Massimo Ciaffoni
Denil Nicolosi
Michele Pasqualini
Francesco Zerbino Di Bernardo

Blockchain-based Ticketing

Software Cybersecurity Project
Ingegneria Informatica e dell'Automazione
Docente: Spalazzi Luca
Novembre 2021

Contents

List of figures

1. Introduction

This project was developed as part of the Software Cybersecurity course, in particular it involves the design and implementation of an IT system dedicated to the sale of tickets for participation in various events. The system is based on blockchain technology, which in recent years is gaining more and more importance. Blockchain is an alternative or integral part to Database or Claud Storage. At the base of the blockchain there are the Smart Contracts that represent the backend and allow the creation of Decentralized Applications (DAPPs). In this report an overview of the project will be made, in particular we will describe the Blockchain technology in detail and how it operates, the software, the languages and the libraries used for the implementation, the requirement analysis guided by the risk analysis, the smart contracts and a user guide.

1.1. Problem area

The management of a blockchain-based ticketing system can simplify the validation process and guarantee the security of users when purchasing a ticket while trying to prevent fraud. The project is commonly developed with a view to the Software Cybersecurity course but trying to understand and model the real scenarios that can occur within the ticketing context.

1.2. Project description

The project was to develop a prototype ticketing system based on blockchain technology. We developed a web application which contains different interfaces, one interface for each module of the system: one interface for the event manager, one interface for the ticket buyer and one interface for the ticket validator. The ticket reseller was considered like a module part of event manager, this because we tried to make more simply the system. We started with the analysis of the reference context by identifying the main figures who act within a system. Therefore, the assets were initially identified, we gave an evaluation both in terms of the asset itself and in terms of impact, that is the impact that would occur if the security policies of a certain asset were violated. Subsequently, all possible threats were identified, and an assessment of all possible attacks was made. Finally, we have identified some control measures together with an evaluation on the feasibility of the latter in order to define specifications for our system. All these project documents will be reported in the dedicated sections below. In the design phase, the technology we were told to use by the professor was the blockchain. Below is a paragraph explaining what the blockchain is and how it works.

2. Blockchain

2.1. What is Blockchain?

In the simplest terms, blockchain is a growing list of records called blocks that get generated and linked using cryptographic hash functions [3]. Each new block gets generated by hashing the current newest block in the chain and adding that hash as a header in the proposed newest block. For this new block and its changes to be approved, a mathematical puzzle involving the hash, has to be solved. Across the network, so-called "miners" attempt to solve this puzzle. When a miner eventually comes up with a solution all the nodes on the network has to check and confirm the change before updating the blockchain [4].

The first blockchain was created in 1991, a Merkle Tree was used to create a “Secured chain of blocks”, where each block contains data, a timestamp and a cryptographic hash of the previous block. This is the “chain” in Blockchain [5]. A Merkle tree is a data structure where each non-leafe node is a hash of its child nodes, where the leaf nodes are the lowest tier of nodes [6].

Late in 2008, an anonymous programmer or a group called Satoshi conceptualized the design of the first blockchain [5]. Then in early 2009, Satoshi created and published the distributed blockchain to be used as a public transaction ledger for a cryptocurrency called Bitcoin [7]. This blockchain would contain a secure history of data exchanges, timestamps and each exchange is verified by utilizing a peer-to-peer network. All of this would be done without a central figure of authority, autonomously. Later that same year, Satoshi Nakamoto released the open-source, decentralized cryptocurrency, Bitcoin, the first cryptocurrency [7]. Since the release of Bitcoin, there have been many so-called Altcoins, different cryptocurrencies with many different names, but Bitcoin is still the most valuable and popular. A cryptocurrency is a digital asset or virtual currency designed to be used for exchanges in a blockchain network [7]. Cryptography is used to secure and verify transactions and control the creation of new units of a cryptocurrency.

Cryptography in blockchain technology is used for transactions, wallets, privacy and security [8]. Blockchain uses a Public-key cryptography system. This system uses a pair of keys for interactions between users in the chain, a public key, and a private key. For example, if Bob wants to send Alice a message, he would need Alice's public key to encrypt the message and send it to her. Alice would then get an encrypted message which she would be able to decrypt with her private key and then read the message. Cryptographic hashing is another fundamental part of blockchain technology [8]. Hashing takes a string as an input, for example, a password of any length, and turns it into a fix length output. A cryptographic hash function has three properties [8]:

- Deterministic: A specific input will always have the same output.
- Irreversible: The output cannot be used to determine input.
- Collision resistance: Two different inputs will always have different outputs.

2.1.1. Decentralization

All blockchain networks are decentralized; this means no single entity owns the blockchain [3]. A decentralized network of many nodes is similar to a supercomputer, but instead of one computer doing all the computing, each computer on the network does a part of a given task. This method is both faster and cheaper than a supercomputer. A User can submit a task, like big data analysis, to the network, and the task will then be divided, processed and reassembled after all the computers are done with their given task.

2.1.2. Different types

There are three different types of blockchain networks: Public, Private, and Federated/Consortium. All these types originated from the Bitcoin blockchain when people realized that it could be used for any value transaction or agreement. Private institutions started using the core idea of blockchain as a distributed ledger, which spawned the creation of permissioned, private and federated blockchains.

Public Blockchain

The most common type of blockchain is a public network, which is based on a Proof of Work (PoW) consensus algorithm. The Public blockchain is open source and not permissioned, meaning anyone can download the code and run a public node on their local computer. Users can make, validate, and see transactions with the public block explorer. Every transaction is transparent, but every user is anonymous. Examples of public blockchains are Bitcoin and Ethereum.

Private Blockchain

In a private blockchain, write permissions are kept locally within an organization, while read permissions may be public or restricted. Groups take advantage of blockchain technology by using it internally within a company and use it to verify transactions. Private blockchains are faster and more scalable. Examples of private blockchains are MONAX and Multichain.

Federated/Consortium Blockchain

A Federated blockchain is controlled by a group of people. While public blockchains allow anyone with an internet connection to participate, Federated blockchains allow only certain people to participate. These blockchains are faster and more private than public blockchains. A Consortium blockchain is used most often by banks. The consensus process is controlled by a selected number of nodes, where a number of financial institutions must sign every block in order for the block to be valid. An example of this type of blockchain is R3 Corda.

2.2. How it works

To explain how blockchain works it is essential to know that it is a peer-to-peer system and distributed ledger. It can reduce cost and increases trust between participants. The increase of trust comes from it being a decentralized, transparent and tamper-proof database. This means that information is stored across all participants instead of in a single database. All participants in a blockchain can see, change and take part in the decision of whether or not to approve changes to the ledger, but ultimatly, it is down to the decision of the majority.

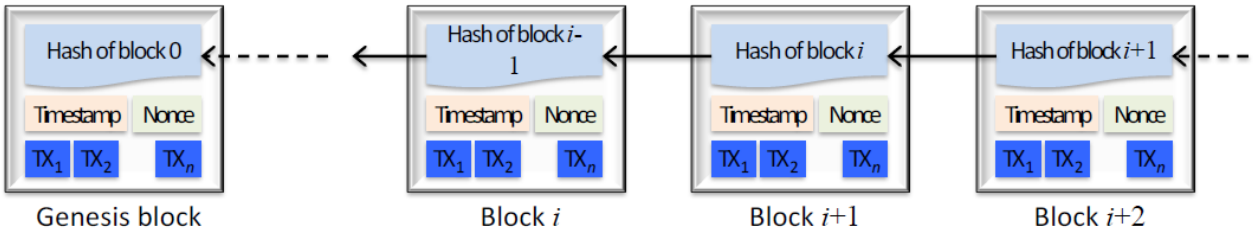


Figure 1 - Example of how blockchain works

Figure 1 show a simplified illustration of how blocks and chains work. The first block in the chain is the Genesis block, where gas limit, chain id, and more is determined. When creating data, a second block containing the data is mined and added to the chain. This process repeats for every additional block mined. These blocks are now holders of the data in the system and cannot be changed. If a user wants to change the data, it will create a new block that now holds the updated data.

An example of a simple transaction is, Alice wants to send money to Bob. The transaction contains the address of the sender (Alice) and receiver (Bob), the amount of money to be traded and a unique hash. This data forms a block that will be added to the chain if the majority approve it. Since blockchain is unchangeable, participants cannot change the data of a block, so if Alice or Bob wants to change the transaction, a new block will be created with the proposed changes.

2.2.1. Distributed ledger

The distributed ledger technology (DLT) is a vital component in the blockchain. It provides a shared database where participants store their identical copy of the ledger. Unlike a traditional database, the ledger is stored across the participants, because of this, blockchain archives are decentralized. One of the benefits of DLT is that without a centralized authority, the level of trust between participants is higher. It is also crucial to understand that a blockchain ledger is a special kind of distributed ledger. The difference is that a blockchain DLT stores the data in blocks that only exists on the blockchain.

The append-only structure of blockchain means that every addition to the ledger is permanent. Entries cannot be removed or altered; all changes are stored in history logs, which makes managing and tracking records easy. These features make the ledger tamper-proof, since nothing can be changed without other participants knowing.

An example of how the distributed ledger works; Alice, Bob, and Charlie are participants on a blockchain network and each have a copy of the distributed ledger. The transaction for Alice's purchase of Bob's bike needs to first be verified by Charlie before it can be added to the distributed ledger. When the process is completed, the blockchain network updates each participant's copy permanently with the new block.

2.2.2. Smart contract

A smart contract is like a function in standard programming that only executes when the system meets a predetermined set of conditions. Since it is just a simple computer code, it is crucial to get the logic right for correct operation. Smart contracts are implemented into distributed ledgers to ensure correct operation [3], and mainly operate with an "if this, then that" methodology.

Smart contracts are a transparent, conflict-free way to handle the purchasing of services without third-party intervention. Because of this, there is no need for a lawyer or bank. The benefits of this: it amplifies the trust between the buyer and seller because blockchain is transparent and can reduce the cost of a transaction.

An example of how a smart contract with two participants work: Bob wants to sell his bike, and Alice wants to buy it. Alice purchases the bike by signing the contract. This is done by using her private key, but the exchange of money happens between her and Bob's blockchain addresses. Then the smart contract is overseen by the participants in the blockchain network. This system reduces the possibilities for scams, because if the seller or buyer does not meet requirements of the contract, the exchange is not approved. In this scenario, the smart contract is approved, Alice has transferred money to Bob and has access to the bike's blockchain address.

2.2.3. Consensum protocol and algorithms

The term consensus is defined as nodes on a network agreeing on the condition on a block. The terms consensus protocol and consensus algorithm are often used interchangeably, but they are not the same. The simplest explanation of the terms: the protocol provides a set of rules the blockchain follows, and the algorithm determines how the protocol follows these rules.

Consensus protocol keeps the participants synchronized on the network. Participants do not need to trust each other when entering an agreement because they just followed the rules provided by consensus protocol. The rules ensures that blocks don't break protocol when validating them.

Consensus algorithms are a method to make decisions within a group and are used to create equality and fairness in the network. It is essential to know that consensus algorithms do not necessarily only agree with the majority, but also agree with what benefits everyone. The blockchain in itself does not provide a decentralized environment, but consensus algorithms make the system decentralized. Blockchain consensus models are the primary way for participants in a blockchain network to reach agreements. There are many different types of consensus algorithms: PoW, Proof-of-Stake (PoS) and Practical Byzantine Faul Tolerance (PBFT).

2.3. Why blockchain in ticketing systems?

Blockchain technology can contribute to a safer ticketing system by addressing the two most prominent issues in today's systems: ticket fraud and resale in the secondary market. These issues plague the ticketing market since it reduces the level of trust a consumer has in the system.

2.3.1. Benefits

Blockchains keep a record of every transaction in the system, this makes it easy to identify resellers and, manage tickets and events. Event organizers can benefit from this because it can help increase revenue by removing the need for a broker or bank, by using cryptocurrency. It can also benefit consumers by lowering fees and help mitigate ticket fraud.

Cryptography makes tickets more secure by having them contain a unique hash, which cannot be changed. By using cryptography tickets become nearly impossible to replicate or double sell, which leads to a safer buying experience for consumers. Identity management can also help prevent attempts to sell event tickets outside of the blockchain.

A ticket can be programmed with a smart contract to enable specific rules and functions. It can also link the consumer's blockchain address to the tickets which provides a way to identify the ticket owner. Then, you can track the flow of tickets.

2.3.2. Challenge

Blockchain technology has the potential to solve many of the current problems in today's ticketing systems, but there are some challenges. Blockchains do not always scale well; it will work great for some users, but at the scale that most ticketing services operate there could be issues. When there are multiple users on a blockchain network, transactions can take a long time to process. This issue can already be seen with Ethereum and Bitcoin, where they are already having trouble solving this.

Privacy is another issue because most blockchains use a public ledger structure. Privacy is a necessity in ticketing systems because they contain personal information about users, like credit card, name, ticket information and more. There are solutions to this problem, but it is dependent on what blockchain platform the system is running on. Quorum is an example of a private blockchain, it uses a module called Tessera that makes transaction private. In this project has been used Ganache. Ganache is a personal blockchain for Ethereum development.

Another challenge for blockchain is the lack of awareness about the technology. If the blockchain community wants more industries to adopt blockchain technology, it needs more attention and acceptance from developers and consumers. Otherwise, a user might not trust a system based on this technology, so for blockchain to become more widespread, awareness needs to be raised to inform the average consumer.

3. Platforms and tools

To be able to offer these types of services on a blockchain, a developer needs a set of rules that allow for the correct operation of the application. In the case of blockchain applications, these rules come in the form of smart contracts and are the core of all functionalities in these applications. For a ticket system to function correctly, a developer needs to make sure that they have smart contracts that handle all possible scenarios. All the tools used to carry out the project are listed below.

3.1. Ganache

Ganache is open-source software that allows us to create a local blockchain. The blockchain that will be generated will be made in Javascript, which replicates the behaviour and characteristics of the famous Ethereum blockchain. Ganache is used for setting up a personal Ethereum blockchain for testing your Solidity contracts. It provides more features when compared to Remix.

After downloading and installing Ganache, we will be faced with the graphical interface of Ganache. Initially a workspace must be configured by linking it to the truffle-config file of our application. This file allows you to set some parameters such as the version of the Solidity compiler, the address and port where the ticketing application will run, the network address and port on the blockchain and the contract directories.

The Ganache homepage displays a list of ten Ethereum accounts, each with 100ETH. These accounts were generated on the local blockchain. By clicking on the key symbol, you can find the private key of each account, necessary to send transactions and to import these accounts into Metamask.

Figure 2 - Logo of Ganache

3.2. MetaMask

MetaMask is a cryptocurrency wallet that allows you to interact with decentralized applications directly from your browser. It is an extension for normal Chrome browser. The extension injects the Ethereum Web3 API into every website's javascript context, so that DAPPs can read from the blockchain. MetaMask also lets the user create and manage their own identities, so when a DAPP wants to perform a transaction and write to the blockchain, the user gets a secure interface to review the transaction, before approving or rejecting it. Because it adds functionality to the normal browser context, MetaMask requires the permission to read and write to any webpage.

Figure 3 - Logo of MetaMask

3.3. Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behaviour of accounts within the Ethereum state. Solidity is a curly-bracket language. It is influenced by C ++, Python and JavaScript, and is designed to target the Ethereum Virtual Machine (EVM). Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features. When deploying contracts, you should use the latest released version of Solidity. This is because breaking changes as well as new features and bug fixes are introduced regularly. Once realized, these smart contracts are compiled, through a compiler called “solc”, in an object that contains the ABI code, that is a javascript object that represents the contract schema, and the bytecode, which will be executed inside the EVM. In addition, we also used the Remix IDE for the development of smart contracts, which provides various tools for their compilation and deployment within a temporary network, which allows testing directly on the browser.

Figure 4 - Logo of Solidity

3.4. Truffle

Truffle was used to interface the contracts developed with the blockchain. Truffle is a development environment, testing framework and asset pipeline for Ethereum. With Truffle, you get:

- Built-in smart contract compilation, linking, deployment and binary management.
- Automated contract testing with Mocha and Chai.
- Configurable build pipeline with support for custom build processes.
- Scriptable deployment & migrations framework.
- Interactive console for direct contract communication.
- Instant rebuilding of assets during development.
- External script runner that executes scripts within a Truffle environment.

It is possible to install it via npm package across *npm install -g truffle*. We used *truffle compile*, *truffle migrate* and *truffle test* to compile contracts, deploy those contracts to the network, and run their associated unit tests.

Figure 5 - Logo of Truffle

3.5. Node.js

Node.js is an event-oriented cross-platform open-source runtime system for running **JavaScript** code. It also allows you to use JavaScript to write server-side code. This design aims to optimize throughput and scalability in web applications with many input/output operations, it is also excellent for web applications real-time system and allows the use of the Node Package Manager (NPM) which provides access to many reusable packages.

3.6. Web3

Web3.js is a collection of libraries that allow you to interact with a local or remote Ethereum node using HTTP, IPC or WebSocket. The Web3.js library was used to connect the application to the blockchain, making calls and transactions to the smart contracts present within it.

4. Requirement Engineering

The developed system pays particular attention to the management of the ticketing service for a specific event. In particular, the actions that can be performed on the system are the creation of an event, the management of tickets, the purchase and validation of tickets. Therefore, the main actors who can interact with the system have been defined, which are:

- **Ticket Reseller**
- **Ticket Validator**
- **Ticket Buyer**
- **Event Manager**

All transactions are managed with an external wallet, called Metamask, which we configured by importing the accounts of the Ganache blockchain with their balance. At the design stage, the system was modelled with the i* language.

i* is a complementary modelling language to UML. It focuses on requirements modelling, leaving UML to represent all other aspects of the software. i* we need to deepen some security aspects in the collection of requirements. In particular, the elements that characterize the language i * are the following:

- **Actor**: an actor is an entity that carries out activities, can have objectives and use resources. Represents someone interacting with the system.
- **Agent**: indicates a specific actor.
- **Role**: indicates the role.
- **Softgoal**: are the vague targets.
- **Topic**: it is important to indicate what a vague objective refers to; this is called a topic.
- **Goal**: are the precise and specific objectives.
- **Resources**: it can be the result of an activity; it is a resource that is needed to carry out an activity.

The elements of an i* diagram can be related to:

- **Strategic dependencies**: an addiction is a relationship between two different actors, we want to model the fact that when we describe a scenario, some actors depend on other actors. To achieve a goal, maybe I need another actor. There are:
    - **Goal Decomposition**: a goal is decomposed into many sub-goals that can be linked by an "and" relation (and decomposition) in which all the sub-goals must be reached to contribute to the achievement of the main goal, or by an "or" relation (or decomposition) in which it is sufficient to achieve at least one of the sub-goal.
    - **Means-end Decomposition**: a certain activity is a means to a certain end
    - **Softgoal Decomposition**: it goes to express the way in which the various elements (objectives, resources, or other activities) help in the pursuit of that vague objective, expressing it through symbols (+ or -) or through keywords.
    - **Task Decomposition**: indicates everything that must be done before the realization of a particular activity, that is, in order to perform a task it is necessary first to acquire a certain resource or to have reached a specific objective.

- **Strategic relationships**: inside the actor's boundary, there are the skills, attitudes, relationships of an actor, it contains everything that is within the competence of an actor.

4.1. Early requirement

In the Early Requirement Analysis, we try to understand the processes that the system must automate, without making assumptions about the tools of use and not handing over the software system that will then be implemented. The goal is to understand how the system works, what the problem is and what are the relevant elements of the problem.

