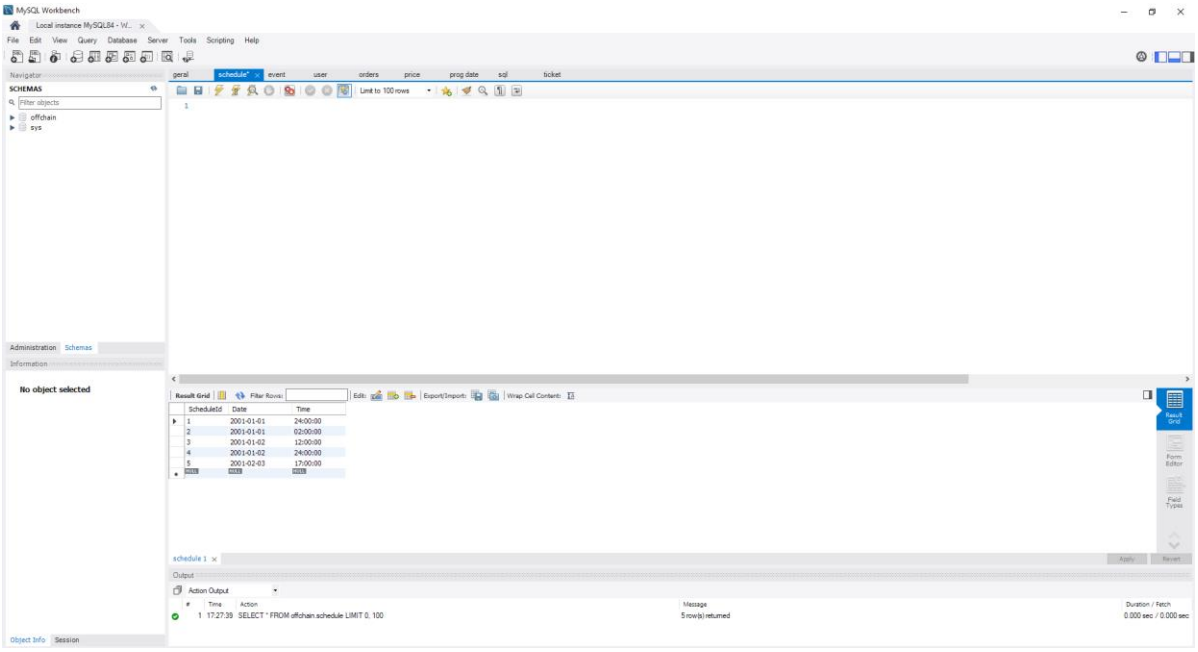

Figure 36 – MySQL Workbench interface environment of the database showing the information in the schedule table.

Phase 3: HTML + Database

In this project, the backend folder and its code files are designed to interface with the database software and support frontend development. To achieve this, Node.js **Error! Reference source not found.** 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, like in Figure 37.

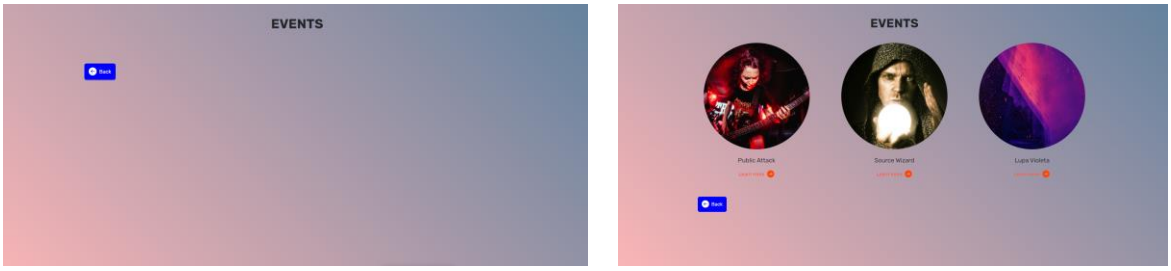

Figure 37 E-Ticketing system webpage without database connection (left) and with database connected (right).

When a customer is on the webpage to purchase a ticket, they will need to fill out several input fields. This information will be recorded in the database. One of these inputs is the Ethereum public key account, which is necessary for validating transactions with Smart contracts.

Phase 4: Smart contract development

In this stage, you need to create a folder within the project to organize all the Smart contract files. Once the code is tested and completed, you must connect the database to this folder to redeem the Ethereum account number and verify and validate the account.

Ganache, Figure 38, is a personal blockchain used for rapid development of Ethereum and File coin distributed applications [45]. It functions like an EVM and provides Ethereum public and private key accounts, along with gas and simulated Ethereum balances. Ganache allows you to test transactions and Smart contracts in a local environment.

Unfortunately, testing must be conducted within a local ecosystem or off-chain blockchain storage [46] as performing these tests on a real Ethereum network or on-chain blockchain storage [46] would be prohibitively expensive due to the high cost of Ethereum transactions.