In our Early Requirement Analysis, we have modelled 5 actors:

- **Event Manager**, the one will create the event.
- **Ticket Reseller**, the one who has the role of selling, the ticket for the single event.
- **Payment System**, system that will turn the monetary transactions, for the purchase of the ticket.
- **Buyer**, the one who buys a ticket, to attend the desired event.
- **Ticket Validator**, the one who has the task of validate the sealed ticket,

Once our actors were in, we moved on to represent the dependencies between them (we follow the order of previous list, and the flow information flows in opposite of direction):

- **Event Manager**, establish a relationship with Ticket Reseller. The first provides the numbers of seats made available for the event. In addition, the tickets sold are returned to him. For these two reports we have introduced the resource **Seat** and goal **Sell Ticket**.
- **Ticket Reseller**, establish a first relationship with Event Manager (described in the previous point). also interface with Buyer and Payment System. The ticket Reseller requests from the Buyer a series of information related to the customer; it also provides information on a ticket to the Payment System. For these two reports we have introduced the resources **Buyer personal Data** and **Ticket**.
- **Payment system**, establish a relationship with Ticket Reseller (described in the previous point). Establishes two other relationships with the buyer. In the first relation, Payment System requires the money for the purchase of the sealed ticket, in second relation, provides the purchased ticket. For these two reports we have introduced the resources **Money** and **Sealed Ticket**.
- **Buyer**, establish relationships with Ticket Reseller, Payment system (described in the previous point) and Ticket Validator. In last relation the buyer requests a sealed ticket from Ticket Validator, for this report we have introduced the resource **Sealed Ticket.**
- **Ticket validator**, establish relation with the Buyer described in the previous point.

A final step is to report the Boundary of each actor, for every actor we have start to one single goal. Below we describe the decomposition of each goal (we follow the order of first list of actors):

- **Event Manager Boundary,** we have introduced the goal **Create Event,** we have decomposed it in the task **Define Events Details**, and in turn decomposed into other 3 distinct tasks:
  - **Define date**, defines the date of the event.
  - **Define number of tickets**, defines the max number of tickets for the event.
  - **Define price**, defines price of single ticket.

  The reasoning behind the decomposition of Event Manger: use 3 info on the event (date, number of tickets and price) to define the details related to his show, one concluded everything will be able to create his event.

- **Ticket Reseller Boundary**, we have introduced the goal **Generate Ticket,** we have decomposed in the:
  - **Buyer Personal Data,** resource that reports customer info.
  - **Check Seat Avaibility**, task that provides number of places available.

The reasoning behind the decomposition of Ticket Reseller: generate a personal ticket with info of customer and checking the availability of free seats. Tickets will be created if info valid and seats are still available.

- **Payment System Boundary**, we have introduced the goal **Deliver Ticket,** we have uses one task, **Seal Ticket.** Seal ticket has been decomposed into 2 resources:
  - **Ticket**.
  - **Money**.

  The reasoning behind the decomposition of Payment System: The payment system wild liver a sealed ticket to the customer only when the transaction is successful. The transaction will be successful only if the purchased ticket coincides with that of customer and if the money coincides with the agreed price.

- **Buyer Boundary**, we have introduced the goal **Attend event**, we have decomposed it into the task **Buy Ticket**. In turn it was decomposed into:
  - resources **Money.**
  - goal **Choose Event**.

  The reasoning behind the decomposition of Buyer: The buyer chooses the desired event and provides the necessary money for the purchase. In conclusion, the client will get a valid ticket to attend the show.

- **Ticker Validator Boundary**, we have introduced the goal **Validate Ticket**, we have decomposed it into the task **Control Ticket** and in turn it was decomposed into:
  - resource **Sealed Ticket**.

  The reasoning behind the decomposition of Ticket Validator: The validator has the task of verifying a sealed ticket. If the ticket seal is original, the ticket will be validated.

In the development of the **Early Requirement**, we considered the separation between ticket generation and then applying the seal. We have also included a payment system for the management of monetary transactions external to the reseller.

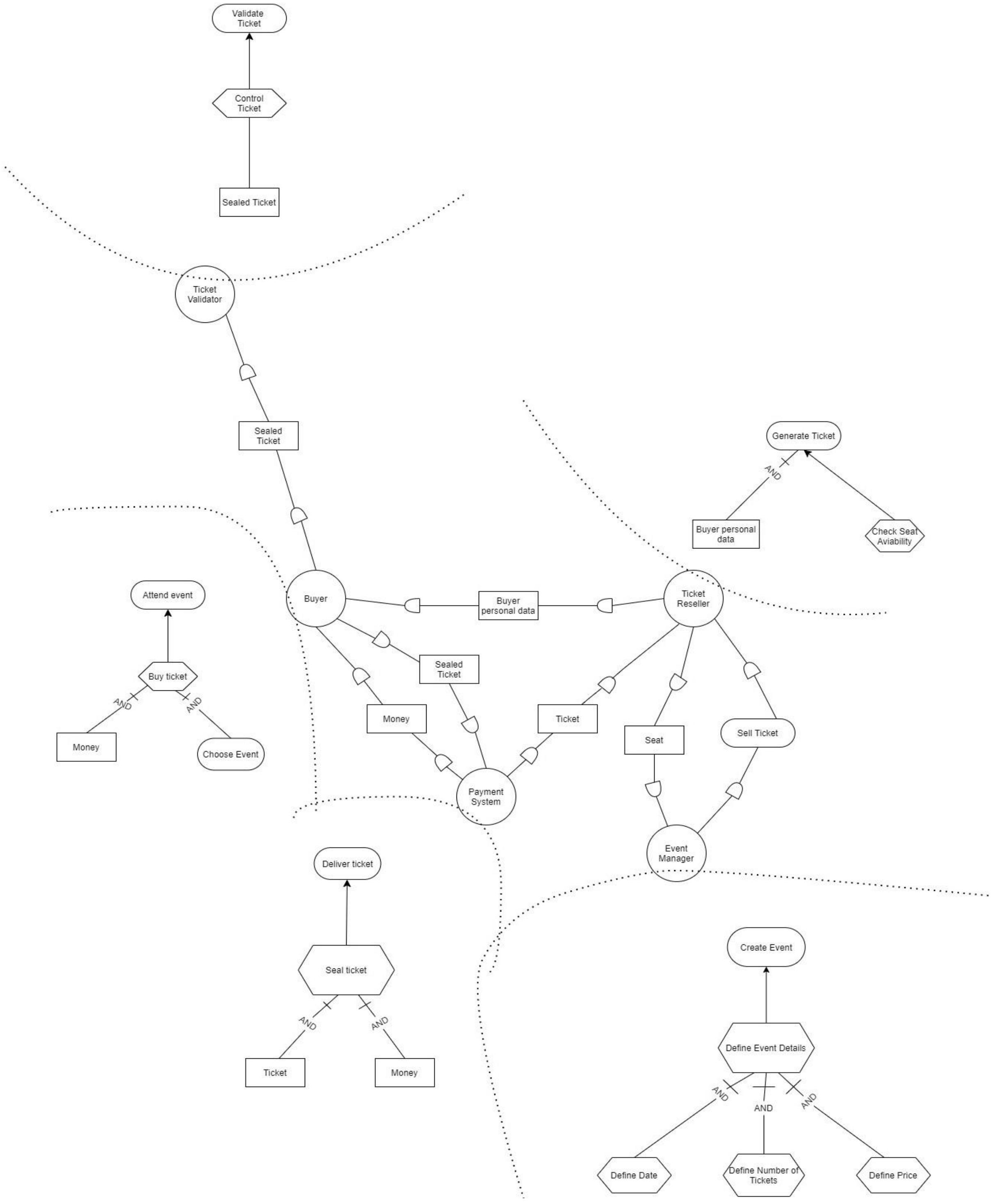


Figure 6 - Early Requirement Analysis

4.2. Late Requirement

In the **Late Requirement** analysis, defines the system's objectives. He introduces the system software that will be considered as an actor within the model and show the dependencies between the actors with environment and identifies function and non-functional requirement.

We have inserted 5 actors, already presented in the previous paragraph, by structuring their dependencies towards the system.

- **Event Manager**, we have established two dependencies with system:
    - First, Event Manager provides the number of seats made available for a single show.
    - Second, The System delivers the tickets sold for the show created by the Event Manager.
- **Ticket Reseller**, we have established two dependencies with system:
    - First, the ticket reseller requires a ticket to be validated.
    - Second, the system provides information on the ticket purchaser, which is necessary for the ticket validator.
- **Payment System**, we have established three dependencies with system:
    - First, the payment system requires the ticket from the system.
    - Second, the payment system requires the sealed ticket from system.
    - Third, the payment system requires the money needed to purchase the ticket.
- **Buyer**, we have established two dependencies with system:
    - First, the buyer provides his information for purchase of a ticket to system.
    - Second, the buyer provides the money necessary for the purchase of the ticket to system.
- **Ticket Validator**, we have established one dependency with system:
    - First, the system provides the ticket seal to be validated.

The system creates an event through the event manager. The ticket reseller will then sell the tickets for the event, so a buyer can choose from the available events in the system and buy a ticket, the payment system will manage the purchase. Upon completion, the sealed ticket will be passed to the ticket validator to be validated.

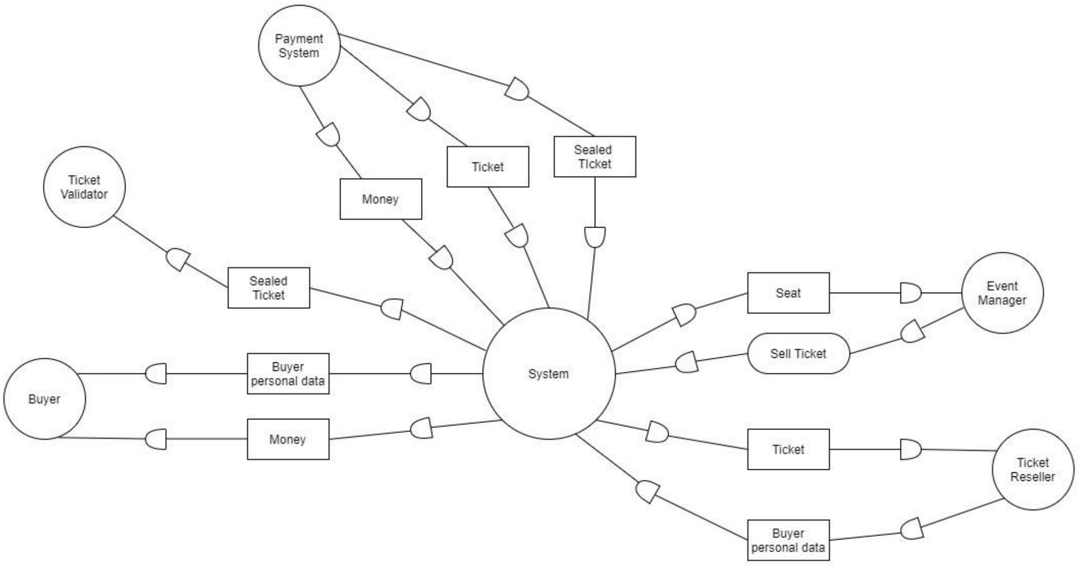


Figure 7 - Late Requirement Analysis

4.3. Architectural Design – Strategic dependency model

The **architectural design** think about the aptitudes of the software, in terms of objectives to follow, activities to be carried out and resources capable of managing.

In the strategic we divided the system into modules to understand how they interact with each other, introducing the objectives of each individual module. We went to define the modules that make up the system, each module has been remapped on a single actor previously define, in the various objectives, of each module, were declared following the delegations of tasks towards components of the system.

We have introduced 5 modules:

- **Event Manager Unit**, his goal is to create an event within our software.
- **Ticket reseller manager**, his goal is to generate the tickets needed to attend the show.
- **Payment system**, his goal is to verify the payment, after a successful transaction deliver the sealed ticket.
- **Buyer manager**, his goal is to buy a ticket, providing his own info and the money necessary to purchase the desired ticket
- **Ticket Validator manager**, his goal is validated sealed tickets.

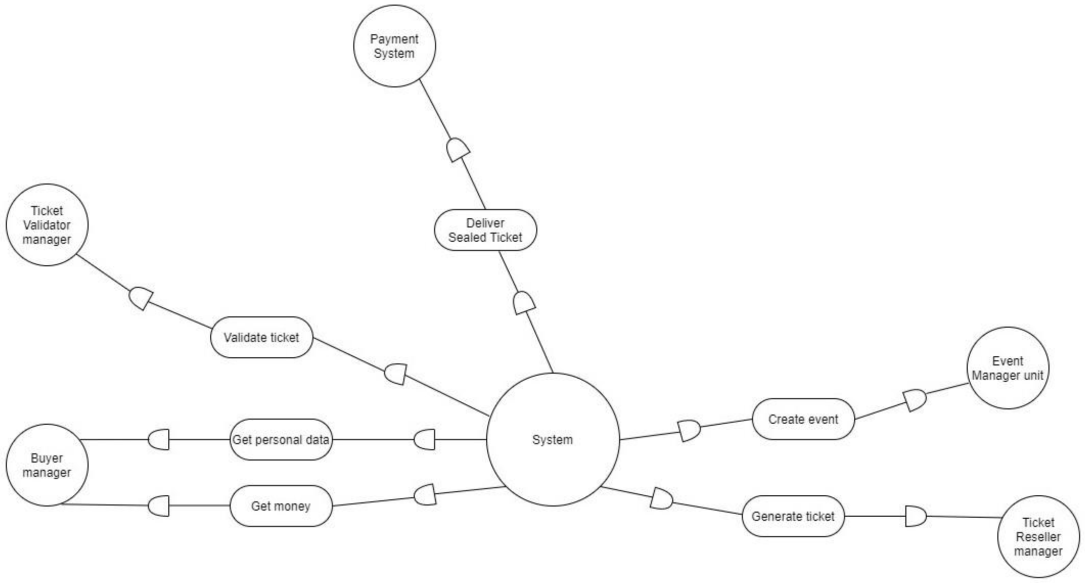


Figure 8 - Strategic dependency model

4.4. Architectural Design – Strategic Rationale model

In this section it was decided to report the **Strategic Relational Model**, that is to describe in more detail the roles of each of the previously identified modules, this was accomplished by going to represent the boundaries of each module of the software system.

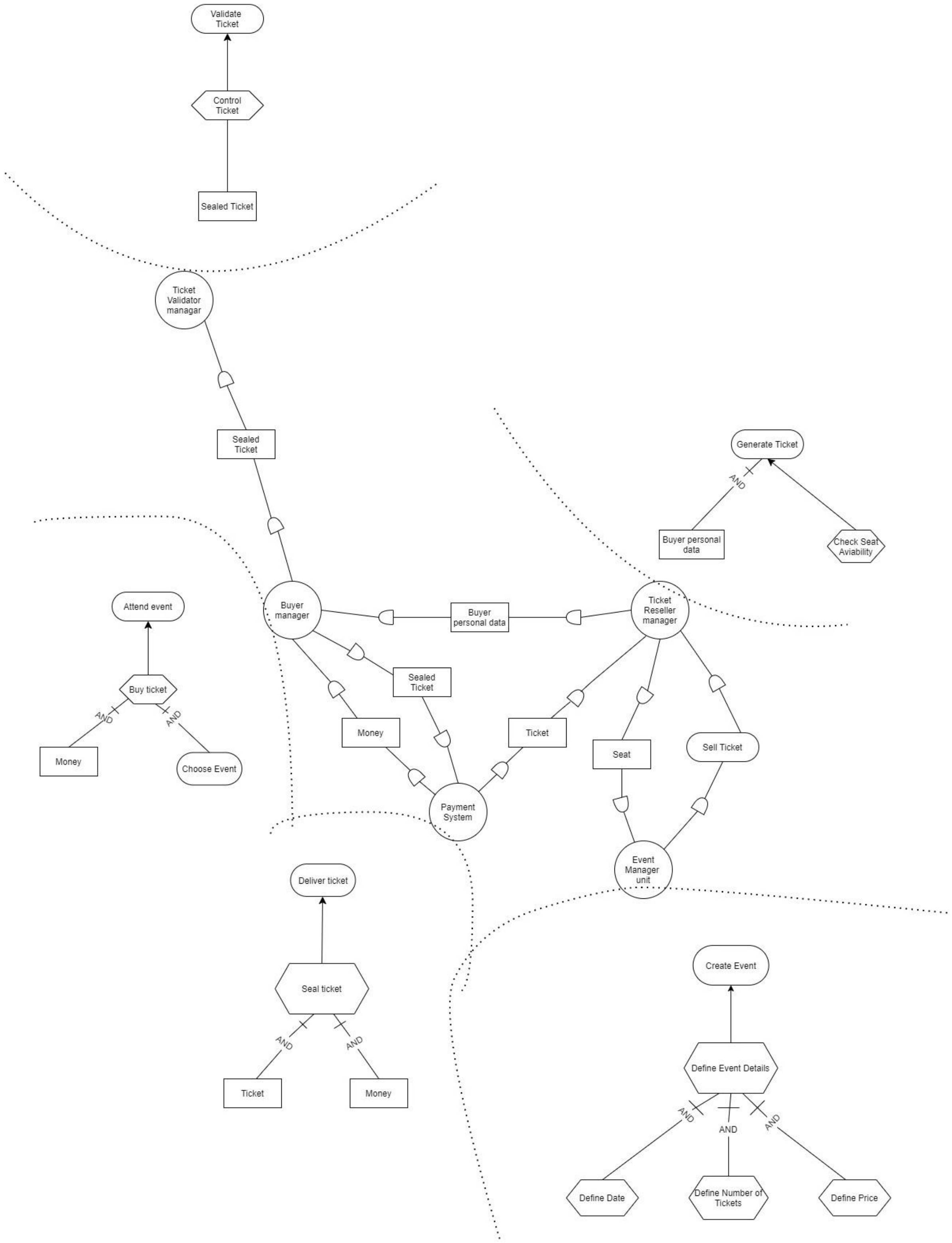


Figure 9 - Strategic rationale model

4.5. Risk identification

In this phase the security policies are analysed. **Security policies** are a set of rules that govern all aspects, relevant to safety, relating to the behaviour of individuals and processes. First of all, it is necessary to **identify the assets**, that is, everything that has value within the organization and therefore must be protected from any cyber-attacks or threats of any kind. Therefore, for each asset we establish which are the security objectives to be guaranteed, for each objective we establish organizational level security policies. The assets can be tangible and intangible: the ticket and the event are tangible assets, these are crucial elements for the system. The other assets are intangible, as they are intangible assets but still essential for the correct functioning of the system.

For the objectives of the security policies, the known triads have been used, namely:

- **CIA Triade**:
    - Confidentiality: guarantee that data not seen by unauthorized parties.
    - Integrity: data or systems are not modified or altered or destroyed.
    - Availability: Systems must always be available when someone is needed.
- **AAA Triade**:
    - **Authenticity**: guarantee that a certain information is genuine.
    - **Assurance**: guarantee that an entity behaves as expected.
    - **Accountability**: guarantee of tracing a certain entity.
- **Safety/Reliability/Resilience**:
    - **Safety**: the system does not harm people or things.
    - **Reliability**: the system delivers a service as users expect.
    - **Resilience**: the system must be able to withstand and recover from harmful events.