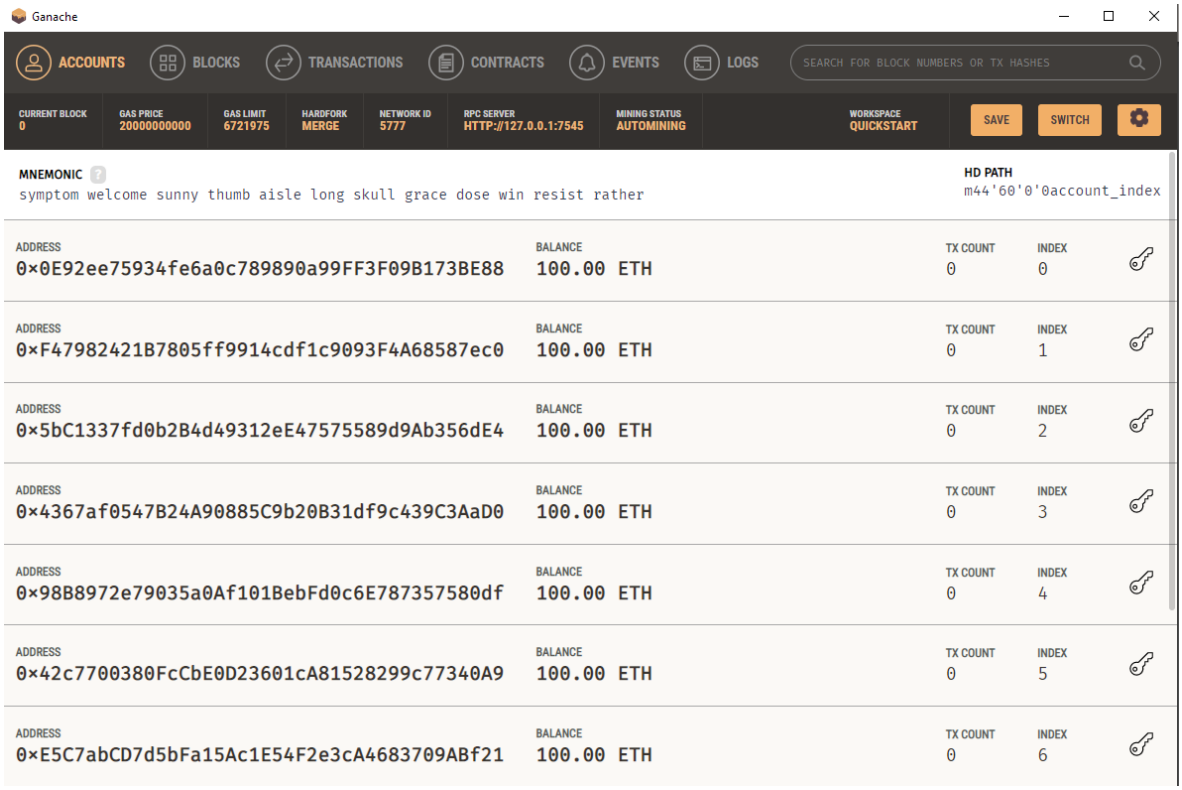

Figure 38 - Ganache application.

The verification phase analyses whether the public key account exists. If it does, the system then verifies the private key account number. If the private key matches, the system checks whether the customer has sufficient funds to cover the gas cost and the ticket purchase. This involves calculating the SHA3 hash of the input data and converting Ether values to Wei to ensure the Ethereum balance is adequate.

Note: Ganache creates a single wallet with multiple accounts, so the security phrases required and requested to validate a transaction will not work at this time. This is because the environment is designed to be developer-friendly, simulating the blockchain without the usual security barriers.

To explain the gas in Ganache, gas it is a unit of measurement for computational work. In Ethereum, every operation or computation that occurs on the blockchain requires a certain amount of gas. This includes executing Smart contracts, making transactions, and more. Gas is used to measure and allocate resources for operations and ensures that no infinite loops or excessive computations can occur [47].

The Alchemy website have a good explanation what gas is [48]:

> *"The purpose of gas is to act as a fee for computing the operations of a* Smart contract *done by each Ethereum node. There needs to be a fee for computation in order to prevent an attacker from bringing the network to a halt by deploying a large amount of complex contracts that require long computation times. This type of DDoDS attack is discouraged because it would be so expensive to run."*

Once the Smart contract is created, it needs to be deployed. To facilitate this, an ABI must be defined. The ABI enables the Smart contract to communicate and interact with external applications and other Smart contracts. Unlike a typical API in web development, the ABI does not allow for direct JSON requests to a Smart contract. Instead, communication is conducted in bytecode, which is translated into a format understandable by the EVM through ABI encoding.

The ABI encoding specifies methods and data types in a JSON-RPC file, detailing function signatures and variable declarations. This encoding ensures that the EVM can accurately execute the correct function within the Smart contract.

When the Smart contract is deployed, the associated transaction is recorded in a block on the blockchain. This blockchain exists within a decentralized network. In the validation phase, after the verification phase is completed, the Smart contract is created, and the operation is logged in Ganache transactions, which are part of the blockchain.

Each block in the blockchain contains transaction information. To ensure accuracy and prevent tampering, this information is replicated across multiple machines within the decentralized network, creating a chain of blocks. This replication process helps maintain the integrity and immutability of the blockchain data, like David Radeck et al says in "What Is Blockchain?" [49]:

> *"Theoretically, a decentralized network, like blockchain, makes it nearly impossible for someone to make fraudulent transactions… blockchain systems use proof-of-stake or proof-of-work transaction verification methods that make it difficult, as well as not in participants' best interests, to add fraudulent transactions."*

Phase 5: Mailtrap

Mailtrap [42] is a platform which provides a fake SMTP server in the development system to test, Figure 39, view and share emails sent from the pre-production environments and test with real data.

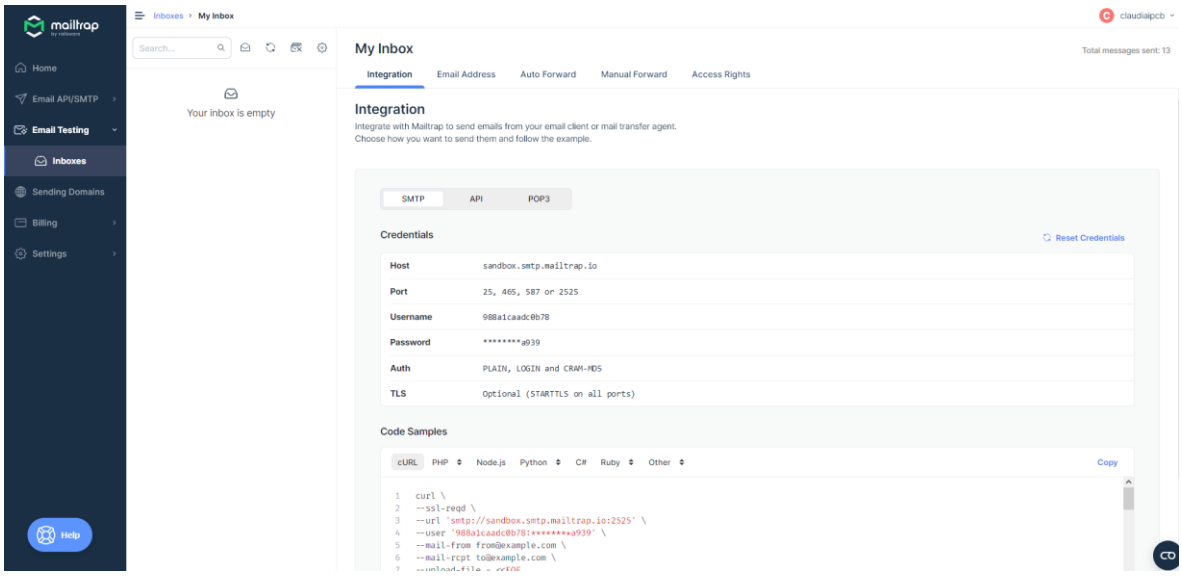

Figure 39 - Mailtrap environment.

This application needs to be implemented in the project to have a connection, after that it's possible to receive custom emails.

5.4. Lifecycle of E-ticket

An e-ticket undergoes a lifecycle from its creation to its destruction, which involves complex processes and different systems. While the lifecycle of the e-ticket was studied and demonstrated in Figure 40, the project did not include the development of an application to read, validate, and deactivate the e-ticket.

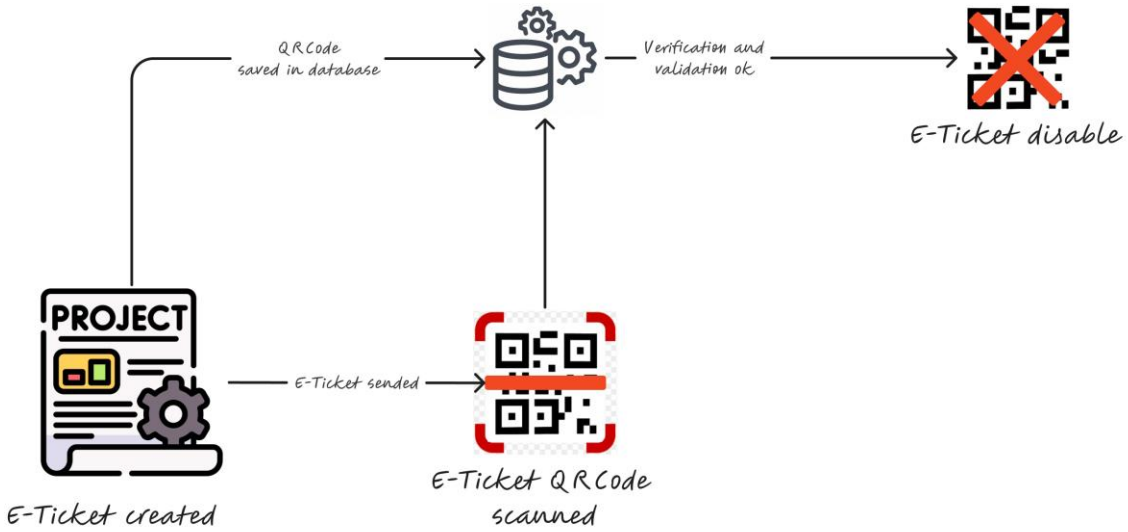

Figure 40 - Lifecycle of qrcode.

Once the e-ticket is created, it is coded, recorded in a centralized database, and sent to the user's email. To validate and deactivate the e-ticket, the user simply needs to present the QR code.

5.5. Chapter Conclusion

Blockchain is a decentralized network where each block contains transaction information. To ensure accuracy and prevent tampering, this information is replicated across multiple machines, creating a chain of blocks. This replication process helps maintain the integrity and immutability of the blockchain data.

Blockchain systems use proof-of-stake or proof-of-work transaction verification methods that make it difficult and not in participants' best interests to add fraudulent transactions.

The development of a blockchain-based smart contracts e-ticketing platform involves several key stages. First, the front-end development uses JavaScript with TypeScript and the Vue.js framework to create an intuitive user interface and enable seamless interactions between customers and the e-ticket purchasing system.

The back-end development employs a MySQL database to store customer information, including Ethereum public accounts submitted during the purchase process. The smart contract development is done using Solidity, a programming language for writing secure and immutable smart contracts on the blockchain.

A local blockchain simulator, Ganache, is used to test the smart contract deployment and interactions, as testing on a real Ethereum network would be prohibitively expensive due to the high cost of Ethereum transactions.

The smart contract functions as a payment bridge between the customer and the system, verifying the Ethereum account, checking gas and ETH balance, and validating the transaction if all conditions are met.

Finally, the project integrates a Mailtrap platform, which provides an alternative SMTP server in the development system to test, view, and share emails sent from the pre-production environments and test with real data. The e-ticket undergoes a lifecycle from its creation to its destruction, involving complex processes and different systems, although the project did not include the development of an application to read, validate, and deactivate the e-ticket.

6. Integrations Results

In this chapter will be demonstrated the results after implementation and testing. For better understanding, we will follow the actions of the created user with ID 206.

This chapter presents the organization of the project, beginning with the local database, in chapter 6.1, that outlines the structure of the data used, in chapter 6.2 followed by the email platform utilized for receiving E-tickets, and then explores the virtual machine (VM) employed to create smart contracts in chapter 6.3, detailing the deployment process of these contracts along with a demonstration, and proceeding to the testing, in chapter 6.5, phase of the created smart contracts, concluding with a discussion on the security aspects related to smart contracts, in chapter 6.6.

6.1. Local Database

In the examples provided, you can see both the randomly generated ticket number and the QR code image on the final page (thank you page). For simplicity and demonstration purposes, since 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 not the same in the database: one represents the user ID, and the other represents the order ID.

In the local database, the user and orders are created, like demonstrated in Figure 41 and Figure 42:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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	0x272534473BB74EA9314afF2D2719Ba43886BE199

Figure 41 – User's data observed in MySQL workbench.

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

OrderId	TicketNum	QRCode	EventId	UserId	ScheduleId
206	5403372003600937e4trbro651i	...	1	206	3
205	1530858396757886quylo7g5pl	...	3	205	3
204	88580241345834 86wrl8e29iz3e	...	2	204	5
203	2492744890339063im4ltuqztp	...	2	203	1
202	2208633571423427102iphal0aq	...	1	202	3

Figure 42 – Orders data observed in MySQL workbench.

6.2. Email

After creating the database, the system can retrieve the necessary data to send to the user via email, demonstrated in Figure 43 and Figure 44. It selects the recipient's email address and extracts the relevant elements from the e-ticket. The HTML content is then generated on the back end and sent to Mailtrap through the API.

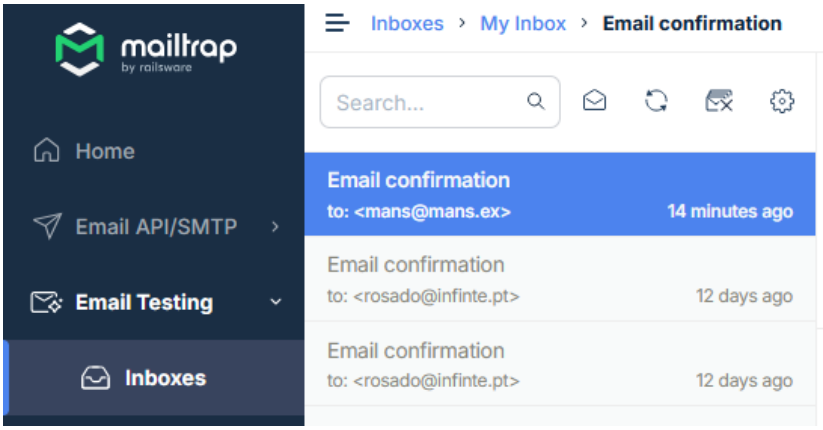

Figure 43 – Mailtrap inbox.