4.5.1. Asset identification

The assets that we have deemed appropriate to consider for our system are:

- **Validate ticket**
- **Sealed ticket**
- **Generate ticket**
- **Buyer personal data**
- **Money**
- **Ticket**
- **Seal ticket**
- **Create event**
- **Buy ticket**

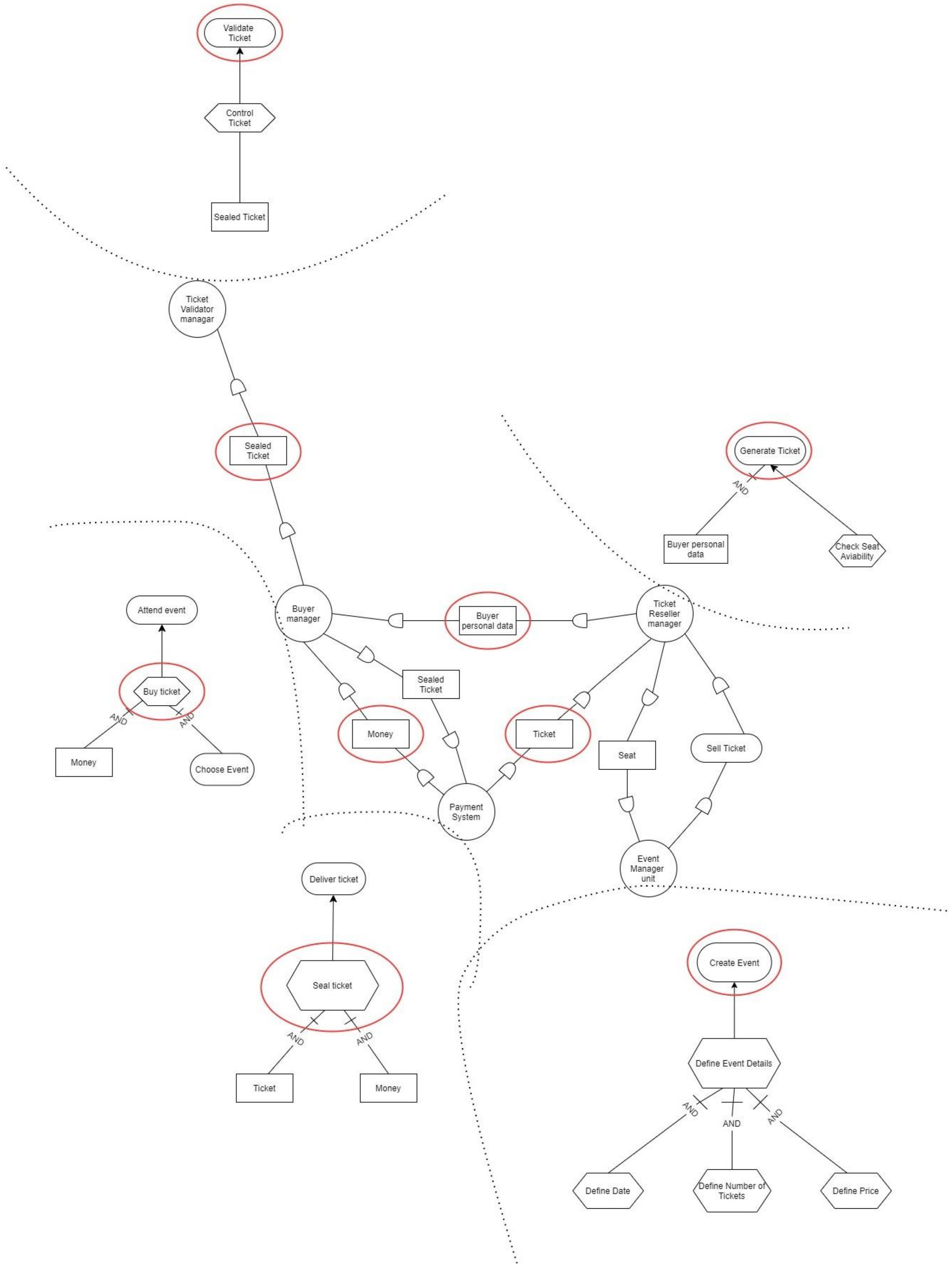


Figure 10 - Figure of assets identification on i diagram*

Each use case of the asset is described using a Jacobson table who contains some information:

- **Actors:** Entity who interact with the asset.
- **Description:** A simple description about the use case.
- **Data:** The actual data used during the use case.
- **Stimulus and Preconditions:** A simple description about condition and stimulus about the use case initialization.
- **Basic flow:** The normal flow performed for realising the use case.
- **Exception Flow:** The alternative flow of the use case when an error occurs.
- **Response and Postconditions:** A simple description about responses of the use case.
- **Non Functional Requirements:** The dependability requirements which the use case must be respect.
- **Comments:** A description about some constraints of the use case.

In this section we report the **Jacobson tables** relating to the main assets identified previously.

Use case ID: ID01 **Use case Name: Validate ticket**	
Actors	Ticket validator manager
Description	The validator authorizes the ticket and marks it as used
Data	Sealed ticket
Stimulus and Preconditions	The customer has the ticket sealed
Basic Flow	1. Customer give the ticket to ticket validator. 2. Ticket validator check the ticket seal 3. If the ticket is valid, the customer is authorized to attend at the event and the ticket is marked as used
Alternative Flow	
Exception Flow	If the ticket is invalid, the customer is not authorized to attend at the event
Response and Postconditions	The ticket cannot be re-used
Non-Functional Requirements	Assurance, Authentication, Accountability, Availability, Confidentiality
Comments	Only the ticket validator can check and mark the ticket

Use case ID: ID02 **Use case Name: Sealed ticket**	
Actors	Ticket validator manager, Buyer manager
Description	The buyer shows the ticket to the ticket validator
Data	Sealed Ticket
Stimulus and Preconditions	The ticket is ready to be validated.
Basic Flow	1. Buyer gives the ticket to the ticket validator. 2. Ticket validator check the ticket.
Alternative Flow	
Exception Flow	If the buyer doesn’t show the ticket, the customer can’t attend to the event

Response and Postconditions	Buyer has bought the ticket
Non-Functional Requirements	Confidentiality, Integrity, Availability
Comments	Only the ticket validator can receive the sealed ticket.

Use case ID: ID03 **Use case Name: Generate Ticket**	
Actors	Ticket reseller manager
Description	The ticket reseller generates a data structure with info on buyer and reserved place.
Data	Buyer personal data
Stimulus and Preconditions	The served seat must be available, receive valid personal data.
Basic Flow	1. Receives reserved seat from event manager 2. Receives personal data about buyer 3. If seat and personal data are valid, generate ticket
Alternative Flow	
Exception Flow	If buyer data are invalid or seats are unavailable the ticket isn't generated
Response and Postconditions	Send ticket at payment system.
Non-Functional Requirements	Confidentiality, Availability, Authenticity, Assurance, Accountability
Comments	The generation of the ticket is valid only if seats are available /checkable and data are valid. Only the ticket reseller can generate the ticket.

Use case ID: ID04 **Use case Name: Buyer Personal Data**	
Actors	Ticket reseller manager, Buyer manager
Description	The buyer communicates his personal data to generate the ticket with his name
Data	Buyer personal data
Stimulus and Preconditions	The buyer chooses the event to attend.
Basic Flow	1. The buyer sends his personal data 2. The ticket reseller receives and use it
Alternative Flow	
Exception Flow	There are any errors in the communication or the data are invalid, therefore is re-initialized the communication
Response and Postconditions	The ticket reseller can generate the ticket
Non-Functional Requirements	Confidentiality, Integrity, Availability
Comments	The personal data must be accessible only to the Ticket reseller.

Use case ID: ID05 **Use case Name: Money**	
Actors	Buyer Manager, Payment system
Description	Money is the cash used for ticket purchase
Data	Money
Stimulus and Preconditions	The ticket is available for purchase, and the money matches the ticket price
Basic Flow	1. Buyer makes payment 2. Payment system valid monetary transaction 3. Buyer receives sealed ticket from Payment system.
Alternative Flow	
Exception Flow	If the payment is less than price of ticket, transaction be cancelled.
Response and Postconditions	Buyers receives sealed ticket and can participate in the event.
Non-Functional Requirements	Integrity, Confidentiality, Availability
Comments	The buyer receives ticket after successful monetary transition.

Use case ID: ID06 **Use case Name: Ticket**	
Actors	Ticket Reseller Manager, Payment System
Description	The Payment System requires the ticket to the Ticket Reseller.
Data	Ticket
Stimulus and Preconditions	The ticket is generated from ticket reseller and the buyer would pay and receive it.
Basic Flow	1. The payment system requires the ticket 2. The ticket reseller sends the ticket to the payment system
Alternative Flow	
Exception Flow	The transmission of the ticket was not successful and the payment system requests again the ticket.
Response and Postconditions	The ticket is available to the Payment System.
Non-Functional Requirements	Confidentiality, Integrity, Availability
Comments	The ticket is available for only the payment system.

Use case ID: ID07 **Use case Name: Seal Ticket**	
Actors	Payment System
Description	The ticket with customer information is sealed
Data	Ticket, Money
Stimulus and Preconditions	The Payment System must have the unsealed ticket and must has received the payment from the customer
Basic Flow	1. The Payment system receives the unsealed ticket from the Ticket Reseller Manager 2. The Payment system receives money from the Buyer Manager 3. The Payment System checks if Buyer’s information is consistent with the unsealed ticket 4. The Payment System seals the ticket
Alternative Flow	
Exception Flow	The Payment System rejects the payment because there was any error in the transition or customer information are inconsistent with the unsealed ticket
Response and Postconditions	The sealed ticket must be accessible to the Buyer after the payment succeeds
Non-Functional Requirements	Confidentiality, Assurance, Accountability, Authenticity, Availability
Comments	Only the payment system can seal the ticket and customer information must be consistent with the unsealed ticket

Use case ID: ID08 **Use case Name: Create Event**	
Actors	Event Manager Unit
Description	The event manager creates a new event
Data	Event details
Stimulus and Preconditions	The event manager would like to organize an event.
Basic Flow	1. The event manager defines event details (description, date, price, number of seats) 2. The new event created is available and tickets can be bought
Alternative Flow	
Exception Flow	

Response and Postconditions	The new event must have all details specified
Non-Functional Requirements	Confidentiality, Integrity, Availability, Assurance, Accountability, Authenticity
Comments	Only the event manager can create an event at any moment or modify event details.

Use case ID: ID09 **Use case Name: Buy ticket**	
Actors	Buyer Manager
Description	The buyer buys the ticket
Data	Ticket, Money
Stimulus and Preconditions	The buyer has enough money to buy a ticket and he gives his personal information to buy ticket.
Basic Flow	1. Buyer has to choose event to attend. 2. Buyer pay the ticket’s price. 3. Buyer can receive the ticket.
Alternative Flow	
Exception Flow	There aren’t enough seats available or the buyer doesn’t have enough money.
Response and Postconditions	The buyer can show the ticket to Ticket Validator and attend the event.
Non-Functional Requirements	Authenticity, Availability, Accountability, Assurance, Confidentiality
Comments	The transition cannot be modified in any way and buyer cannot disown the purchase.

4.6. Risk analysis

All asset identified have different **non functional requirements** which had to be analysed in order to evaluate what are the different **risks**.

4.6.1. Asset value assessment & Exposure assessment

Any security policy must be translated into an appropriate system requirement. In this phase every asset identified is valued on considering the violation of security policy objectives and for single asset is valued the impact breach of security requirements. To do this, it was decided to use a qualitative rather based on a 3-level Likert scale, with numbers ranging from 1 to 3.

Each asset has been classified with value and impact.

Asset	Value	Exposure (Impact)
Validate ticket	3 Require monitor the validation process. Potentially critical for the event safety.	3 Can be used fake ticket, with economic loss. High-value loss for the event manager.
Sealed ticket	3 Require protect the validation process. Potentially safety critical.	3 The ticket can be stolen during communication
Generate Ticket	2 Require the secure generation of the ticket.	3 The ticket cannot be generated and sold. Custom ticket can be generated by anyone. Cost of restoring system.
Buyer personal data	3 Require protect the buyer personal information. Potentially safety critical.	3 Buyer personal data cannot be disclosed to anyone is not authorized. Damage to the user.
Money	3 Require controlling monetary transaction for the payment process. Potentially safety critical.	3 A monetary transaction must not be compromised. Cost of restoring system.
Ticket	2 Require ticket be consistent with the buyer and seller.	3 The ticket cannot be manipulated, defending event and buyer info.
Seal ticket	3 Require create sealed ticket necessary for the validation process.	3 Unpaid tickets can be sealed with economic loss. High-value loss
Create event	2 Require create event and generate possible sales.	3 The event details can be modified, or the entire event can be cancelled. High value losses but is possible to create a new event if the first is compromised.
Buy ticket	2 Require give a user the opportunity to purchase a ticket.	2 The buyer cannot buy the ticket and attend at the event, with economic loss

4.6.2. Threat identification

In this approach we define all possible threats for each asset using the Microsoft **STRIDE** model. This model is used in risk analysis to identify all possible threats. The table identify six possible threats which violates different properties:

- **Spoofing:** Is when someone impersonate something or someone else violating authentication process.
- **Tampering:** When the asset can be modified or delated (violation of integrity).

- **Repudiation:** An entity in the system can claim to have not performed an action.
- **Information Disclosure:** The asset information is exposed to someone unauthorized violating confidentiality.
- **Denial of Service:** The system/asset is unavailable, or the system performance are degraded.
- **Elevation of Privilege:** It’s possible to gain high capabilities without a proper authorization.

The following table shows the STRIDE table for our assets with some extra info about the other security policies (Safety, Reliability, Resilience).

	Authenticity	Integrity	Accountability	Confidentiality	Availability	Authorization/ Assurance	Safety	Reliability	Resilience
Asset	**Spoofing**	**Tampering**	**Repudiation**	**Information Disclosure**	**DoS**	**Elevation of privilege**	**Danger**	**Unreliability**	**Absence of resilience**
Validate ticket	x		x	x	x	x	x	x	x
Sealed ticket		x		x	x		x	x	x
Generate Ticket	x		x	X	x	x	x	x	x
Buyer personal data		x		x	x		x	x	x
Money		x		x	x		x	x	x
Ticket	x	x		x	x		x	x	x
Seal ticket	x		x	x	x	x	x	x	x
Create event	x	x	x	x	x	x	x	x	x
Buy ticket	x		x	X	x	X	x	x	x

4.7. Risk decomposition

In this section we describe how all identified threats unfold and how to calculate the **risk factor** of each possible attack at our system.

4.7.1. Attack assessment

At this stage, we analyse all possible attacks. It breaks down each threat into attacks that could be carried out on the system and the possible ways in which these attacks can take place. To do this, we think about attack vectors by creating Attack Trees. In them all the techniques that can be used and that contribute to the violation of a certain requirement are expressed. Some of these are:

- **DoS (Denial of Service):** indicates a malfunction due to a cyber attack which causes the stop of the resources of a system that provides a service to clients. It can cause corruption of data copies or a hardware attack that could lead to a shutdown of the entire system thus violating reliability and availability.
- **Session ID from cookies**: consists in modifying the contents of a cookie in order to circumvent the security mechanisms. The attacker can obtain private and unauthorized information from a user. Above we find the escalation of privileges (the attacker manages to acquire access to resources which normally should be protected). This refers to unauthorized access that violates the requirements of confidentiality, integrity, availability, authenticity and responsibility.
- **Data Breach:** it is a cybersecurity incident in which information is accessed without authorization. User information is likely to become public without your consent. Data Breach is often caused by the careless use of data by third-party apps. It is immediately underneath credential theft that leads to unauthorized access.
- **Ransomware:** is a type of malware that restricts access to the infected device, demanding a ransom to be paid to remove the restriction. Only some data or even the entire system may be encrypted. This type of attack violates the requirements for authenticity, reliability, and resilience.
- **Phishing:** in this case an attacker deceives the victim, pretending to be a reliable body, convincing her to provide personal information, access codes, bank details, etc. The direct consequence of Phishing is the theft of credentials and spoofing which lie below interception and unauthorized access.

A case of abuse is an interaction between the system and one or more actors whose result is harmful to the system, or to one of the actors or to one of the stakeholders. They describe how a possible external attacker can take actions to get to violate one or more security objectives.

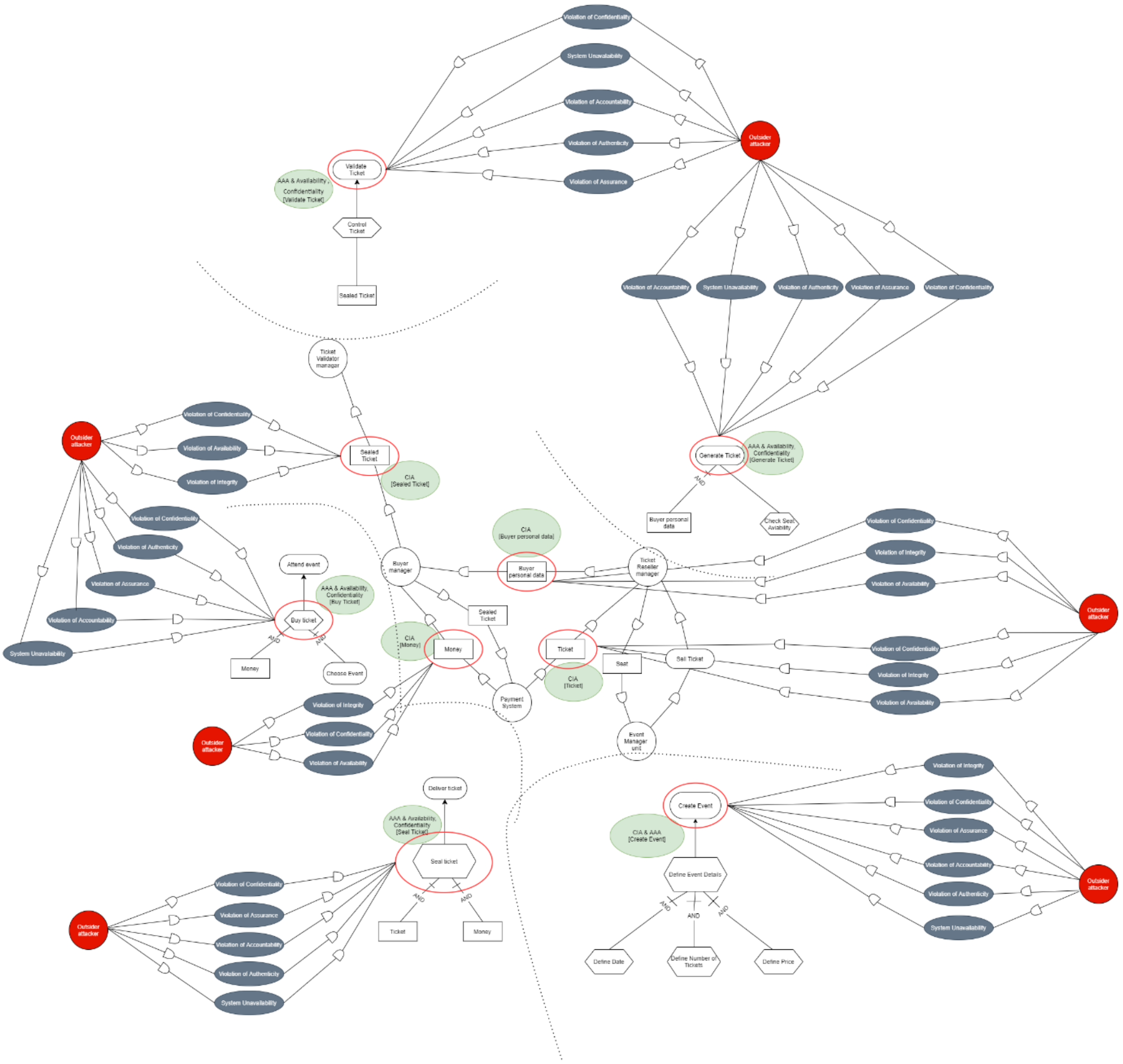


Figure 11 - Figure of Abuse Case

The cases of improper use or misuse case, the inverse of the use cases, describe functions that the system must not allow. Misuse cases describe something that has an impact and a cost. A mis-actor is someone who intentionally or accidentally initiates a misuse case. They help us define additional users who can cause a security target to be breached.

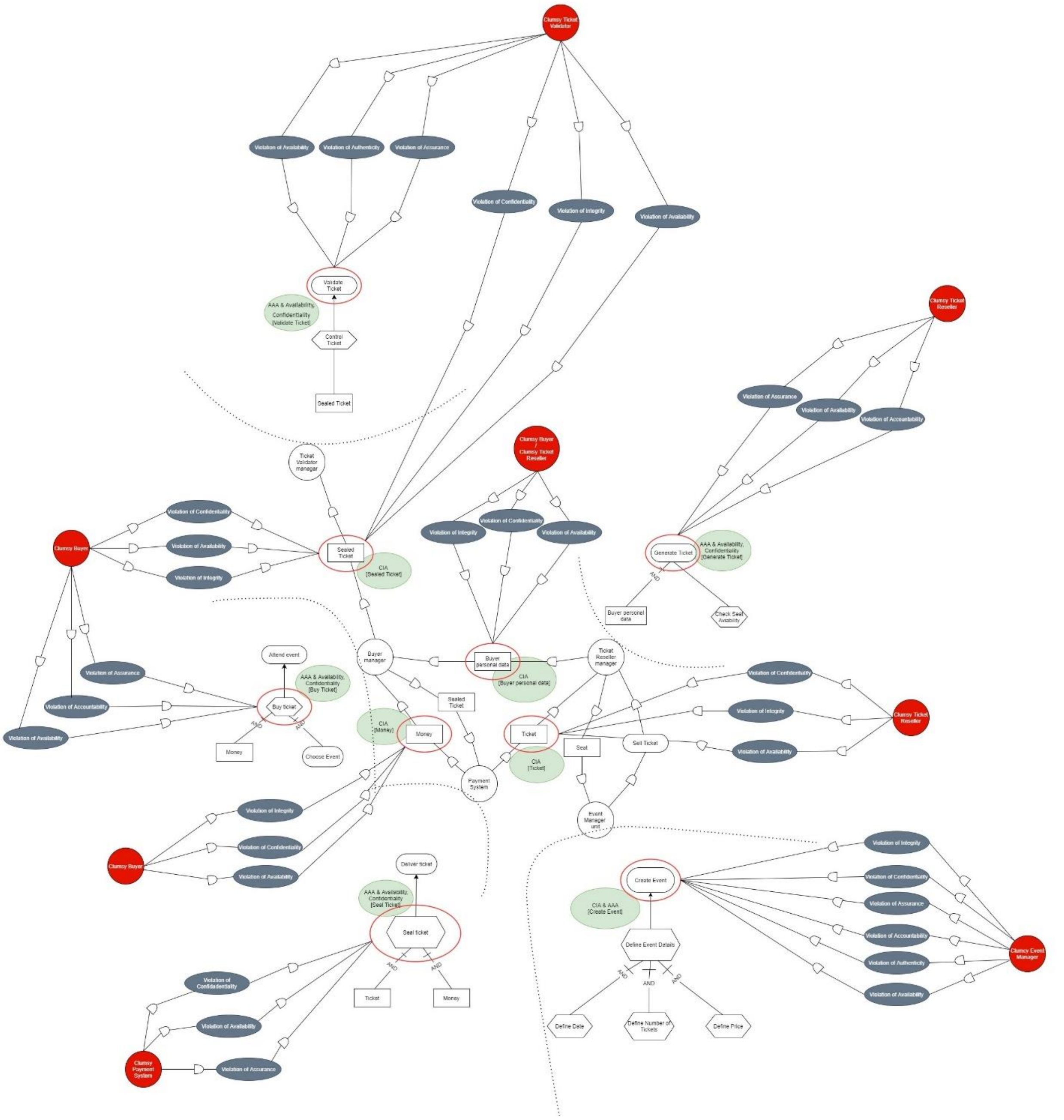


Figure 12 - Figure of Misuse Case

After defining the attack trees (abuse and misuse cases) for each asset we describe all possible attack trees for each attacker/mis-actor in order to discover all possible attacks and understand which threats they realize. How we saw in the abuse case there is an **Outside Attacker** who can threaten our system with different types of attacks. The figure below shows the Attack Tree relating to abuse case in which it is assumed that the external attacker has malicious intentions.

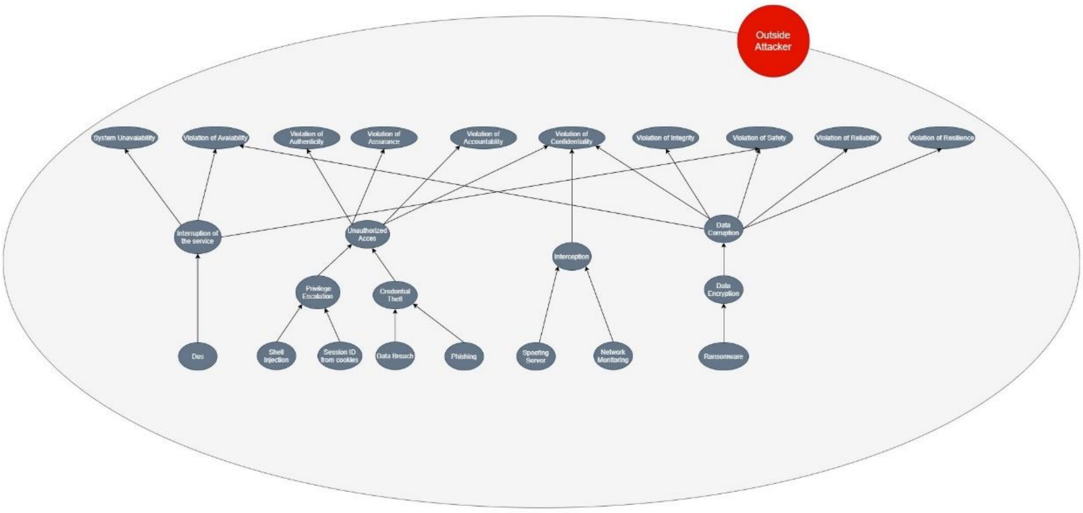


Figure 13 - Attack Tree outside attacker

The attack trees must be for the different mis-actors of our system. We have identified different mis-actors:

- **Clumsy Ticket Validator:** A ticket validator access sealed ticket information and can accidentally modify/expose these information's or not validate properly the ticket. He can also lose his credential data which will lead to the loss of service.
- **Clumsy Ticket Reseller:** A ticket reseller module generates ticket structure and access customer personal information which can accidentally being exposed to unauthorized entities.
- **Clumsy Payment System:** The payment system module seal ticket data after the customer confirms monetary transaction. This is a software module and if is not developed properly can accidentally expose ticket info or not seal the ticket.
- **Clumsy Event Manager:** An event manager can accidentally create/modify or end events or lose his credential data.
- **Clumsy Buyer:** A inattentive customer can buy wrong ticket, modify his personal info, or expose his ticket info.

The figures below show the Attack Trees relating to misuse cases in which it is assumed that users can be inattentive and not malicious.

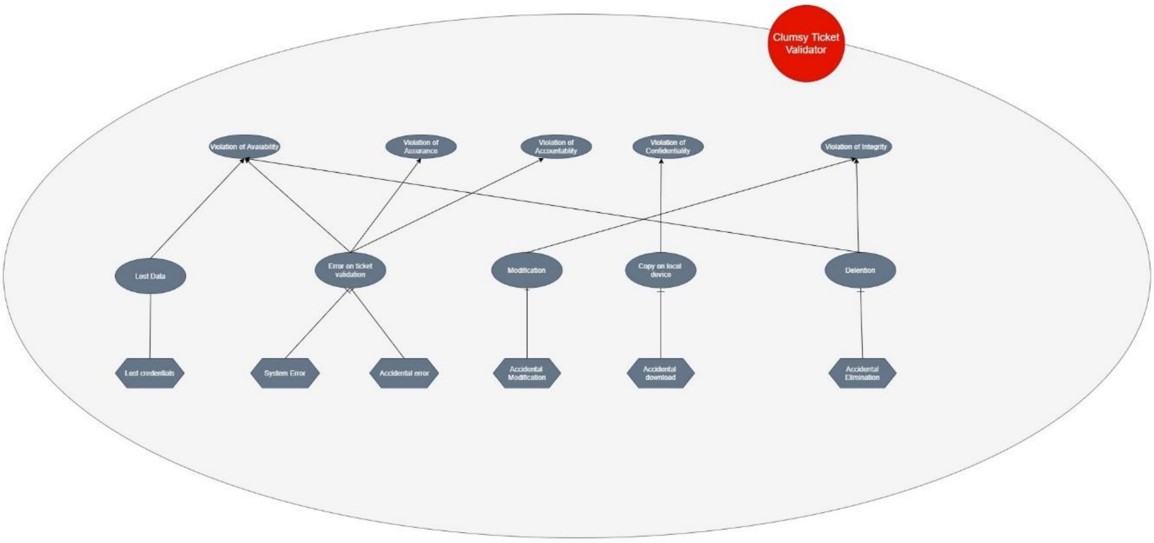


Figure 14 - Figure of Clumsy Ticket Validator

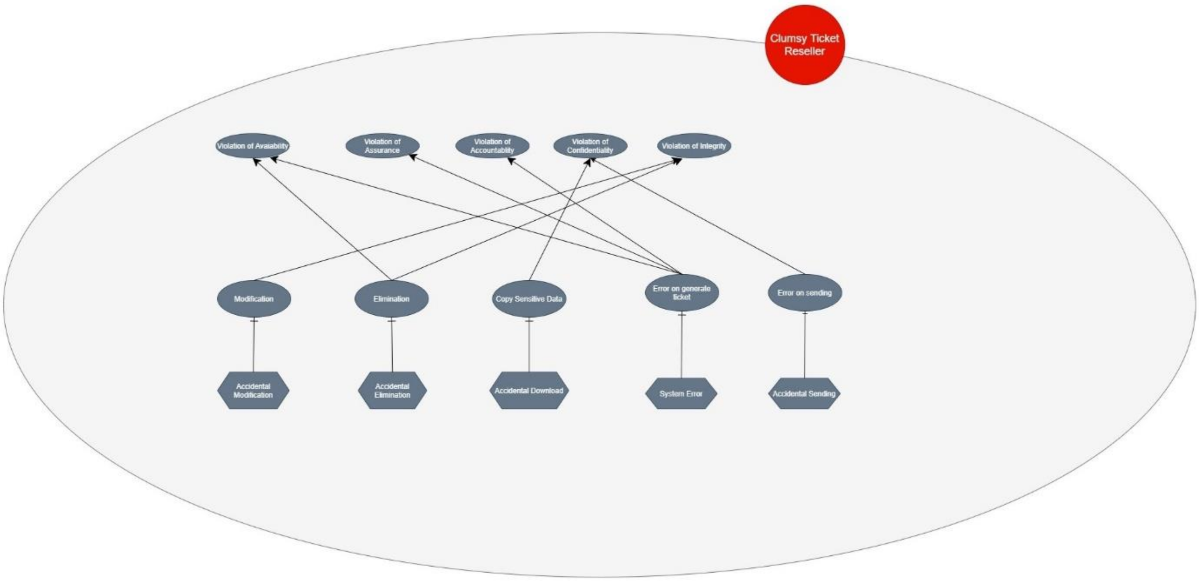


Figure 15 - Figure of Clumsy Ticker Reseller

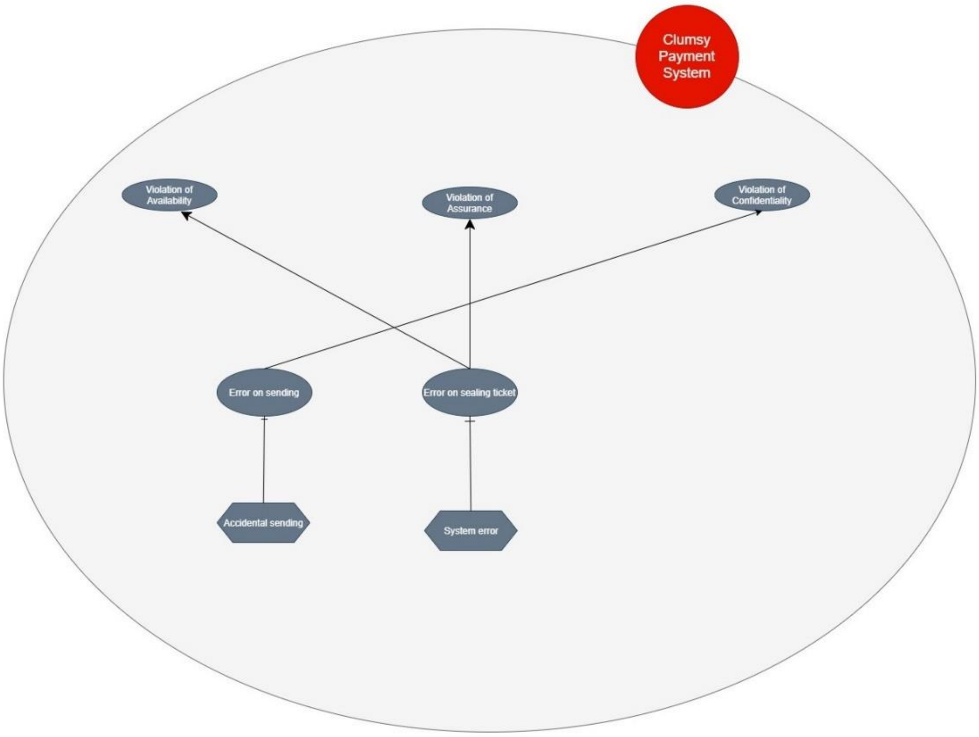


Figure 16 - Figure of Clumsy Payment System

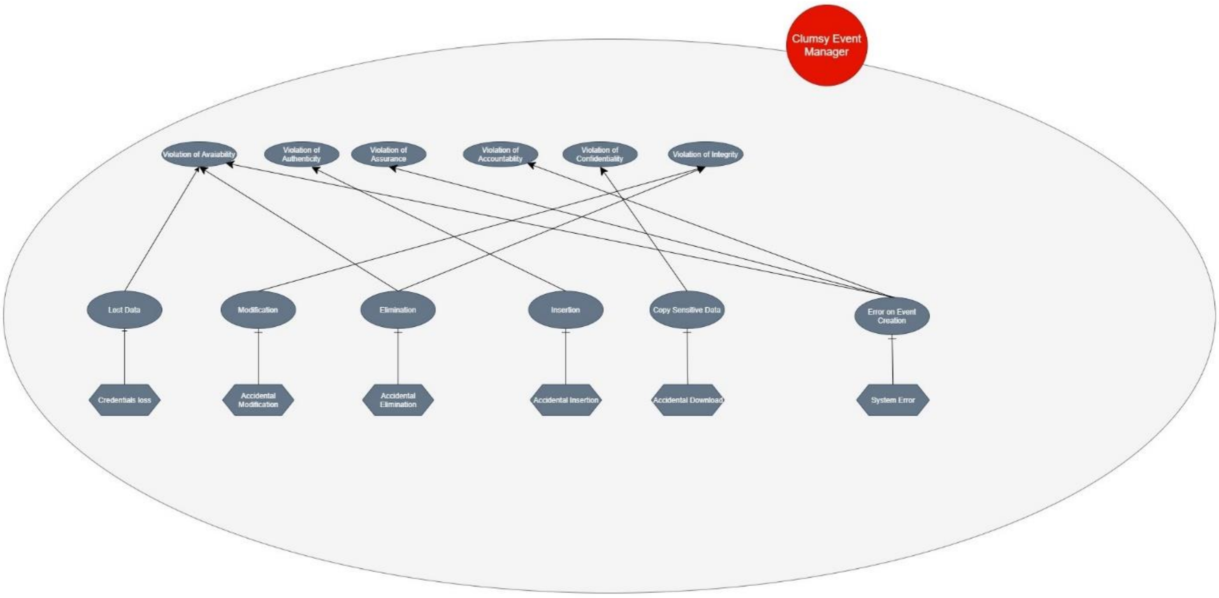


Figure 17 - Figure of Clumsy Event Manager

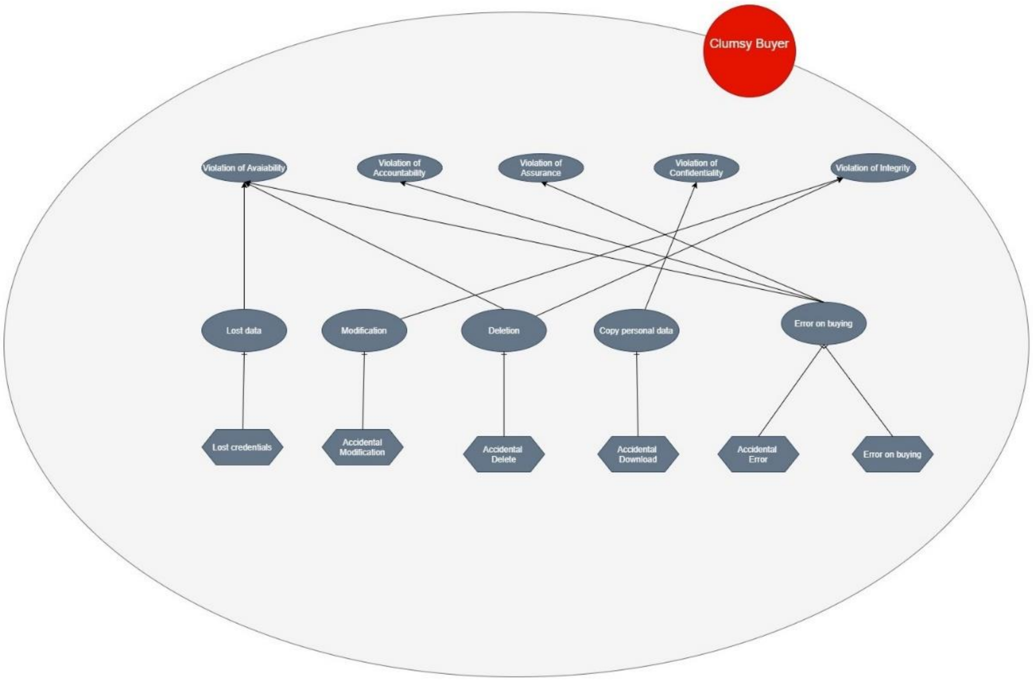


Figure 18 - Figure of Clumsy Buyer

After the attack trees specification, we report some information for each attack identified in a Jacobson use case table. This table contains some information about:

- **Actors**: actors involved in the scenario.
- **Description**: a short description of the scenario.
- **Data**: data involved in the scenario.
- **Stimulus and Preconditions**: conditions that must occur to make this scenario possible and stimulus that initiates it.
- **Attack Flow:**
- **Response and Post-conditions**: the state after the successful completion of this scenario and the produced result.
- **Mitigations:** Technology used to reduce attack risk
- **Non-functional requirements**: non-functional requirements for this scenario.

In this phase we fill the table till response and post-conditions row. The last two rows will be completed during risk reduction phase. In this section we report the Jacobson complete tables for each use case.