Figure 44 – Illustration of the HTML generated sent to Mailtrap.

The e-ticket information is integrated into the local database, with the relevant tables being:

- User (to push email, Ethereum public account).
- Orders (to push ticket number, where presents a picture generated to the function in the project).
- Event (to push the prefix, event image, local, event name).
- Schedule (to push the date and time).
- Event_schedule (to push the price).

6.3. Smart Contract

The generation of the SC is more complex. We will be tracking the block 36, which is related to user ID 206, for a better understanding. First, create the transaction for the e-ticket in the block:

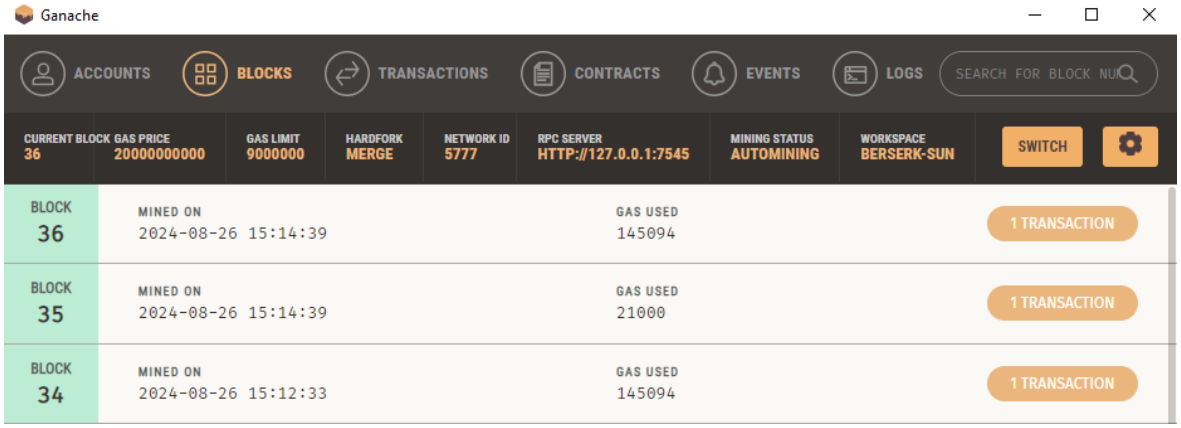

Figure 45 - Illustration of a mined block in Ganache.

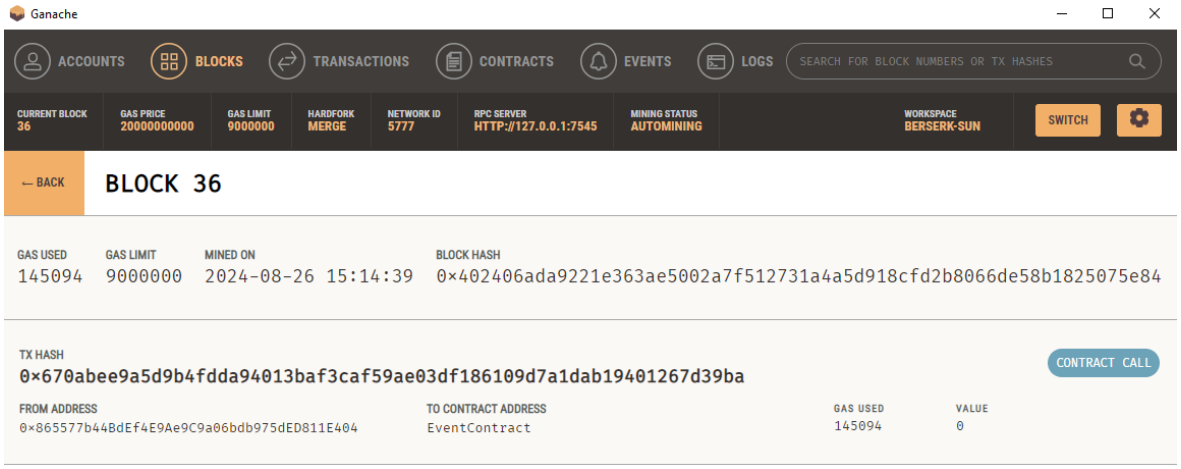

Figure 46 - Scope and details of the block 36.

By clicking on the block, you can view more detailed information about it. 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.

6.4. Contract Deployment

After testing transactions and mining blocks, the Smart contract needs to be deployed. In this project, deployment is performed manually via the Visual Studio Code terminal with Truffle library. Once deployed, Figure 47, Ganache will display the following information:

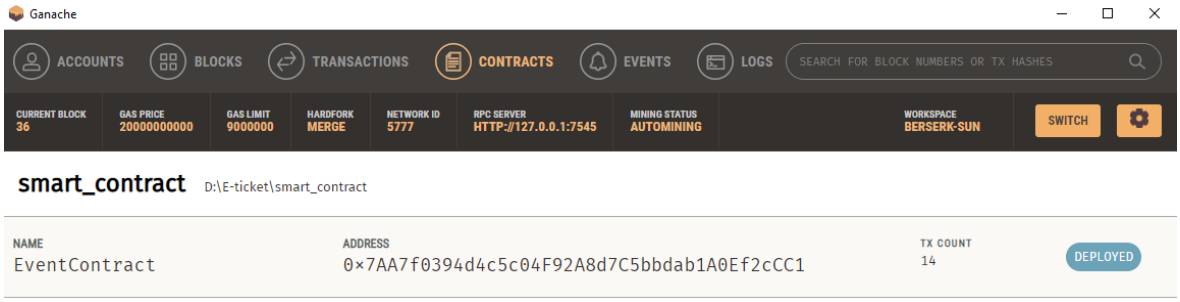

Figure 47 - Illustration of the deployed contract in Ganache.