Use case ID: AT-01-1
Use case Name: Unauthorized Access Validate Ticket

Actors	Ticket Validator Manager, Attacker
Description	The sealed ticket can be theft from ticket validation services by credential theft or privilege escalation using many bugs
Data (asset)	Sealed Ticket
Stimulus and Pre.	There are several security issues that affect the server
Attack 1 Flow	There is a privilege escalation do by shell injection or session ID theft from cookies
Attack 2 Flow	There is a credential theft from data breach or a phishing attack.
Response and Post.	The attacker has stolen tickets and can use it.
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in.
Non Functional Requirements	Implements ACLs system

Use case ID: AT-01-2
Use case Name: Interrupt of Service Validate Ticket

Actors	Ticket Validator Manager, Attacker
Description	An attacker performs a dos attack to interrupt the service of validation ticket.
Data (asset)	Sealed Ticket
Stimulus and Pre.	The server is not protected from dos attack
Attack 1 Flow	The sealed ticket is unavailable after a Denial of Service attack
Response and Post.	It is not possible to validate a ticket
Mitigations	Each type of user can only perform the operations defined for that type of user and only some requests are filtered, while other requests are blocked.
Non Functional Requirements	See the requests to filter

Use case ID: AT-01-3 Use case Name: Data corruption Validate Ticket	
Actors	Ticket Validator Manager, Attacker
Description	An attacker corrupts the data of sealed ticket making it unvalidable
Data (asset)	Sealed Ticket
Stimulus and Pre.	Presence of vulnerabilities within Validate Ticket System
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data, so the data is unusable by the system.
Response and Post.	A ticket has wrong data and it can’t be validated.
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in, so there are less chance that your data will be compromised.
Non Functional Requirements	Implements ACLs system

Use case ID: AT-01-4 Use case Name: Error on Ticket Validation	
Actors	Ticket Validator Manager, Clumsy Ticket Validator
Description	The ticket Validator accidentally doesn’t validate the ticket or is unable to validate the ticket
Data (asset)	Sealed Ticket
Stimulus and Pre.	There are many bugs on validation process or the ticket validator is inattentive.
Attack 1 Flow	The ticket validator is distracted and forgets to validate the ticket
Response and Post.	A ticket not validated cannot be used
Mitigations	Through input sanitization, the system tries to avoid the mistake on the ticket validation task.
Non Functional Requirements	Implement input sanitization

Use case ID: AT-01-5 **Use case Name: Lost credential data (Ticket Validator)**	
Actors	Ticket Validator Manager, Clumsy Ticket Validator
Description	The ticket Validator losses his credential data and cannot validate tickets anymore
Data (asset)	Sealed Ticket
Stimulus and Pre.	The attacker can exploit system vulnerabilities or deceive a user
Attack 1 Flow	The attacker luring a user of our system, through malicious message and email.
Attack 2 Flow	The attacker can get the credentials of the user.
Attack 3 Flow	Unfortunate user lost credentials of your account.
Response and Post.	The validator user can’t access in his account and cannot validate ticket, so the buyers cannot attend at the event.
Mitigations	With a password update system, the user can insert his email and the system send it one link to reset the password.
Non Functional Requirements	Implement a password update system

Use case ID: AT-02-1 **Use case Name: Interception of Sealed Ticket**	
Actors	Ticket Validator Manager, Buyer Manager, Attacker
Description	The Buyer transfer sealed ticket data to the Ticket Validator. An Attacker intercepts the data transfer and takes a copy.
Data (asset)	Sealed Ticket
Stimulus and Pre.	The communication between buyer and ticket validator is not encrypted or the server is not authenticated
Attack 1 Flow	A spoof server is set up between the server and the buyer, so the buyer believes it is interacting with the real system
Attack 2 Flow	A network monitor is added to the system and packets from the buyer to the server are intercepted.
Response and Post.	The intercepted ticket can be stealing and used for malicious purposes
Mitigations	Sealed tickets are encrypted
Non Functional Requirements	Implements hash functions to encrypt data

Use case ID: AT-02-2 **Use case Name: Corruption of Sealed Ticket**	
Actors	Ticket Validator Manager, Buyer Manager, Attacker
Description	An Attacker corrupts the sealed ticket data making unreadable
Data (asset)	Sealed Ticket
Stimulus and Pre.	Presence of vulnerabilities within Buyer System
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data. The encrypted data is unusable by the Buyer and Ticket Validator Systems.
Response and Post.	A ticket sealed has wrong data and it can’t be used.
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in, so there are less chance that sealed ticket will be intercepted.
Non Functional Requirements	Implements ACLs system

Use case ID: AT-02-3 **Use case Name: Sealed Ticket Unavailable**	
Actors	Ticket Validator Manager, Buyer Manager, Attacker
Description	An Attacker makes unavailable the sealed ticket with a Denial of Service Attack
Data (asset)	Sealed Ticket
Stimulus and Pre.	The server is not protected from dos attack
Attack 1 Flow	The sealed ticket is unavailable after a Denial of Service attack
Response and Post.	User cannot use and does not have access to the sealed ticket
Mitigations	The server infrastructure is mirroring in more server, so the redundant architecture makes impossible an dos attack.
Non Functional Requirements	Replicate infrastructure

Use case ID: AT-02-4 **Use case Name: Lost credential data (Buyer)**	
Actors	Buyer Manager, Clumsy Buyer
Description	The Buyer losses credential data and cannot access sealed ticket anymore
Data (asset)	Sealed Ticket
Stimulus and Pre.	The buyer is distracted, does not remember the credentials data.
Attack 1 Flow	The buyer cannot login on his account, in the system, and cannot verify his tickets.
Response and Post.	The buyer user can’t access in his account and cannot buy ticket, so he cannot attend at the event.
Mitigations	With a password update system, the user can insert his email and the system send it one link to reset the password.
Non Functional Requirements	Implement a password update system

Use case ID: AT-02-5 **Use case Name: Accidental Elimination Sealed Ticket**	
Actors	Buyer Manager, Ticket Validator, Clumsy Buyer, Clumsy Ticket Validator
Description	The Buyer or the ticket validator accidentally eliminate the sealed ticket
Data (asset)	Sealed Ticket
Stimulus and Pre.	The buyer or the ticket validator are inattentive and the system doesn’t require another confirmation of the operation.
Attack 1 Flow	The sealed ticket is deleted by the buyer because is distracted or click “delete” accidentally
Attack 2 Flow	The sealed ticket is deleted by the ticket validator because is distracted or click “delete” accidentally
Response and Post.	The sealed ticket is no longer available to the user
Mitigations	For each delete operation will request a confirmation, to avoid accidental clicks.
Non Functional Requirements	Implement a system that require a confirmation for the operation

Use case ID: AT-02-6 **Use case Name: Accidental Modification Sealed Ticket**	
Actors	Buyer Manager, Ticket Validator, Clumsy Buyer, Clumsy Ticket Validator
Description	The Buyer or the ticket validator accidentally modifies the sealed ticket
Data (asset)	Sealed Ticket
Stimulus and Pre.	The buyer or the ticket validator is inattentive and the system doesn't require another confirmation of the operation.
Attack 1 Flow	The sealed ticket is modified from the buyer because is distracted
Attack 2 Flow	The sealed ticket is modified from the ticket reseller because is distracted
Response and Post.	The ticket is modified and therefore the data can be compromised.
Mitigations	For each modification will request a confirmation, to avoid accidental operation.
Non Functional Requirements	Implement a system that require a confirmation for the operation

Use case ID: AT-02-7 **Use case Name: Copy on local device (Sealed Ticket)**	
Actors	Buyer Manager, Ticket Validator, Clumsy Buyer, Clumsy Ticket Validator
Description	The Buyer or the ticket validator accidentally download a copy of the sealed ticket
Data (asset)	Sealed Ticket
Stimulus and Pre.	The buyer or the ticket validator is inattentive(distracted) and the system doesn't require another confirmation of the operation.
Attack 1 Flow	The buyer clicks on "download" accidentally.
Attack 2 Flow	The ticket validator clicks on "download" accidentally.
Response and Post.	The buyer or ticket validator has a copy of the ticket in their local system, this is potentially dangerous.
Mitigations	All ticket validator can only open with a custom PDF viewer with the "Save" function disabled.
Non Functional Requirements	Create a custom PDF viewer with iText 7 API

Use case ID: AT-03-1 Use case Name: Interrupt of service generation ticket	
Actors	Ticket Reseller Manager, Attacker
Description	An attacker performs a dos attack to interrupt the service of generation ticket.
Data (asset)	Buyer personal data
Stimulus and Pre.	The server is not protected from dos attack
Attack 1 Flow	The ticket is unavailable after a Denial of Service attack
Response and Post.	It is not possible to generate ticket
Mitigations	The server infrastructure is mirroring in more server, so the redundant architecture makes impossible an dos attack.
Non Functional Requirements	Replicate infrastructure

Use case ID: AT-03-2 Use case Name: Unauthorized Access Generate Ticket	
Actors	Ticket Reseller Manager, Attacker
Description	The ticket or buyer personal data can be theft from ticket generation services by privilege escalation using many bugs
Data (asset)	Ticket, Buyer personal data
Stimulus and Pre.	There are several security issues that affect the server
Attack 1 Flow	There is a privilege escalation do by shell injection
Response and Post.	An unauthorized entity can generate tickets deliberately
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in.
Non Functional Requirements	Implements ACLs system

Use case ID: AT-03-3 **Use case Name: Data corruption (Generate Ticket)**	
Actors	Ticket Reseller Manager, Attacker
Description	An attacker corrupts the data of the ticket or buyer personal data making it unusable
Data (asset)	Ticket, Buyer personal data
Stimulus and Pre.	Presence of vulnerabilities within Ticket reseller system
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data, so the data is unusable by the system.
Response and Post.	A ticket has wrong data, and it can't be used.
Mitigations	Filtering the accesses, only authorized users can access the functions of ticket reseller manager
Non Functional Requirements	Implement filtering of accesses

Use case ID: AT-03-3 **Use case Name: Data corruption (Generate Ticket)**	
Actors	Ticket Reseller Manager, Attacker
Description	An attacker corrupts the data of the ticket or buyer personal data making it unusable
Data (asset)	Ticket, Buyer personal data
Stimulus and Pre.	Presence of vulnerabilities within Ticket reseller system
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data, so the data is unusable by the system.
Response and Post.	A ticket has wrong data, and it can't be used.
Mitigations	Filtering the accesses, only authorized users can access the functions of ticket reseller manager
Non Functional Requirements	Implement filtering of accesses

Use case ID: AT-04-1
Use case Name: Interception of personal data

Actors	Buyer Manager, Ticket Reseller Manager, Attacker
Description	The buyer transfers personal data to ticket reseller. An attacker can intercept the data transfer and takes a copy.
Data (asset)	Buyer personal data
Stimulus and Pre.	The communication between buyer and ticket reseller is not encrypted or the server is not authenticated
Attack 1 Flow	A spoof server is set up between the server and the buyer, so the buyer believes it is interacting with the real system
Attack 2 Flow	A network monitor is added to the system and packets from the buyer to the server are intercepted.
Response and Post.	The customer personal data can be stolen from an attacker
Mitigations	All personal data are encrypted
Non Functional Requirements	Implements hash functions to encrypt data

Use case ID: AT-04-2
Use case Name: Corruption of personal data

Actors	Buyer Manager, Ticket Reseller Manager, Attacker
Description	An Attacker corrupts the buyer personal data making unreadable
Data (asset)	Buyer personal data
Stimulus and Pre.	Presence of vulnerabilities within Buyer System
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data. The encrypted data is unusable by the Buyer and Ticket Reseller Systems.
Response and Post.	The personal data has wrong data, and it does not coincide with the originals.
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in.
Non Functional Requirements	Implements ACLs system

Use case ID: AT-04-3 **Use case Name: Personal data Unavailable**	
Actors	Buyer Manager, Ticket Reseller Manager, Attacker
Description	An Attacker makes unavailable the buyer personal data with a Denial of Service Attack
Data (asset)	Buyer personal data
Stimulus and Pre.	The server is not protected from dos attack
Attack 1 Flow	The buyer personal data is unavailable after a Denial of Service attack
Response and Post.	The user and the ticket reseller can’t access buyer personal data
Mitigations	The server infrastructure is mirroring in more server, so the redundant architecture makes impossible an dos attack.
Non Functional Requirements	Replicate infrastructure

Use case ID: AT-04-4 **Use case Name: Accidental Elimination of personal data**	
Actors	Buyer Manager, Clumsy Buyer
Description	The buyer accidentally eliminates his personal data from the system
Data (asset)	Buyer personal data
Stimulus and Pre.	The buyer is inattentive, and the system doesn’t require another confirmation of the delete operation.
Attack 1 Flow	The buyer personal data is deleted by the buyer because is distracted or click “delete” accidentally
Response and Post.	The personal data is no longer available
Mitigations	For each delete operation will request a confirmation, to avoid accidental clicks.
Non Functional Requirements	Implement a system that require a confirmation for the operation

Use case ID: AT-04-5 **Use case Name: Accidental Modification of personal data**	
Actors	Buyer Manager, Ticket Reseller Manager, Clumsy Buyer, Clumsy Ticket Reseller
Description	The buyer or the ticket reseller unit accidentally modify personal data
Data (asset)	Buyer personal data
Stimulus and Pre.	The buyer is inattentive and the system doesn't require another confirmation of the operation.
Attack 1 Flow	The buyer personal data is modified from the buyer because is distracted
Response and Post.	The buyer personal data are compromised
Mitigations	For each modification will request a confirmation, to avoid accidental operation.
Non Functional Requirements	Implement a system that require a confirmation for the operation

Use case ID: AT-04-6 **Use case Name: Copy on local device (Personal Data)**	
Actors	Buyer Manager, Clumsy Buyer
Description	The Buyer accidentally downloads a copy of his personal data stored in the system
Data (asset)	Buyer personal data
Stimulus and Pre.	The buyer is inattentive(distracted) and the system doesn't require another confirmation of the operation.
Attack 1 Flow	The buyer clicks on "download" accidentally.
Response and Post.	The buyer has a copy of the personal data in their local system, this is potentially dangerous.
Mitigations	All reports can only be opened with a custom PDF viewer integrated in the software with the "Save" function disabled
Non Functional Requirements	Create a custom PDF viewer with iText 7 API

Use case ID: AT-05-1 Use case Name: Interception of money transaction	
Actors	Buyer Manager, Payment system, Attacker
Description	The attacker intercepts the data transfer regarding the money transaction.
Data (asset)	Money
Stimulus and Pre.	The communication between buyer and payment system is not encrypted or the server is not authenticated.
Attack 1 Flow	A spoof server is set up between the server and the buyer, so the buyer believes it is interacting with the real system
Attack 2 Flow	A network monitor is added to the system and packets from the buyer to the server are intercepted.
Response and Post.	The transaction can be stolen from an attacker
Mitigations	The transaction data are encrypted
Non Functional Requirements	Implements hash functions to encrypt data

Use case ID: AT-05-2 Use case Name: Corruption of transition	
Actors	Buyer Manager, Payment system, Attacker
Description	The attacker corrupts money transition and make it unusable.
Data (asset)	Money
Stimulus and Pre.	Presence of vulnerabilities for Payment System
Attack 1 Flow	The attacker exploits vulnerabilities on Payment System to encrypts transition.
Attack 2 Flow	The transition data is unusable for Buyer Manager and Payment System.
Response and Post.	A money transition has wrong data, and it can’t be used.
Mitigations	All accesses are checked by using an ACLs, where only restricted users have the permission to log in.
Non Functional Requirements	Implement ACLs system

Use case ID: AT-05-3 **Use case Name: Copy on local device (Transition)**	
Actors	Buyer Manager, Clumsy Buyer
Description	The buyer accidentally downloads a copy of sensitive data of transition.
Data (asset)	Money
Stimulus and Pre.	The buyer is inattentive, and the system doesn’t require another confirmation of the operation.
Attack 1 Flow	The buyer clicks on “download” accidentally.
Response and Post.	A copy of the transaction is stored in the local device
Mitigations	All reports can only be opened with a custom PDF viewer integrated in the software with the "Save" function disabled
Non Functional Requirements	Create a custom PDF viewer with iText 7 API

Use case ID: AT-05-4 **Use case Name: Interrupt of Service (Transition)**	
Actors	Buyer Manager, Payment system, Attacker
Description	The system is not available, and the transaction cannot be executed.
Data (asset)	Money
Stimulus and Pre.	The payment system is not protected from dos attack
Attack 1 Flow	The attacker performs a Denial of Service attack to payment system
Response and Post.	The transition is no longer available for confirmation and sealing ticket task
Mitigations	The server infrastructure is mirroring in more server, so the redundant architecture makes impossible an dos attack.
Non Functional Requirements	Replicate infrastructure

Use case ID: AT-06-1 **Use case Name: Interception of ticket**	
Actors	Payment system, Ticket Reseller Manager, Attacker
Description	The attacker intercepts the ticket and take a copy.
Data (asset)	Ticket
Stimulus and Pre.	There are vulnerabilities in the communication system Payment System and Ticket reseller
Attack 1 Flow	A spoof server is set up between Payment System and Ticket Reseller Manager. The payment system believes it is interacting with the real system.
Attack 2 Flow	A network monitor is added to system and packets from Payment System to the server are intercepted.
Response and Post.	The ticket can be stolen from an attacker
Mitigations	Ticket data structure contains encrypt data
Non Functional Requirements	Implements hash functions to encrypt data

Use case ID: AT-06-2 **Use case Name: Corruption of ticket**	
Actors	Payment system, Ticket Reseller Manager, Attacker
Description	An attacker corrupts the data of ticket making it unusable
Data (asset)	Ticket
Stimulus and Pre.	Presence of vulnerabilities within payment system or ticket reseller services
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data, so the data is unusable by the system.
Response and Post.	A ticket has wrong data, and it can’t be used.
Mitigations	Filtering the accesses, only authorized users can access the functions of ticket reseller manager
Non Functional Requirements	Implement filtering of accesses

Use case ID: AT-06-3 Use case Name: Ticket unavailable	
Actors	Payment system, Ticket Reseller Manager, Attacker
Description	An attacker makes unavailable the ticket with a Denial of Service attack
Data (asset)	Ticket
Stimulus and Pre.	The server is not protected from dos attack
Attack 1 Flow	The attacker performs a Denial of Service attack to payment system
Response and Post.	The ticket is no longer available for sealing task
Mitigations	The server infrastructure is mirroring in more server, so the redundant architecture makes impossible an dos attack.
Non Functional Requirements	Replicate infrastructure

Use case ID: AT-06-4 Use case Name: Accidental modification of ticket	
Actors	Ticket Reseller Manager, Clumsy Ticket Reseller
Description	The Ticket Reseller Manager accidentally modify ticket
Data (asset)	Ticket
Stimulus and Pre.	The ticket reseller is inattentive, and system doesn't require another confirmation of the operation
Attack 1 Flow	The ticket is modified from the Ticker Reseller cause is distracted.
Response and Post.	The modified ticket has wrong data, it does not coincide with the original.
Mitigations	For each modification will request a confirmation, to avoid accidental operation.
Non Functional Requirements	Implement a system that require a confirmation for the operation

Use case ID: AT-06-5 **Use case Name: Accidental elimination of ticket**	
Actors	Ticket Reseller Manager, Clumsy Ticket Reseller
Description	The Ticket Reseller accidentally eliminates the ticket from the system.
Data (asset)	Ticket
Stimulus and Pre.	The Ticket Reseller are inattentive, and system doesn’t require another confirm of delete operation.
Attack 1 Flow	The ticket is deleted from the Ticket Reseller cause is distracted.
Response and Post.	The ticket is no longer available to the user
Mitigations	For each delete operation will request a confirmation, to avoid accidental clicks.
Non Functional Requirements	Implement a system that require a confirmation for the operation

Use case ID: AT-06-6 **Use case Name: Accidentally sending the ticket to someone not authorized**	
Actors	Ticket Reseller Manager, Clumsy Ticket Reseller
Description	The Ticket Reseller accidentally sends the ticket to someone not authorized.
Data (asset)	Ticket
Stimulus and Pre.	The Ticket Reseller are inattentive, and system doesn’t require another confirm of delete operation.
Attack 1 Flow	The ticket is sends to not authorized unit, from the Ticket Reseller cause is distracted.
Response and Post.	The ticket data is accessible to an unauthorized entity
Mitigations	The ticket data is encrypted
Non Functional Requirements	Implements encryption primitives

Use case ID: AT-07-1 **Use case Name: Interrupt of Service Seal Ticket activity**	
Actors	Payment System, Attacker
Description	The system is unavailable after a Denial of Service attack and the payment system can’t seal the ticket
Data (asset)	Ticket, Money
Stimulus and Pre.	The payment system is not protected from dos attack
Attack 1 Flow	The attacker performs a Denial of Service attack to payment system
Response and Post.	Impossible to seal ticket
Mitigations	The server infrastructure is mirroring in more server, so the redundant architecture makes impossible a dos attack.
Non Functional Requirements	Replicate infrastructure

Use case ID: AT-07-2 **Use case Name: Unauthorized access (Seal Ticket)**	
Actors	Payment System, Attacker
Description	The ticket or transaction data can be theft from seal ticket services by privilege escalation using many bugs
Data (asset)	Ticket, Money
Stimulus and Pre.	There are several security issues that affect the server
Attack 1 Flow	There is a privilege escalation do by shell injection
Response and Post.	An unauthorized entity accesses the sealed ticket
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in.
Non Functional Requirements	Implements ACLs system

Use case ID: AT-07-3 **Use case Name: Data corruption (Seal Ticket)**	
Actors	Payment System, Attacker
Description	The ticket or transaction data can be corrupted by an attacker making sealing impossible
Data (asset)	Ticket, Money
Stimulus and Pre.	Presence of vulnerabilities within Sealing Ticket System
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data, so the data is unusable by the system.
Response and Post.	A ticket has wrong data, and it cannot be sold.
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in.
Non Functional Requirements	Implement ACLs system

Use case ID: AT-07-4 **Use case Name: Error on seal ticket activity**	
Actors	Payment System, Clumsy Payment System
Description	The Payment System accidentally don’t seal the ticket or is unable to do it.
Data (asset)	Ticket, Money
Stimulus and Pre.	There are many bugs on seal process
Attack 1 Flow	There are many bugs on sealing process, or the ticket validator is inattentive
Response and Post.	The ticket isn’t sealed
Mitigations	Through input sanitization, the system tries to avoid the mistake on the sealing activity.
Non Functional Requirements	Implements input sanitization

Use case ID: AT-07-5 **Use case Name: Accidental sending of the sealed ticket**	
Actors	Payment System, Clumsy Payment System
Description	The Payment System accidentally sends the sealed ticket to an unauthorized entity
Data (asset)	Sealed Ticket
Stimulus and Pre.	There are many bugs on seal process
Attack 1 Flow	There are many bugs on sending process, or the ticket validator is inattentive
Response and Post.	The sealed ticket is accessible to an unauthorized entity
Mitigations	The sealed ticket data is encrypted
Non Functional Requirements	Implements encryption primitives

Use case ID: AT-08-1 **Use case Name: Interrupt of service Create Event**	
Actors	Event manager, Attacker
Description	The event manager cannot create an event for an interrupt of service.
Data (asset)	Event
Stimulus and Pre.	The system is not protected from dos attack.
Attack 1 Flow	The attacker performs a Denial of Service attack to the system
Response and Post.	Impossible to create a new event
Mitigations	Filtering the accesses, only authorized users can access the functions of event manager
Non Functional Requirements	Implement filtering of accesses

Use case ID: AT-08-2 **Use case Name: Unauthorized access Create Event**	
Actors	Event manager, attacker
Description	The event manager platform can access from an attacker by credential can be theft or through privilege escalation using many bugs
Data (asset)	Event
Stimulus and Pre.	There are several security issues that affect the server
Attack 1 Flow	There is a privilege escalation do by shell injection or session ID theft from cookies
Attack 2 Flow	There is a credential theft from data breach or a phishing attack.
Response and Post.	The attacker can create false events
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in.
Non Functional Requirements	Implements ACLs system

Use case ID: AT-08-3 **Use case Name: Data corruption (Create Event)**	
Actors	Event Manager, Attacker
Description	The data of event created is corrupted by attacker, the Event Manager can’t use and read data.
Data (asset)	Event
Stimulus and Pre.	Presence of vulnerabilities in Event Manager System.
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data, so the data is unusable by the system.
Response and Post.	The event has wrong data, and it cannot be seen or used.
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in.
Non Functional Requirements	Implement ACLs system

Use case ID: AT-08-4 **Use case Name: Error on Create Event**	
Actors	Event manager, Clumsy event manager
Description	The event manager accidentally doesn't create event or is unable to do it.
Data (asset)	Event
Stimulus and Pre.	There are many bugs on create event process.
Attack 1 Flow	There are many bugs on create event or event manager is inattentive.
Response and Post.	An event cannot be created
Mitigations	Through input sanitization, the system tries to avoid the mistake on the create event task.
Non Functional Requirements	Implements input sanitization

Use case ID: AT-08-5 **Use case Name: Lost credential data (Event Manager)**	
Actors	Event Manager, Clumsy event manager
Description	The event manager lost its credential data
Data (asset)	Event
Stimulus and Pre.	The event manager is distracted, does not remember the credentials data.
Attack 1 Flow	The event manager cannot login on his account, in the system, and cannot verify his event
Response and Post.	The event manager user can't access in his account and cannot create an event
Mitigations	With a password update system, the user can insert his email and the system send it one link to reset the password.
Non Functional Requirements	Implement a password update system

Use case ID: AT-08-6 **Use case Name: Accidental modification on event**	
Actors	Event Manager, Clumsy event manager
Description	The event manager accidentally modify event.
Data (asset)	Event
Stimulus and Pre.	The event manager accidentally modifies event data
Attack 1 Flow	The event is modified from the event manager cause is distracted.
Response and Post.	The event is modified and therefore the data can be compromised.
Mitigations	For each modification will request a confirmation, to avoid accidental operation.
Non Functional Requirements	Implement a system that require a confirmation for the operation

Use case ID: AT-08-7 **Use case Name: Accidental elimination of event**	
Actors	Event Manager, Clumsy event manager
Description	The event manager accidentally delete event.
Data (asset)	Event
Stimulus and Pre.	The event manager accidentally deletes event data.
Attack 1 Flow	The event is deleted from the event manager cause is distracted.
Response and Post.	The event data is no longer available to the user
Mitigations	For each delete operation will request a confirmation, to avoid accidental clicks.
Non Functional Requirements	Implement a system that require a confirmation for the operation

Use case ID: AT-08-8 **Use case Name: Accidental insertion**	
Actors	Event Manager, Clumsy event manager
Description	The event manager accidentally inserts wrong event.
Data (asset)	Event
Stimulus and Pre.	The event manager accidentally inserts wrong event.
Attack 1 Flow	The wrong event is insert from the event manager cause is distracted.
Response and Post.	The buyers can buy the ticket for wrong event, that maybe is not possible to attend.
Mitigations	For all operation is required a confirmation of action, where the event manager can re-read the information inserted.
Non Functional Requirements	Implement a confirmation operation for all action.

Use case ID: AT-09-1 **Use case Name: Interrupt of service Buy Ticket activity**	
Actors	Buyer Manager, Attacker
Description	The ticket buying activity is interrupted by a Denial of Service attack and the buyer is unable to purchase the tickets.
Data (asset)	Money, Buyer personal data
Stimulus and Pre.	The system is not protected from dos attack.
Attack 1 Flow	The attacker performs a Denial of Service attack to system.
Response and Post.	It is not possible buy the ticket to attend at the event.
Mitigations	Each type of user can only perform the operations defined for that type of user and only some requests are filtered, while other requests are blocked.
Non Functional Requirements	See the requests to filter

Use case ID: AT-09-2
Use case Name: Unauthorized access (Buy Ticket)

Actors	Buyer Manager, Attacker
Description	An attacker theft credentials data and can buy tickets
Data (asset)	Money, Buyer personal data
Stimulus and Pre.	There are several security issues that affect the server
Attack 1 Flow	There is a privilege escalation do by shell injection or session ID theft from cookies
Attack 2 Flow	There is a credential theft from data breach or a phishing attack.
Response and Post.	The attacker can buy tickets with the payment system of the buyer.
Mitigations	All accesses are checked by using an ACLs, where only restricted people have the permission to log in.
Non Functional Requirements	Implements ACLs system

Use case ID: AT-09-3
Use case Name: Data corruption (Buy Ticket)

Actors	Buyer Manager, Attacker
Description	An attacker corrupts event data and the buyer can’t choose an event and buy a ticket
Data (asset)	Ticket
Stimulus and Pre.	Presence of vulnerabilities in the System.
Attack 1 Flow	The attacker exploits one or more vulnerability on system to encrypt data, so the data is unusable by the system and the buyer cannot buy ticket.
Response and Post.	The ticket has wrong data, and it can’t be used.
Mitigations	Filtering the accesses, only authorized user can access and edit the data of the ticket.
Non Functional Requirements	Implement filtering of accesses

Use case ID: AT-09-4 **Use case Name: Lost credentials data (Buy Ticket)**	
Actors	Buyer Manager, Clumsy Buyer
Description	The buyer lost its credential data.
Data (asset)	Money, Buyer personal data
Stimulus and Pre.	The buyer is distracted, does not remember the credentials data.
Attack 1 Flow	The buyer cannot login on his account, in the system, and cannot buy ticket.
Response and Post.	The buyer user can’t access in his account and cannot buy ticket, so he cannot attend at the event.
Mitigations	With a password update system, the user can insert his email and the system send it one link to reset the password.
Non Functional Requirements	Implement a password update system

Use case ID: AT-09-5 **Use case Name: Error on buy activity**	
Actors	Buyer Manager, Clumsy Buyer
Description	The buyer fails to buy a ticket.
Data (asset)	Money, Buyer personal data
Stimulus and Pre.	The buyer is distracted, and system doesn’t require another confirm operation to buy.
Attack 1 Flow	The buyer fails to buy ticket and takes an unwanted ticket.
Response and Post.	A ticket cannot be bought
Mitigations	Through input sanitization, the system tries to avoid the mistake on the buy activity.
Non Functional Requirements	Implements input sanitization

Now each attack identified must be evaluated in order to calculate the inheritance risk. Each asset is divided in different violations (one for each non-functional requirement). Each violation has his own impact (equal or less in relation of asset impact). For each attack we estimate qualitatively a value about his likelihood using a 3-value scale (*Low=1, Medium=2, High=3*). Then we calculate for each attack the inheritance risk multiplying the violation impact and the probability of the attack (*R= IxP*). At the end all inheritance risks are grouped by violation adding all relative attacks.

The table below reports attacks assessment, and it shows impact, type of attack, probability and inheritance risk for each violation of a property for certain asset.

ASSET	Requirement	Impact	Attack	Probability	Inheritance Risk	Total Inheritance Risk
VALIDATE TICKET	Violation of **Authenticity**	2	Unauthorized access	2	4	4
	Violation of **Assurance**	2	Unauthorized access	2	4	8
			Error on validation	2	4	
	Violation of **Accountability**	2	Error on validation	2	4	4
	Violation of **Confidentiality**	1	Unauthorized access	2	2	2
	Violation of **Availability**	3	Interrupt of service	3	9	24
			Data corruption	2	6	
			Error on validation	2	6	
			Lost credential data	1	3	
SEALED TICKET	Violation of **Confidentiality**	2	Sealed Ticket Interception	3	6	14
			Sealed Ticket Data corruption	2	4	
			Copy on local device	2	4	
	Violation of **Integrity**	3	Sealed Ticket Data corruption	2	6	18
			Accidental modification	2	6	
			Accidental elimination	2	6	
	Violation of **Availability**	3	Sealed Ticket Data corruption	2	6	24
			System Unavailability	3	9	
			Lost credential data	1	3	
			Accidental elimination	2	6	
GENERATE TICKET	Violation of **Authenticity**	2	Unauthorized access	2	4	4
	Violation of **Assurance**	2	Unauthorized access	2	4	8
			Error on generation	2	4	
	Violation of **Accountability**	2	Error on generation	2	4	4
	Violation of **Confidentiality**	1	Unauthorized access	2	2	2
	Violation of **Availability**	3	Interrupt of service	3	9	21
			Data corruption	2	6	
			Error on generation	2	6	

BUYER PERSONAL DATA	Violation of **Confidentiality**	3	Personal data interception	3	9	21
			Buyer personal data corruption	2	6	
			Copy on local device	2	6	
	Violation of **Integrity**	2	Buyer personal data corruption	2	4	10
			Accidental modification	2	4	
			Accidental elimination	2	4	
	Violation of **Availability**	3	Buyer personal data corruption	2	6	18
			System unavailability	3	9	
			Accidental elimination	2	6	
MONEY	Violation of **Confidentiality**	2	Transaction interception	3	6	14
			Transaction corruption	2	4	
			Copy on local device	2	4	
	Violation of **Integrity**	3	Transaction corruption	2	6	6
	Violation of **Availability**	3	Transaction corruption	2	6	15
			Interrupt of service	3	9	
TICKET	Violation of **Confidentiality**	2	Ticket interception	3	6	14
			Ticket data corruption	2	4	
			Accidental sending of the ticket	2	4	
	Violation of **Integrity**	3	Ticket data corruption	2	6	18
			Accidental modification	2	6	
			Accidental elimination	2	6	
	Violation of **Availability**	3	Ticket data corruption	2	6	15
			System unavailability	3	9	
	Violation of **Authenticity**	2	Accidental sending of the ticket	2	4	4
SEAL TICKET	Violation of **Authenticity**	2	Unauthorized access	2	4	8
			Accidental sending of sealed ticket	2	4	
	Violation of **Assurance**	2	Unauthorized access	2	4	8
			Error on sealing ticket	2	4	
	Violation of **Accountability**	2	Error on sealing ticket	2	4	4
	Violation of **Confidentiality**	1	Unauthorized access	2	2	4
			Accidental sending of sealed ticket	2	2	
	Violation of **Availability**	3	Interrupt of service	3	9	21
			Data corruption	2	6	
			Error on sealing ticket	2	6	
		2	Unauthorized access	2	4	8

CREATE EVENT	Violation of **Authenticity**		Accidental insertion	2	4	
	Violation of **Assurance**	2	Unauthorized access	2	4	8
			Error on creation	2	4	
	Violation of **Accountability**	2	Error on creation	2	4	4
	Violation of **Confidentiality**	1	Unauthorized access	2	2	4
			Event data corruption	2	2	
	Violation of **Integrity**	3	Event data corruption	2	6	18
			Accidental modification	2	6	
			Accidental elimination	2	6	
	Violation of **Availability**	3	Interrupt of service	3	9	30
			Event data corruption	2	6	
			Error on creation	2	6	
			Lost credential data	1	3	
			Accidental elimination	2	6	
BUY TICKET	Violation of **Authenticity**	2	Unauthorized access	2	4	4
	Violation of **Assurance**	2	Unauthorized access	2	4	8
			Error on buying	2	4	
	Violation of **Accountability**	2	Error on buying	2	4	4
	Violation of **Confidentiality**	1	Unauthorized access	2	2	3
			Lost credential data	1	1	
	Violation of **Availability**	2	Interrupt of service	3	6	14
			Data corruption	2	4	
			Error on buying	2	4	

4.8. Risk reduction

After identifying all possible attacks for each violation of asset security policy with use and misuse cases, we must identify some possible control measurements to reduce the likelihood/impact of the relative attacks. In order to achieve this purpose, we report for every asset all the possible attacks with the relative threats realized and likelihood.

Asset	Spoofing	Tampering	Repudiation	Information Disclosure	DoS	Elevation of privilege	Danger	Unreliability	Absence of resilience	Attack	Probability
VALIDATE TICKET	X			X		X				Unauthorized access	2
					X		X			Interrupt of service	3
					X		X	X	X	Data corruption	2
			X		X	X	X	X	X	Error on validation	2
					X					Lost credential Data	1

SEALED TICKET				X						Sealed Ticket Interception	3
		X		X	X		X	X	X	Sealed Ticket Data corruption	2
					X		X			System unavailability	3
					X					Lost credential data	1
		X								Accidental Modification	2
		X			X					Accidental elimination	2
				X						Copy on local device	2
GENERATE TICKET					X		X			Interrupt of service	3
	X			X		X				Unauthorized access	2
					X		X	X	X	Data corruption	2
			X		X	X	X	X	X	Error on generation	2
BUYER PERSONAL DATA				X						Personal data Interception	3
		X		X	X		X	X	X	Buyer personal Data corruption	2
					X		X			System unavailability	3
		X			X					Accidental elimination	2
		X								Accidental modification	2
				X						Copy on local device	2
MONEY				X						Transaction Interception	3
		X		X	X		X	X	X	Transaction Corruption	2
				X						Copy on local device	2
					X		X			Interrupt of service	3
TICKET				X						Ticket Interception	3
		X		X	X		X	X	X	Ticket data Corruption	2

					X		X			System unavailability	3
		X								Accidental modification	2
		X			X					Accidental elimination	2
	X			X						Accidental sending of the ticket	2
SEAL TICKET					X		X			Interrupt of Service	3
	X			X		X				Unauthorized access	2
					X		X	X	X	Data corruption	2
			X		X	X	X	X	X	Error on sealing ticket	2
	X			X						Accidental sending of sealed ticket	2
CREATE EVENT					X					Interrupt of service	3
	X			X		X				Unauthorized access	2
		X		X	X		X	X	X	Event Data corruption	2
			X		X	X	X	X	X	Error on creation	2
					X					Lost credential data	1
		X								Accidental modification	2
		X			X					Accidental elimination	2
	X									Accidental insertion	2
BUY TICKET					X					Interrupt of service	3
	X			X		X				Unauthorized access	2
					X		X	X	X	Data corruption	2
				X						Lost credential data	1
			X		X	X	X	X	X	Error on buying	2

4.8.1. Control identification & Feasibility assessment

In this section we have concentrated on identifying the possible mitigation Tecnologies for each possible attack. For each attack, are identified one or more control measures that they can eliminate or mitigate the risk in terms of probability or impact, using the methodology STRIDE.

After identifying the mitigation techniques for different attack, the next step is to identify the cost and feasibility of each control measure.