Ganache displays the Smart contract's address, which you need to copy and paste into your project code 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, Figure 48, Ganache provides detailed information such as storage, transactions, and events:

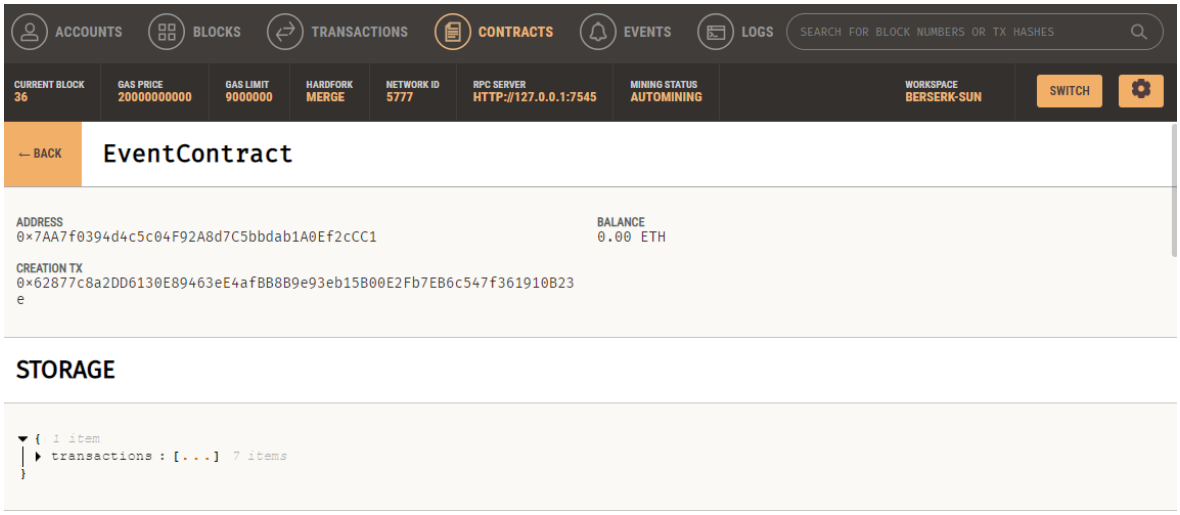

Figure 48 - Illustration of the smart contract details in Ganache.

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.

Allowed in storage:

- Persistent data that a contract can store on the Ethereum blockchain.
- Each Smart contract has its own storage, demonstrated in Figure 49, space where it can save variables and state information.
- When a Smart contract deployed, its initial state is stored in the contract's storage.

Allowed in transactions:

- Transactions represent actions performed on the Ethereum network.
- When a Smart contract interacts (e.g., calling a function or deploying a new contract), is created a transaction.
- Transactions include details like the sender's address, the contract being interacted with, the function called, and any input data.
- Transactions are recorded on the blockchain and are irreversible once confirmed.

Allowed in events:

- Events are a way for Smart contracts to communicate with external applications or other contracts.
- When certain conditions are met within a contract (e.g., a payment received or an order fulfilled), an event is emitted.
- Events provide a log of important occurrences within the contract.
- External applications can listen for these events and react accordingly.

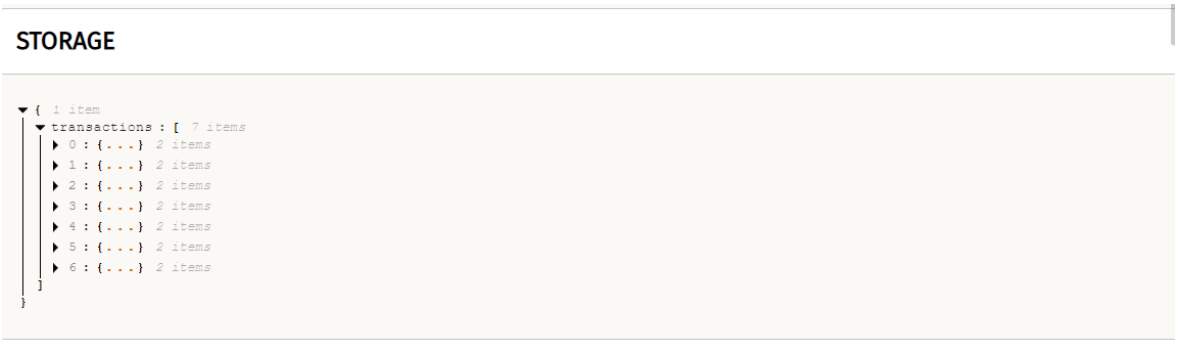

Figure 49 - Illustration of the inside storage for a smart contract.

Inside storage, is 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 50.

Figure 50 – Illustration of the transactions and events in a smart contract.

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 showed in Figure 51.

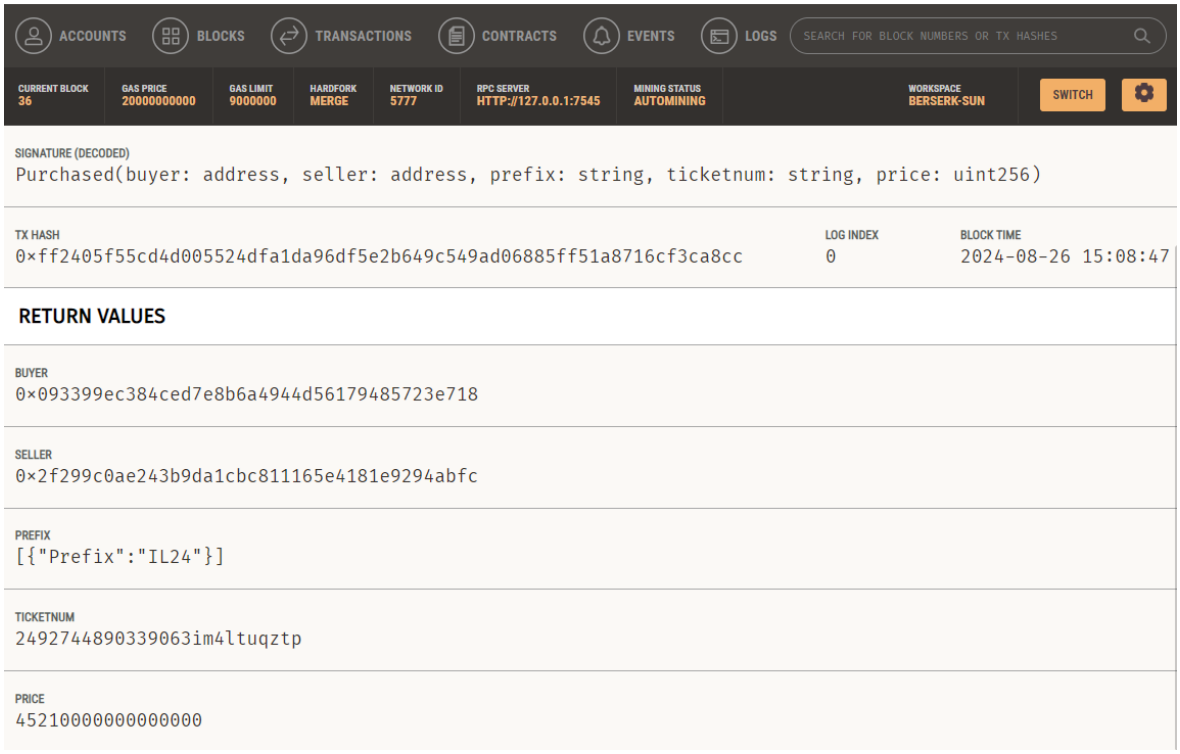

Figure 51 – Illustration of the event details for the deployed smart contract.

6.5. Testing

To do tests in the SC, the Truffle framework [50] 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 the Truffle. These tests help ensure the Smart contracts behave as expected before deploying them to a live blockchain, in example:

- **Compiles Contracts:** Before running the tests, Truffle ensures that the Smart contracts are compiled. This compilation step generates the necessary artifacts (e.g., ABI and bytecode) required to interact with the contracts during testing.
- **Deploys Contracts:** Truffle deploys the SC to a local in-memory Ethereum blockchain (provided by Ganache or similar) or a specified test network. This blockchain environment is isolated from the live network, allowing safe testing and development.
- **Runs Test Scripts:** Truffle executes test scripts written in JavaScript (or TypeScript) using Mocha and Chai libraries. These scripts contain test cases that interact with the deployed Smart contracts to verify their behaviour.
- **Provides Test Results:** After executing the test scripts, Truffle outputs the results, showing which tests passed or failed. This feedback helps developers identify issues in the contracts' logic, functionality, or deployment scripts.

The necessity to conduct tests on smart contracts imposes significant constraints on both the scope and the range of possibilities. This limitation is primarily attributed to the lack of diversity in testing capabilities provided by the Truffle framework [51]. Consequently, the testing process is restricted, which may impact the comprehensiveness and robustness of the smart contract evaluations.

In the present case, shown in Figure 52, 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.

These passing results in Ganache, Figure 53, 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.

```
> Compiled successfully using:
   - solc: 0.8.0+commit.c7dfd78e.Emscripten.clang


  Contract: EventContract
    ✓ should deploy correctly
    ✓ should perform the buy function correctly (60ms)


  2 passing (125ms)
```

Figure 52 – Illustration of unit test in smart contract.

Figure 53 – Illustration of the test result in Ganache.

The test suite for *EventContract*, Figure 54, shows that the contract deploys correctly, but the buy function test fails due to an invalid address error, so the deploy 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 being 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*.

Figure 54 - Illustration of a failure result.

Based in test results in Figure 55, 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."

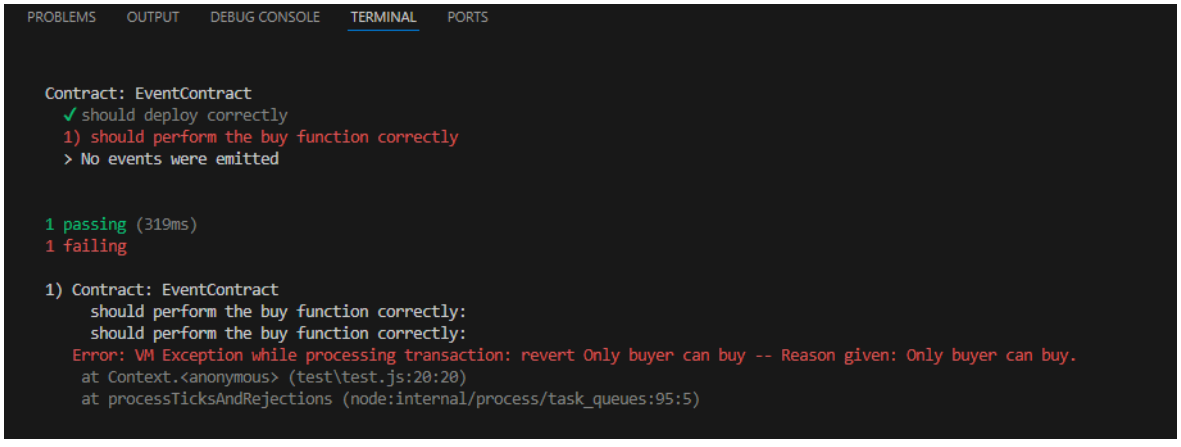

Figure 55 - Illustration of a buyer failure.

6.6. Security

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 SC offer 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 [52].

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

By separating user credentials from financial transactions, blockchain can enhance user privacy and reduce the risk of data leaks [52]. Which cryptocurrency wallets have a system protection, like multi-phase or multi-signature security features. For example:

- **Ledger Nano S/X:** These hardware wallets use a 24-word recovery phrase. If you lose this phrase, you cannot recover your wallet. They also support multi-signature transactions, which require multiple approvals [53];
- **Trezor:** Another popular hardware wallet that uses a 24-word recovery seed. It also supports multi-signature transactions for added security [54];
- **MetaMask:** This software wallet uses a 12-word recovery phrase. While it doesn't use a multi-phase system for transactions, it emphasizes the importance of securely storing the recovery phrase [55].

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) 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.

6.7. Chapter Conclusion

In summary, the risk of losing the 12-phase code or recovery phrase is a significant concern for cryptocurrency wallets, as there is no recovery option if users permanently lose access to their funds, this can be a major drawback compared to wallets that offer recovery options.

However, the multi-phase or multi-signature security features provided by wallets like Ledger Nano S/X and Trezor can enhance user privacy and reduce the risk of data leaks.

The complexity of these security systems may deter some less tech-savvy users from adopting them. Smart contracts have their own advantages and disadvantages, with the testing process being a crucial aspect to ensure their functionality and security.

Ultimately, the choice of cryptocurrency wallet and the use of smart contracts involve balancing security, user-friendliness, and the specific needs of the user or application.

7. Conclusion

This chapter concludes the dissertation by highlighting key areas for enhancement and identifying specific directions for future work. It underscores essential aspects that can further refine and expand upon the findings presented, paving the way for continued development and exploration in this field.

7.1. Achievements

The development of SCs on blockchain technology represents a revolutionary shift in how we approach digital transactions and agreements. Through the integration of blockchain's inherent decentralization, transparency, and security, Smart contracts have the potential to eliminate intermediaries and build trust across a variety of sectors, particularly in ticketing, where fraud and inefficiencies remain significant issues.