Asset	Attack	Probability	Control	Cost	Feasibility
VALIDATE TICKET	Unauthorized access	2	ACLs	1	Technically feasible
	Interrupt of service	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Data corruption	2	ACLs	1	Technically feasible
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Error on validation	2	Input Sanitization	1	Technically feasible but onerous
	Lost credential Data	1	Credential recovery	3	Not always feasible, requires a backup of sensitive data
SEALED TICKET	Sealed Ticket Interception	3	Encryption	2	Technically feasible but increase latency
			ACLs	1	Technically feasible
	Sealed Ticket Data corruption	2	ACLs	1	Technically feasible
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	System unavailability	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Lost credential data	1	Credential recovery	3	Not always feasible, requires a backup of sensitive data
	Accidental Modification	2	Require confirmation	1	Technically feasible and easy to implement
	Accidental elimination	2	Require confirmation	1	Technically feasible and easy to implement
	Copy on local device	2	Encryption	2	Technically feasible but increase latency

			Create a custom PDF viewer	3	Technically feasible but complex
GENERATE TICKET	Interrupt of service	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Unauthorized access	2	ACLs	1	Technically feasible
	Data corruption	2	ACLs	1	Technically feasible
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Error on generation	2	Input Sanitization	1	Technically feasible but onerous
BUYER PERSONAL DATA	Personal data Interception	3	Encryption	2	Technically feasible but increase latency
			ACLs	1	Technically feasible
	Buyer personal Data corruption	2	ACLs	1	Technically feasible
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	System unavailability	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Accidental elimination	2	Require confirmation	1	Technically feasible and easy to implement
	Accidental modification	2	Require confirmation	1	Technically feasible and easy to implement
	Copy on local device	2	Encryption	2	Technically feasible but increase latency
			Create a custom PDF viewer	3	Technically feasible but complex
MONEY	Transaction Interception	3	Encryption	2	Technically feasible but increase latency
			ACLs	1	Technically feasible
		2	ACLs	1	Technically feasible

	Transaction Corruption		Filtering	2	Technically feasible but pay attention to not be too restrictive
	Copy on local device	2	Encryption	2	Technically feasible but increase latency
			Create a custom PDF viewer	3	Technically feasible but complex
	Interrupt of service	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
TICKET	Ticket Interception	3	Encryption	2	Technically feasible but increase latency
			ACLs	1	Technically feasible
	Ticket data Corruption	2	ACLs	1	Technically feasible
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	System unavailability	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Accidental modification	2	Require confirmation	1	Technically feasible and easy to implement
	Accidental elimination	2	Require confirmation	1	Technically feasible and easy to implement
	Accidental sending of the ticket	2	Encryption	2	Technically feasible but increase latency
			Data separation	3	Technically feasible but complex
SEAL TICKET	Interrupt of Service	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Unauthorized access	2	ACLs	1	Technically feasible
	Event Data corruption	2	ACLs	1	Technically feasible
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Error on sealing ticket	1	Input Sanitization	1	Technically feasible but onerous

	Accidental sending of the sealed ticket	2	Encryption	2	Technically feasible but increase latency
			Data separation	3	Technically feasible but complex
CREATE EVENT	Interrupt of service	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Unauthorized access	2	ACLs	1	Technically feasible
	Data corruption	2	ACLs	1	Technically feasible
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Error on creation	2	Input Sanitization	1	Technically feasible but onerous
	Lost credential data	1	Credential recovery	3	Not always feasible, requires a backup of sensitive data
	Accidental modification	2	Require confirmation	1	Technically feasible and easy to implement
	Accidental elimination	2	Require confirmation	1	Technically feasible and easy to implement
	Accidental insertion	2	Require confirmation	1	Technically feasible and easy to implement
BUY TICKET	Interrupt of service	3	Mirroring	3	Technically feasible but require more maintenance
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Unauthorized access	2	ACLs	1	Technically feasible
	Data corruption	2	ACLs	1	Technically feasible
			Filtering	2	Technically feasible but pay attention to not be too restrictive
	Lost credential data	1	Credential recovery	3	Not always feasible, requires a backup of sensitive data
	Error on buying	2	Input Sanitization	1	Technically feasible but onerous

Each control mitigation technology identified has a cost and it reduces the impact or the likelihood of the relative attack. So, we can recalculate the new residual risk which will be reduced. In order to evaluate the feasibility of the control measures and calculate the new residual risk of each attack in relation of each violation.

ASSET	REQUIREMENT	IMPACT	ATTACK	CONTROL	COST	RESIDUAL PROBABILITY/ IMPACT	RESIDUAL RISK
VALIDATE TICKET	Violation of **Authenticity**	2	Unauthorized access	ACLs	1	1	2
	Violation of **Assurance**	2	Unauthorized access	ACLs	1	1	2
			Error on validation	Filtering	2	2	4
	Violation of **Accountability**	2	Error on validation	Input Sanitization	1	1	2
	Violation of **Confidentiality**	1	Unauthorized access	ACLs	1	1	1
	Violation of **Availability**	3	Interrupt of service	Mirroring	3	2	6
				Filtering	2	2	6
			Data corruption	ACLs	1	1	3
				Filtering	2	1	3
			Error on validation	Input Sanitization	1	1	3
			Lost credential data	Credential recovery	3	2 (I)*	3
SEALED TICKET	Violation of **Confidentiality**	2	Sealed Ticket Interception	Encryption	2	1	2
				ACLs	1	2	4
			Sealed Ticket Data corruption	ACLs	1	1	2
				Filtering	2	1	2
			Copy on local device	Encryption	2	1	2
				Create a custom PDF viewer	3	1	2
	Violation of **Integrity**	3	Sealed Ticket Data corruption	ACLs	1	1	3
				Filtering	2	1	3
			Accidental modification	Require confirmation	1	1	3
			Accidental elimination	Require confirmation	1	1	3
	Violation of **Availability**	3	Sealed Ticket Data corruption	ACLs	1	1	3
				Filtering	2	1	3
			System Unavailability	Mirroring	3	2	6
				Filtering	2	2	6
			Lost credential data	Credential recovery	3	2(I)*	3
			Accidental elimination	Require confirmation	1	1	3
	Violation of **Authenticity**	2	Unauthorized access	ACLs	1	1	2
	Violation of **Assurance**	2	Unauthorized access	ACLs	1	1	2
			Error on generation	Input sanitization	1	1	2

GENERATE TICKET	Violation of **Accountability**	2	Error on generation	Input sanitization	1	1	2
	Violation of **Confidentiality**	1	Unauthorized access	ACLs	1	1	1
	Violation of **Availability**	3	Interrupt of service	Mirroring	3	2	6
				Filtering	2	2	6
			Event Data corruption	ACLs	1	1	3
				Filtering	2	1	3
			Error on generation	Input sanitization	1	1	3
BUYER PERSONAL DATA	Violation of **Confidentiality**	3	Personal data interception	Encryption	2	1	3
				ACLs	1	2	6
			Buyer personal data corruption	ACLs	1	1	3
				Filtering	2	1	3
			Copy on local device	Encryption	2	1	3
				Create a custom PDF viewer	3	1	3
	Violation of **Integrity**	2	Buyer personal data corruption	ACLs	1	1	2
				Filtering	2	1	2
			Accidental modification	Require confirmation	1	1	2
			Accidental elimination	Require confirmation	1	1	2
	Violation of **Availability**	3	Buyer personal data corruption	ACLs	1	1	3
				Filtering	2	1	3
			System unavailability	Mirroring	3	2	6
				Filtering	2	2	6
			Accidental elimination	Require confirmation	1	1	3
MONEY	Violation of **Confidentiality**	2	Transaction interception	Encryption	2	1	2
				ACLs	1	2	4
			Transaction corruption	ACLs	1	1	2
				Filtering	2	1	2
			Copy on local device	Encryption	2	1	2
				Create a custom PDF viewer	3	1	2
	Violation of **Integrity**	3	Transaction corruption	ACLs	1	1	3
				Filtering	2	1	3
	Violation of **Availability**	3	Transaction corruption	ACLs	1	1	3
				Filtering	2	1	3
			Interrupt of service	Mirroring	3	2	6
				Filtering	2	2	6
TICKET	Violation of **Confidentiality**	2	Ticket interception	Encryption	2	1	2
				ACLs	1	2	4
			Ticket data corruption	ACLs	1	1	2
				Filtering	2	1	2
			Accidental sending of the ticket	Encryption	2	1	2
				Data separation	3	1	2
	Violation of **Integrity**	3	Ticket data corruption	ACLs	1	1	3
				Filtering	2	1	3

			Accidental modification	Require confirmation	1	1	3
			Accidental elimination	Require confirmation	1	1	3
	Violation of **Availability**	3	Ticket data corruption	ACLs	1	1	3
				Filtering	2	1	3
			System unavailability	Mirroring	3	2	6
				Filtering	2	2	6
	Violation of **Authenticity**	2	Accidental sending of the ticket	Encryption	2	1	2
				Data separation	3	1	2
SEAL TICKET	Violation of **Authenticity**	2	Unauthorized access	ACLs	1	1	2
			Accidental sending of sealed ticket	Encryption	2	1	2
				Data separation	3	1	2
	Violation of **Assurance**	2	Unauthorized access	ACLs	1	1	2
			Error on sealing ticket	Input sanitization	1	1	2
	Violation of **Accountability**	2	Error on sealing ticket	Input sanitization	1	1	2
	Violation of **Confidentiality**	1	Unauthorized access	ACLs	1	1	1
			Accidental sending of sealed ticket	Encryption	2	1	1
				Data separation	3	1	1
	Violation of **Availability**	3	Interrupt of service	Mirroring	3	2	6
				Filtering	2	2	6
			Event Data corruption	ACLs	1	1	3
				Filtering	2	1	3
			Error on sealing ticket	Input sanitization	1	1	3
CREATE EVENT	Violation of **Authenticity**	2	Unauthorized access	ACLs	1	1	2
			Accidental insertion	Require confirmation	1	1	2
	Violation of **Assurance**	2	Unauthorized access	ACLs	1	1	2
			Error on creation	Input sanitization	1	1	2
	Violation of **Accountability**	2	Error on creation	Input sanitization	1	1	2
	Violation of **Confidentiality**	1	Unauthorized access	ACLs	1	1	1
			Event data corruption	ACLs	1	1	1
				Filtering	2	1	1
	Violation of **Integrity**	3	Event data corruption	ACLs	1	1	3
				Filtering	2	1	3
			Accidental modification	Require confirmation	1	1	3
			Accidental elimination	Require confirmation	1	1	3
	Violation of **Availability**	3	Interrupt of service	Mirroring	3	2	6
				Filtering	2	2	6
			Data corruption	ACLs	1	1	3
				Filtering	2	1	3
			Error on creation	Input sanitization	1	1	3

			Lost credential data	Credential recovery	3	1	3
			Accidental elimination	Require confirmation	1	1	3
BUY TICKET	Violation of **Authenticity**	2	Unauthorized access	ACLs	1	1	2
	Violation of **Assurance**	2	Unauthorized access	ACLs	1	1	2
			Error on buying	Input sanitization	1	1	2
	Violation of **Accountability**	2	Error on buying	Input sanitization	1	1	2
	Violation of **Confidentiality**	1	Unauthorized access	ACLs	1	1	1
			Lost credential data	Credential recovery	3	2(I)*	1
	Violation of **Availability**	2	Interrupt of service	Mirroring	3	2	4
				Filtering	2	2	4
			Data corruption	Filtering	2	1	2
				ACLs	1	1	2
			Error on buying	Input sanitization	1	1	2

(I)* the value refers to the impact

Then we regroup the control measurements for each violation in order to calculate total residual risk for each mitigation technology implemented to prevent a certain violation. For example, if a violation was achieved with different type of attacks which can be mitigated using the same control mitigation, then we associate at that mitigation the sum of the residual risks of the relative attacks.

ASSET	**REQUIREMENT**	**CONTROL**	**TOTAL RESIDUAL RISK**
VALIDATE TICKET	Violation of **Authenticity**	ACLs	2
	Violation of **Assurance**	ACLs	2
		Filtering	4
	Violation of **Accountability**	Input Sanitization	2
	Violation of **Confidentiality**	ACLs	1
	Violation of **Availability**	Mirroring	6
		ACLs	2
		Filtering	9

		Input Sanitization	3
		Credential recovery	2
SEALED TICKET	Violation of **Confidentiality**	Encryption	4
		ACLs	6
		Filtering	2
		Create a custom PDF viewer	2
	Violation of **Integrity**	ACLs	3
		Filtering	3
		Require confirmation	6
	Violation of **Availability**	ACLs	3
		Filtering	9
		Mirroring	6
		Credential recovery	3
		Require confirmation	3
GENERATE TICKET	Violation of **Authenticity**	ACLs	2
	Violation of **Assurance**	ACLs	2
		Input sanitization	2
	Violation of **Accountability**	Input sanitization	2
	Violation of **Confidentiality**	ACLs	1
	Violation of **Availability**	Mirroring	6
		Filtering	9
		ACLs	3
		Input sanitization	3

BUYER PERSONAL DATA	Violation of **Confidentiality**	Encryption	6
		ACLs	9
		Filtering	3
		Create a custom PDF viewer	3
	Violation of **Integrity**	ACLs	2
		Filtering	2
		Require confirmation	4
	Violation of **Availability**	ACLs	6
		Filtering	9
		Mirroring	6
		Require confirmation	3
MONEY	Violation of **Confidentiality**	Encryption	4
		ACLs	6
		Filtering	2
		Create a custom PDF viewer	2
	Violation of **Integrity**	ACLs	3
		Filtering	3
	Violation of **Availability**	ACLs	3
		Filtering	9
		Mirroring	6
TICKET	Violation of **Confidentiality**	Encryption	4
		ACLs	6

		Filtering	2
		Data separation	2
	Violation of **Integrity**	ACLs	3
		Filtering	3
		Require confirmation	6
	Violation of **Availability**	ACLs	3
		Filtering	9
		Mirroring	6
	Violation of **Authenticity**	Encryption	2
		Data separation	2
SEAL TICKET	Violation of **Authenticity**	ACLs	2
		Encrypted	2
		Data separation	2
	Violation of **Assurance**	ACLs	2
		Input sanitization	2
	Violation of **Accountability**	Input sanitization	2
	Violation of **Confidentiality**	ACLs	1
		Encryption	1
		Data separation	1
	Violation of **Availability**	Mirroring	6
		Filtering	9
		ACLs	3
		Input sanitization	3

CREATE EVENT	Violation of **Authenticity**	ACLs	2
		Require confirmation	2
	Violation of **Assurance**	ACLs	2
		Input sanitization	2
	Violation of **Accountability**	Input sanitization	2
	Violation of **Confidentiality**	ACLs	2
		Filtering	1
	Violation of **Integrity**	ACLs	3
		Filtering	3
		Require confirmation	6
	Violation of **Availability**	Mirroring	6
		Filtering	9
		ACLs	3
		Input sanitization	3
		Credential recovery	3
		Require confirmation	3
BUY TICKET	Violation of **Authenticity**	ACLs	2
	Violation of **Assurance**	ACLs	2
		Input sanitization	2
	Violation of **Accountability**	Input sanitization	2

	Violation of **Confidentiality**	ACLs	1
		Credential recovery	2
	Violation of **Availability**	Mirroring	4
		Filtering	6
		ACLs	2
		Input sanitization	2

With this final table now, we find the residual risk for each mitigation technology identified.

4.8.2. Security requirements definition

Considering previously perfumed we have to draw up the security requirement that our system must respect. This section we have assess value to cost ratio for each asset requirement and we use a 3-level scale. We define one security requirement for each asset:

- **SR1 Implement a control mechanism to protect Validate Ticket Asset**
- **SR2 Implement a control mechanism to protect Sealed Ticket Asset**
- **SR3 Implement a control mechanism to protect Generate Ticket Asset**
- **SR4 Implement a control mechanism to protect Buyer Personal Data Asset**
- **SR5 Implement a control mechanism to protect Money Asset**
- **SR6 Implement a control mechanism to protect Ticket Asset**
- **SR7 Implement a control mechanism to protect Ticket Asset**
- **SR8 Implement a control mechanism to protect Create Event Asset**
- **SR9 Implement a control mechanism to protect Buy Ticket Asset**

REQUIREMENT	VALUE	CONTROL	COST	VALUE TO COST RATIO
SR1	3	ACLs	1	3.0
		Filtering	2	1.5
		Input Sanitization	1	3.0
		Mirroring	3	1.0
		Credential Recovery	3	1.0
SR2	3	Encryption	2	1.5
		ACLs	1	3.0
		Filtering	2	1.5
		Mirroring	3	1.0
		Create a custom PDF viewer	3	1.0
		Credential Recovery	3	1.0
		Require confirmation	1	3.0
SR3	2	ACLs	1	2.0
		Input Sanitization	1	2.0

		Mirroring	3	0.7
		Filtering	2	1.0
SR4	3	Encryption	2	1.5
		ACLs	1	3.0
		Filtering	2	1.5
		Create a custom PDF viewer	3	1.0
		Mirroring	3	1.0
		Require confirmation	1	3.0
SR5	3	Encryption	2	1.5
		ACLs	1	3.0
		Filtering	2	1.5
		Create a custom PDF viewer	3	1.0
		Mirroring	3	1.0
SR6	2	Encryption	2	1.0
		ACLs	1	2.0
		Filtering	2	1.0
		Data separation	3	0.7
		Mirroring	3	0.7
		Require confirmation	1	2.0
SR7	3	Mirroring	3	1.0
		Filtering	2	1.5
		ACLs	1	3.0
		Input sanitization	1	3.0
		Encryption	2	1.5
		Data separation	3	1.0
SR8	2	Mirroring	3	0.7
		Filtering	2	1.0
		ACLs	1	2.0
		Input sanitization	1	2.0
		Credential recovery	3	0.7
		Require confirmation	1	2.0
SR9	2	Mirroring	3	0.7
		Filtering	2	1.0
		ACLs	1	2.0
		Credential recovery	3	0.7
		Input Sanitization	1	2.0

Value-to-cost		COST		
		1	2	3
VALUE	1	1.0	0.5	0.3
	2	2.0	1.0	0.7
	3	3.0	1.5	1.0

HIGH
MEDIUM
LOW

The next step consists in prioritizing the requirements in order to understand which of them should focus on. The table below shows the ratio between the residual risk associated with each individual control measure and the maximum residual risk that can be obtained for a given safety requirement.