While the project outlined in this document is ambitious, it also highlights the challenges associated with implementing SCs, such as excessive costs, energy consumption, and technical complexity. However, the benefits, particularly in terms of scalability, security, and user autonomy, make this endeavour a worthwhile pursuit.

The project detailed in this document centres on the creation of an innovative e-ticketing platform powered by blockchain technology, leveraging smart contracts to ensure secure, transparent, and tamper-proof ticketing transactions.

By utilizing blockchain’s decentralized nature, this platform aims to eliminate fraud, streamline ticket verification, and provide a seamless experience for both organizers and attendees. The smart contract architecture automates key processes, from ticket issuance to resale, making the entire e-ticketing process more efficient, trustworthy, and accessible.

It begins with the recognition of the challenges and inefficiencies in traditional event ticketing systems, such as fraud, ticket duplication, and the need for manual verification processes. To address these issues, the project aims to leverage the capabilities of Smart contracts and blockchain to create a more secure, efficient, and user-friendly e-ticketing solution.

The project unfolds in several phases, starting with a comprehensive review of the systemic review technologies and methodologies related to Smart contracts, blockchain, and e-ticketing. This phase involved identifying the key features and benefits of Smart contracts, as well as the potential applications of blockchain in the context of event ticketing.

Following the review phase, the project transitioned into the system design stage, where the architecture, user interface, and database model for the e-ticketing platform were outlined. This involved creating sequence diagrams, class diagrams, and entity-relationship models to visualize the system's structure and functionality.

The subsequent phase focused on the development and implementation of the Smart contract-based e-ticketing platform. Technologies such as JavaScript, Solidity, Truffle, and Ganache were used to create, test, and deploy the Smart contracts on a blockchain local network.

This project concludes with this summary on its achievements, challenges, and future directions. It recognizes the potential of SCs and blockchain technology in revolutionizing event ticketing systems and emphasizes the need for further improvements and real-world testing to validate the platform's performance and security.

As the Internet transitions to Web 3.0, SCs will play an integral role in reshaping how we handle digital identity, property, and finance, giving users greater control over their data and interactions. By exploring the potential of SCs for event ticketing, this project not only seeks to solve specific problems within that sector but also contributes to the broader development of decentralized systems in the digital economy.

7.2. Future Work

Future work should prioritize code improvements, particularly by isolating the script component into separate files for better security. Testing the Smart contracts in real-world scenarios will also provide insights into performance and help uncover potential security issues.

This project has established a foundation for blockchain-based event ticketing, but there’s room for growth. Key areas include scalability and performance optimization, especially given the limitations of Ethereum in terms of transaction throughput. Exploring other platforms like Polygon, Solana, or Avalanche could improve scalability and reduce costs.

While Smart contracts enhance security, additional fraud prevention measures, such as biometric authentication and multi-factor authorization, could be integrated. Expanding beyond core ticketing functions, the system could add secondary marketplaces, dynamic pricing, loyalty programs, and event management integration.

Improving the user interface would make the platform more accessible, especially by simplifying wallet management. Integrating Web3 logins would allow users to reuse their Ethereum accounts, avoiding duplicate entries in the database and simplifying account management.

As the regulatory landscape evolves, it’s crucial to ensure compliance with digital asset, data privacy, and consumer protection laws. Engaging with blockchain and ticketing communities can provide valuable feedback, reveal new use cases, and drive the project forward. By addressing these areas, this project can become a more robust, scalable, and user-friendly blockchain-based ticketing solution, enhancing transparency, security, and efficiency in event ticketing.

7.3. Contributions

1. Cláudia Silva, Mónica Costa, Alexandre Fonte. Blockchain-based Smart Contracts Platform for Transparent and Efficient E-Ticketing. International Journal of Computer Applications (IJCA). 186, 59 (Jan 2025), 21-26. DOI=10.5120/ijca2024924335SN.

2. Cláudia Silva, Mónica Costa, Alexandre Fonte. Blockchain-based Smart Contracts E-ticketing — a Systematic Review. Springer Nature Computer Science (SN Computer Science).

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] "Smart contract docs", https://ethereum.org/en/developers/docs/smart-contracts/, last consulting in august of 2024.

[4] Catchlove, Paul. "Smart contracts: a new era of contract use." Available at SSRN 3090226 (2017).

[5] "Solidity Programming Language", https://soliditylang.org/, last consulting in March of 2024.

[6] Shivalingaiah, D., and Umesha Naik. "Comparative study of web 1.0, web 2.0 and web 3.0." (2008).

[7] Rainer Böhme, Rainer. Christin, Nicolas. Edelman, Edelman. Moore, Tyler. (2015) BITCOIN: ECONOMICS, TECHNOLOGY, AND GOVERNANCE, JOURNAL OF ECONOMIC PERSPECTIVES—VOLUME 29, NUMBER 2—SPRING 2015—PAGES 213–238, HTTP://DX.DOI.ORG/10.1257/JEP.29.2.213 DOI=10.1257/JEP.29.2.213.

[8] Johnson, Jackie, Is Cardano a Serious Rival to Ethereum? (July 14, 2021). Available at SSRN: https://ssrn.com/abstract=3886108 or http://dx.doi.org/10.2139/ssrn.3886108.

[9] Jiameng Liu, Shaoliang Peng, Chengnian Long, Lijun Wei, Yunhao Liu, and Zhihui Tian. 2020. Blockchain for Data Science. In Proceedings of the 2020 The 2nd International Conference on Blockchain Technology (ICBCT'20). Association for Computing Machinery, New York, NY, USA, 24–28. https://doi.org/10.1145/3390566.3391681.

[10] Felipe Rodrigues Oliveira, Ronaldo Colucci, and Liriane Soares de Araújo. "Um estudo sobre a web 3.0: evolução, conceitos, princípios, benefícios e impactos." Revista Interface Tecnologica 15.2 (2018): 60-71.

[11] Yang Liu, Yuwen Zhang, Siyu Zhu, and Cheng Chi. 2020. A Comparative Study of Blockchain-Based DNS Design. In Proceedings of the 2019 2nd International Conference on Blockchain Technology and Applications (ICBTA '19). Association for Computing Machinery, New York, NY, USA, 86–92. https://doi.org/10.1145/3376044.3376057.

[12] Wensheng Gan, Zhenqiang Ye, Shicheng Wan, and Philip S. Yu. 2023. Web 3.0: The Future of Internet. In Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), 1268. Association for Computing Machinery, New York, NY, USA, 1266–1275. https://doi.org/10.1145/3543873.3587583.

[13] Jiameng Liu, Shaoliang Peng, Chengnian Long, Lijun Wei, Yunhao Liu, and Zhihui Tian. 2020. Blockchain for Data Science. In Proceedings of the 2020 The 2nd International Conference on Blockchain Technology (ICBCT'20). Association for Computing Machinery, New York, NY, USA, 24–28. https://doi.org/10.1145/3390566.3391681.

[14] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.

[15] Szabo, N. (1994). Smart contracts. https://web.archive.org/web/20140413000357/http://szabo.best.vwh.net/smart.contracts.html, last consulting in March of 2024.

[16] Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. IEEE International Congress on Big Data.

[17] Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A Survey of Attacks on Ethereum Smart contracts (SoK). International Conference on Principles of Security and Trust.

[18] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., ... & Wattenhofer, R. (2016). On Scaling Decentralized Blockchains. International Conference on Financial Cryptography and Data Security.

[19] L. Liu, S. Zhou, H. Huang and Z. Zheng, "From Technology to Society: An Overview of Blockchain-Based DAO," in IEEE Open Journal of the Computer Society, vol. 2, pp. 204-215, 2021, doi: 10.1109/OJCS.2021.3072661.

[20] 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, last consulting in November of 2023.

[21] K. -P. Lin, Y. -W. Chang, Z. -H. Wei, C. -Y. Shen and M. -Y. Chang, "A Smart Contract-Based Mobile Ticketing System with Multi-Signature and Blockchain," 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE), Osaka, Japan, 2019, pp. 231-232, doi: 10.1109/GCCE46687.2019.9015425.

[22] Abdelhamid, Manar. Hassan, Ghada. 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.

[23] 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.

[24] R. K. Kaushal, N. Kumar, S. N. Panda and V. Kukreja, "Immutable Smart Contracts on Blockchain Technology: Its Benefits and Barriers," 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2021, pp. 1-5, doi: 10.1109/ICRITO51393.2021.9596538.

[25] M. Muneeb, Z. Raza, I. U. Haq and O. Shafiq, "SmartCon: A Blockchain-Based Framework for Smart Contracts and Transaction Management," in IEEE Access, vol. 10, pp. 23687-23699, 2022, doi: 10.1109/ACCESS.2021.3135562.

[26] K. Saini, A. Roy, P. R. Chelliah and T. Patel, "Blockchain 2.0: A Smart Contract," 2021 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 2021, pp. 524-528, doi: 10.1109/ComPE53109.2021.9752021.

[27] 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.

[28] H. -Q. Nguyen, H. T. Nguyen and T. -T. Pham, "Applying Smart Contract in Blockchain Technology to Manage the Ticketing Issuance and Ticketing Traceability," 2023 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), Amman, Jordan, 2023, pp. 160-164, doi: 10.1109/JEEIT58638.2023.10185771.

[29] Sayal, C. Vasundhara, V. Gupta, A. Gupta, H. Maheshawri and M. Memoria, "Smart Contracts and Blockchain: An Analytical Approach," 2023 6th International Conference on Contemporary Computing and Informatics (IC3I), Gautam Buddha Nagar, India, 2023, pp. 1139-1142, doi: 10.1109/IC3I59117.2023.10397748.

[30] "Truffle Docs", https://archive.trufflesuite.com/docs/, last consulting in March of 2024.

[31] "What is Hardhat", https://hardhat.org/, last consulting in March of 2024.

[32] "Visual Studio Page", https://code.visualstudio.com/, last consulting in august of 2024.

[33] D'Andrade, R. and Egan, M. (1974), the colors of emotion1. American Ethnologist, 1: 49-63. https://doi.org/10.1525/ae.1974.1.1.02a00030.

[34] "Google Rubik Font", https://fonts.google.com/specimen/Rubik, last consulting in March of 2024.

[35] "Dia Software", https://sourceforge.net/projects/dia-installer/, last consulting in December of 2023.

[36] "Miro platform", https://miro.com/, last consulting in June of 2024.

[37] B. Ashok, S. Denis Ashok, C. Ramesh Kumar. A review on control system architecture of a SI engine management system, Annual Reviews in Control, Volume 41, 2016, Pages 94-118, ISSN 1367-5788, https://doi.org/10.1016/j.arcontrol.2016.04.005, last consulting in April of 2024.

[38] "Github plaftorm", https://github.com/, last consulting in august of 2024.

[39] "TypeScript page", https://www.typescriptlang.org/, last consulting in august of 2024.

[40] "Vue page", https://vuejs.org/, last consulting in august of 2024.

[41] "Workbench page", https://www.mysql.com/products/workbench/, last consulting in august of 2024.

[42] "Mailtrap Platform", https://mailtrap.io/, last consulting in June of 2024.

[43] "About NPM", https://docs.npmjs.com/about-npm, last consulting in May of 2024.

[44] "NodeJS", https://nodejs.org/en, last consulting in May of 2024.

[45] "Ganache Docs", https://archive.trufflesuite.com/docs/Ganache/, last consulting in May of 2024.

[46] Reijers, Wessel, et al. "Now the code runs itself: On-chain and off-chain governance of blockchain technologies." Topoi 40 (2021): 821-831, last consulting in June of 2024.

[47] Werner, S.M., Pritz, P.J., Perez, D. (2020). Step on the Gas? A Better Approach for Recommending the Ethereum Gas Price. In: Pardalos, P., Kotsireas, I., Guo, Y., Knottenbelt, W. (eds) Mathematical Research for Blockchain Economy. Springer Proceedings in Business and Economics. Springer, Cham. https://doi.org/10.1007/978-3-030-53356-4_10.

[48] "Alchemy Platform", https://www.alchemy.com/, last consulting in June of 2024.

[49] "Rodeck, David. Curry, Benjamin. "What is blockchain." Forbes Media, Apr 28 (2022).

[50] "Truffle Documentation", https://archive.trufflesuite.com/docs/truffle/, last consulting in august of 2024.

[51] "Truffle tests documentation", https://archive.trufflesuite.com/docs/truffle/how-to/debug-test/test-your-contracts/, last consulting in septembre of 2024.

[52] "Open Access publisher empowering researchers", https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0284166, last consulting in august of 2024.

[53] "Ledger Wallets", https://www.ledger.com/, last consulting in august of 2024.

[54] "Crypto Security trezor", https://trezor.io/, last consulting in august of 2024.

[55] "Metamask Wallets", https://metamask.io/, last consulting in august of 2024.