REQUIREMENT		CONTROL	TOTAL RESIDUAL RISK	RESIDUAL RISK MAX	RISK %
SR1	Violation of **Authenticity**	ACLs	2	2	100%
	Violation of **Assurance**	ACLs	2	6	33,3%
		Filtering	4		66,6%
	Violation of **Accountability**	Input Sanitization	2	2	100%
	Violation of **Confidentiality**	ACLs	1	1	100%
	Violation of **Availability**	Mirroring	6	22	27,27%
		ACLs	2		9,1%
		Filtering	9		40,9%
		Input Sanitization	3		13,63%
		Credential recovery	2		9,1%

SR2	Violation of **Confidentiality**	Encryption	4	14	28,57%
		ACLs	6		42,85%
		Filtering	2		14,28%
		Create a custom PDF viewer	2		14,28%
	Violation of **Integrity**	ACLs	3	12	25%
		Filtering	3		25%
		Require confirmation	6		50%
	Violation of **Availability**	ACLs	3	24	12,5%
		Filtering	9		37,5%
		Mirroring	6		25%
		Credential recovery	3		12,5%
		Require confirmation	3		12,5%
SR3	Violation of **Authenticity**	ACLs	2	2	100%
	Violation of **Assurance**	ACLs	2	4	50%
		Input sanitization	2		50%
	Violation of **Accountability**	Input sanitization	2	2	100%
	Violation of **Confidentiality**	ACLs	1	1	100%
	Violation of **Availability**	Mirroring	6	21	28,6%
		Filtering	9		42,8%
		ACLs	3		14,3%
		Input sanitization	3		14,3%
	Violation of **Confidentiality**	Encryption	6	21	28,57%

SR4		ACLs	9		42,86%
		Filtering	3		14,3%
		Create a custom PDF viewer	3		14,3%
	Violation of **Integrity**	ACLs	2	7	28,5%
		Filtering	2		28,5%
		Require confirmation	3		42,8%
	Violation of **Availability**	ACLs	6	24	25%
		Filtering	9		37,5%
		Mirroring	6		25%
		Require confirmation	3		12.5%
SR5	Violation of **Confidentiality**	Encryption	4	14	28,57%
		ACLs	6		42,85%
		Filtering	2		14,28%
		Create a custom PDF viewer	2		14,28%
	Violation of **Integrity**	ACLs	3	6	50%
		Filtering	3		50%
	Violation of **Availability**	ACLs	3	18	16,6%
		Filtering	9		50%
		Mirroring	6		33,3%
SR6	Violation of **Confidentiality**	Encryption	4	14	28,57%
		ACLs	6		42,85%
		Filtering	2		14,28%

		Data separation	2		14,28%
	Violation of **Integrity**	ACLs	3	12	25%
		Filtering	3		25%
		Require confirmation	6		50%
	Violation of **Availability**	ACLs	3	18	16,6%
		Filtering	9		50%
		Mirroring	6		33,3%
	Violation of **Authenticity**	Encryption	2	4	50%
		Data separation	2		50%
SR7	Violation of **Authenticity**	ACLs	2	6	33,3%
		Encrypted	2		33,3%
		Data separation	2		33,3%
	Violation of **Assurance**	ACLs	2	4	50%
		Input sanitization	2		50%
	Violation of **Accountability**	Input sanitization	2	2	100%
	Violation of **Confidentiality**	ACLs	1	3	33,3%
		Encryption	1		33,3%
		Data separation	1		33,3%
	Violation of **Availability**	Mirroring	6	21	28,5%
		Filtering	9		42,8%
		ACLs	3		14,2%
		Input sanitization	3		14,2%

SR8	Violation of **Authenticity**	ACLs	2	4	50%
		Require confirmation	2		50%
	Violation of **Assurance**	ACLs	2	4	50%
		Input sanitization	2		50%
	Violation of **Accountability**	Input sanitization	2	5	40%
	Violation of **Confidentiality**	ACLs	2		40%
		Filtering	1		20%
	Violation of **Integrity**	ACLs	3	12	25%
		Filtering	3		25%
		Require confirmation	6		50%
	Violation of **Availability**	Mirroring	6	27	22,22%
		Filtering	9		33,33%
		ACLs	3		11,11%
		Input sanitization	3		11,11%
		Credential recovery	3		11,11%
		Require confirmation	3		11,11%

SR9	Violation of **Authenticity**	ACLs	2	2	100%
	Violation of **Assurance**	ACLs	2	4	50%
		Input sanitization	2		50%
	Violation of **Accountability**	Input sanitization	2	2	100%
	Violation of **Confidentiality**	ACLs	1	3	33,3%
		Credential recovery	2		66,6%
	Violation of **Availability**	Mirroring	4	14	25,5%
		Filtering	6		43%
		ACLs	2		14,2%
		Input sanitization	2		14,2%

The next table correlates the results of the value-to-cost ratios and the percentages of residual risks calculated previously. From this comparison, an index called value-to-risk is obtained, calculated using a specific matrix. We have assigned some labels for every measure of control of the matrix (red, orange, yellow). Using a colour scale allows us to understand which control measures are better and which are worse.

Value-to-risk		**RISK**		
		Low	**Medium**	**High**
VALUE TO COST	**High**	🟩	🟨	🟧
	Medium	🟨	🟧	🟥
	Low	🟧	🟥	🟥

REQUIREMENT	VALUE	CONTROL	COST	VALUE TO COST RATIO	RISK %	VALUE TO RISK
SR1	3	ACLs	1	3.0	100%	🟧
		Filtering	2	1.5	66,6%	🟧
		Input Sanitization	1	3.0	100%	🟧
		Mirroring	3	1.0	27,27%	🟨
		Credential Recovery	3	1.0	9,1%	🟨
SR2	3	Encryption	2	1.5	28,57%	🟨
		ACLs	1	3.0	42,85%	🟨
		Require confirmation	1	3.0	50%	🟨
		Mirroring	3	1.0	25%	🟨
		Create a custom PDF viewer	3	1.0	14,28%	🟨
		Credential Recovery	3	1.0	12,5%	🟨
		Filtering	2	1.5	37,5%	🟨
SR3	2	ACLs	1	2.0	100%	🟥
		Input Sanitization	1	2.0	100%	🟥
		Mirroring	3	0.7	28,6%	🟨
		Filtering	2	1.0	42,8%	🟧
SR4	3	Encryption	2	1.5	28,57%	🟨
		ACLs	1	3.0	42,86%	🟨
		Require confirmation	1	3.0	42,8%	🟨
		Create a custom PDF viewer	3	1.0	14,3%	🟨
		Mirroring	3	1.0	25%	🟨
		Filtering	2	1.5	37,5%	🟨
SR5	3	Encryption	2	1.5	28,57%	🟨
		ACLs	1	3.0	50%	🟨
		Filtering	2	1.5	50%	🟧
		Create a custom PDF viewer	3	1.0	14,28%	🟨
		Mirroring	3	1.0	33,3%	🟨
SR6	2	Encryption	2	1.0	50%	🟧
		Require confirmation	1	3.0	50%	🟨
		Filtering	2	1.0	50%	🟧
		Data separation	3	0.7	50%	🟥
		Mirroring	3	0.7	33,3%	🟧
SR7	3	Mirroring	3	1.0	28,5%	🟨
		Filtering	1	3.0	42,8%	🟨
		ACLs	1	3.0	50%	🟨
		Input sanitization	1	3.0	100%	🟧
		Encryption	2	1.5	33,3%	🟨
		Data separation	3	1.0	33,3%	🟨
SR8	2	Mirroring	3	0.7	22,22%	🟧
		Filtering	2	1.0	33,33%	🟨
		ACLs	1	2.0	50%	🟧
		Input sanitization	1	2.0	50%	🟧
		Credential recovery	3	0.7	11,11%	🟧
		Require confirmation	1	3.0	50%	🟨
SR9	2	Mirroring	3	0.7	25,5%	🟧

		Filtering	2	1.0	43%	🟧
		ACLs	1	2.0	100%	🟥
		Credential recovery	3	0.7	66,6%	🟧
		Input Sanitization	1	2.0	100%	🟥

Lastly, we checked the residual risk for each measure of control combined with a certain safety requirement with inheritance risk. The choice falls on those measures for which the residual risk is strictly lower than the inherent risk, effectively indicating an effective benefit. The control measures highlighted are those we have decided to not implement.

Requirement		Attack	Inheritance Risk	Control	Cost	Residual Risk	Total Residual Risk
SR1	Violation of **Authenticity**	Unauthorized access	4	ACLs	1	2	2
	Violation of **Assurance**	Unauthorized access	4	ACLs	1	2	6
		Error on validation	4	Filtering	2	4	
	Violation of **Accountability**	Error on validation	4	Input Sanitization	1	2	2
	Violation of **Confidentiality**	Unauthorized access	2	ACLs	1	1	1
	Violation of **Availability**	Interrupt of service	9	Mirroring	3	6	15
		Data corruption	6	ACLs	1	3	
		Error on validation	6	Input sanitization	1	3	
		Lost credential data	3	Credential recovery	3	3	
SR2	Violation of **Confidentiality**	Sealed Ticket Interception	6	Encryption	2	4	9
		Sealed Ticket Data corruption	4	ACLs	1	3	
		Copy on local device	4	Create a custom PDF viewer	3	2	
	Violation of **Integrity**	Sealed Ticket Data corruption	6	ACLs	1	3	9
		Accidental modification	6	Require confirmation	1	3	
		Accidental elimination	6	Require confirmation	1	3	
	Violation of **Availability**	Sealed Ticket Data corruption	6	ACLs	1	3	15
		System Unavailability	9	Mirroring	3	6	

		Lost credential data	3	Credential recovery	3	3	
		Accidental elimination	6	Require confirmation	1	3	
SR3	Violation of **Authenticity**	Unauthorized access	4	ACLs	1	2	2
	Violation of **Assurance**	Unauthorized access	4	ACLs	1	2	4
		Error on generation	4	Input sanitization	1	2	
	Violation of **Accountability**	Error on generation	4	Input sanitization	1	2	2
	Violation of **Confidentiality**	Unauthorized access	2	ACLs	1	1	1
	Violation of **Availability**	Interrupt of service	9	Mirroring	3	6	12
		Data corruption	6	Filtering	2	3	
		Error on generation	6	Input sanitization	1	3	
SR4	Violation of **Confidentiality**	Personal data interception	9	Encryption	2	6	12
		Buyer personal data corruption	6	ACLs	1	3	
		Copy on local device	6	Create a custom PDF viewer	3	3	
	Violation of **Integrity**	Buyer personal data corruption	4	ACLs	1	2	6
		Accidental modification	4	Require confirmation	1	2	
		Accidental elimination	6	Require confirmation	1	2	
	Violation of **Availability**	Buyer personal data corruption	6	ACLs	1	3	11
		System unavailability	9	Mirroring	3	6	
		Accidental elimination	3	Require confirmation	1	2	
SR5	Violation of **Confidentiality**	Transaction interception	6	Encryption	2	4	8
		Transaction corruption	4	ACLs	1	2	
		Copy on local device	4	Create a custom PDF viewer	3	2	
	Violation of **Integrity**	Transaction corruption	6	ACLs	1	3	3
	Violation of **Availability**	Transaction corruption	6	ACLs	1	3	9

		Interrupt of service	9	Mirroring	3	6	
SR6	Violation of **Confidentiality**	Ticket interception	6	Encryption	2	4	8
		Ticket data corruption	4	Filtering	2	2	
		Accidental sending of the ticket	4	Encryption	2	2	
	Violation of **Integrity**	Ticket data corruption	6	Filtering	2	3	9
		Accidental modification	6	Require confirmation	1	3	
		Accidental elimination	6	Require confirmation	2	3	
	Violation of **Availability**	Ticket data corruption	6	Filtering	2	3	9
		System unavailability	9	Mirroring	3	6	
	Violation of **Authenticity**	Accidental sending of the ticket	4	Encryption	2	2	2
SR7	Violation of **Authenticity**	Unauthorized access	4	ACLs	1	2	4
		Accidental sending of sealed ticket	4	Encryption	2	2	
	Violation of **Assurance**	Unauthorized acce ss	4	ACLs	1	2	4
		Error on sealing ticket	4	Input sanitization	1	2	
	Violation of **Accountability**	Error on sealing ticket	4	Input sanitization	1	2	2
	Violation of **Confidentiality**	Unauthorized access	2	ACLs	1	1	2
		Accidental sending of sealed ticket	2	Encryption	2	1	
	Violation of **Availability**	Interrupt of service	9	Mirroring	3	6	12
		Data corruption	6	ACLs	1	3	
		Error on sealing ticket	6	Input sanitization	1	3	
	Violation of **Authenticity**	Unauthorized acce ss	4	ACLs	1	2	4
		Accidental insertion	4	Require confirmation	1	2	
	Violation of **Assurance**	Unauthorized acce ss	4	ACLs	1	2	4

SR8		Error on creation	4	Input sanitization	1	2	
	Violation of **Accountability**	Error on creation	4	Input sanitization	1	2	2
	Violation of **Confidentiality**	Unauthorized access	2	ACLs	1	1	2
		Event data corruption	2	ACLs	1	1	
	Violation of **Integrity**	Event data corruption	6	ACLs	1	3	9
		Accidental modification	6	Require confirmation	1	3	
		Accidental elimination	6	Require confirmation	1	3	
	Violation of **Availability**	Interrupt of service	9	Filtering	2	6	18
		Event data corruption	6	ACLs	1	3	
		Error on creation	6	Input sanitization	1	3	
		Lost credential data	3	Credential recovery	3	3	
		Accidental elimination	6	Require confirmation	1	3	
SR9	Violation of **Authenticity**	Unauthorized access	4	ACLs	1	2	2
	Violation of **Assurance**	Unauthorized access	4	ACLs	1	2	4
		Error on buying	4	Input sanitization	1	2	
	Violation of **Accountability**	Error on buying	4	Input sanitization	1	2	2
	Violation of **Confidentiality**	Unauthorized access	2	ACLs	1	1	2
		Lost credential data	1	Credential recovery	3	1	
	Violation of **Availability**	Interrupt of service	6	Filtering	2	4	8
		Data corruption	4	Filtering	2	2	
		Error on buying	4	Input sanitization	1	2	

5. Design

This project requires a blockchain technology.

5.1. Secure design

After Requirement Definition step, we detect the main control techniques. Analysing the residual risk, the cost of the control and inheritance risk we choose many convenient controls:

- **ACLs**
- **Filtering**
- **Credential recovery**
- **Input sanitization**
- **Require confirmation**
- **Mirroring**
- **Encryption**

Some of these, can be implemented adopting some technologies and third-party software.

In accordance with our security choices, we use:

- **Metamask** (to ensure ACLs and credential recovery control)
- **Blockchain** (to ensure filtering, mirroring and encryption)

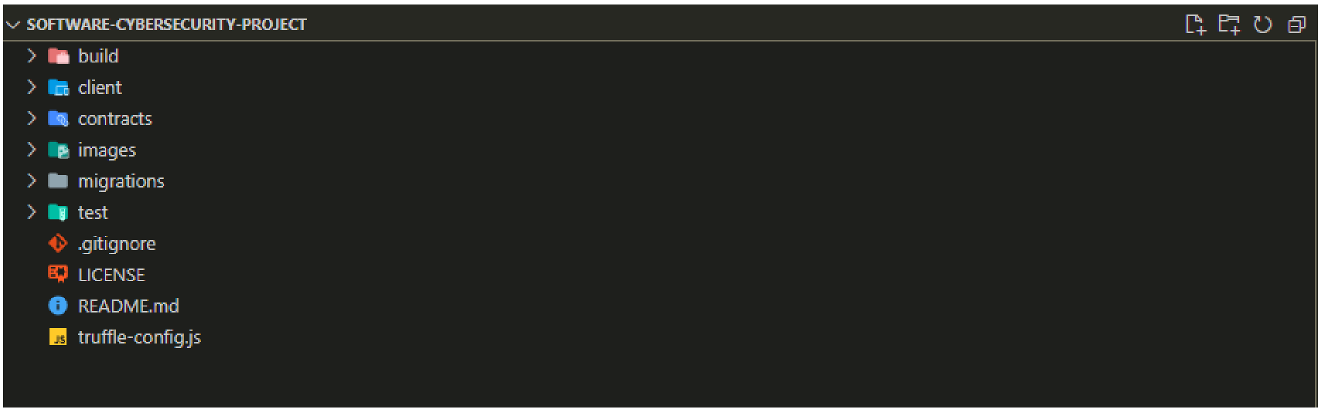
The remaining controls are implemented with development techniques:

- **Input sanitization** (for each field check the validation of the data)
- **Require confirmation** (for each critical action, use dialog to confirm the operation)

6. Implementation

The project was implemented using a GitHub repository which contains different directories:

- **Contracts:** This directory contains smart contracts files written on solidity.
- **Build:** Holds artifacts file generated after solidity compilation of the smart contracts. Each artifact is a json file with the ABI and other info related to the contract.
- **Migrations:** Contains the JavaScript file used by Truffle to deploy smart contracts to Ganache blockchain.
- **Test:** The directory contains a JavaScript file used to test the backend code of the blockchain using truffle command.
- **Client:** The directory holds the web DAPP code developed with React and Web3.



```
SOFTWARE-CYBERSECURITY-PROJECT
  > build
  > client
  > contracts
  > images
  > migrations
  > test
    .gitignore
    LICENSE
    README.md
    truffle-config.js
```

Figure 19 - GitHub repository Structure

6.1. Smart contracts

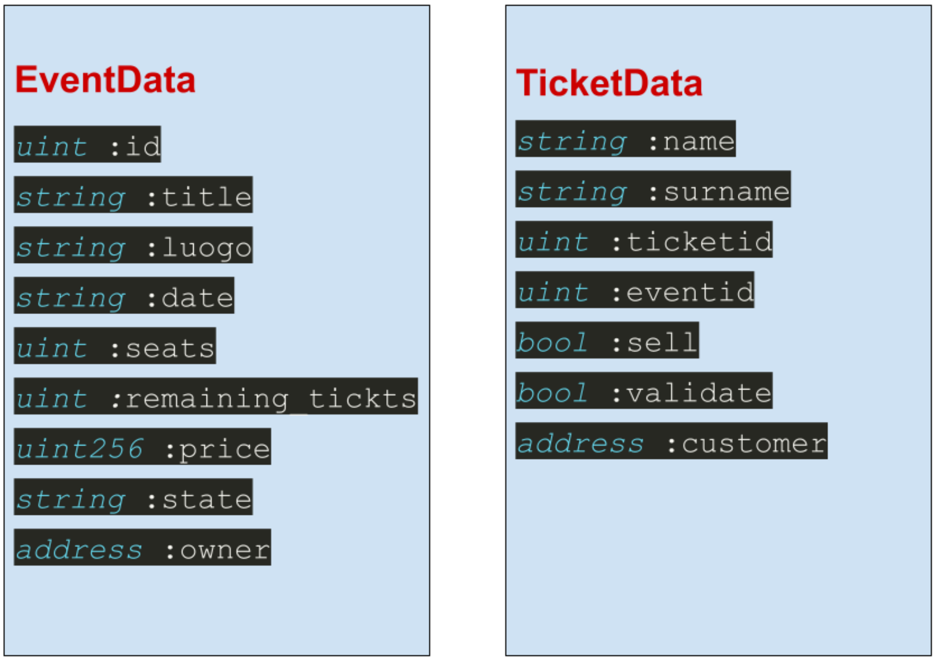
1. Event.sol

The Event contract implements most of the functionalities of the backend. This contract defines the data structures for the event and ticket data and will be deployed by the event manager. The event data structure holds some information about title, place, date, price, seats of the event that needs to be created and other info for his representation (like the state of the event or the address of event creator), while ticket data structure holds information about event id, name and surname of the customer who will purchase the ticket and other useful info (like customer address and the state of the ticket's validation and sale). The contract maintains two arrays of this data structures.

The contract defines different methods both for get useful data like array lengths, array values (like all event created, tickets bought by a certain address or tickets that needs to be validated) and for implements functionalities like create a new event, buy a new ticket for a certain event, validate a purchased ticket or the modification of existing event or ticket. There are also different modifiers defined to prevent certain methods to be executable by unauthorized entities. The modifiers are:

- **Only_owner:** Allows only the contract deployer (who'll be the event manager) to create, finish, invalidate events or to change validator/reseller address.
- **Only_validator:** Allows only defined validator to validate sold ticket.
- **Only_reseller:** Allows only defined reseller to generate ticket of a certain event.

In the end the contract contains different emitted event which will be used to register different actions and calls of the methods on log files.



EventData

```
uint :id
string :title
string :luogo
string :date
uint :seats
uint :remaining_tickts
uint256 :price
string :state
address :owner
```

TicketData

```
string :name
string :surname
uint :ticketid
uint :eventid
bool :sell
bool :validate
address :customer
```

Figure 20 - Event and Ticket data structures

2. Ticket.sol

The ticket smart contract is used to call the buy ticket and validate ticket methods of the event contract. To create this connection between ticket and event smart contracts the constructor needs the event contract address. So, ticket smart contract will be deployed after event deployment because it needs its address. This contract will be deployed by the ticket validator and holds two important methods:

- **Buy_ticket:** This method is public and takes as input customer information like name, surname, and the *event_id* which the customer wants to participate. Then it calls the relative event contract method in order to modify a ticket with customer information and set sell state at true value (only after various checks about event existence/state and ticket availability).

```
function buy_ticket(uint eventid, string memory nome, string memory cognome) public
payable returns(uint){
        Event ev= Event(eventAddress);
        bool control=ev.check_ticket(eventid);
        require(control==true, "Biglietti finiti o evento concluso");
        address customer=msg.sender;
        uint biglietto=ev.buy_ticket{value: msg.value}(payable(customer),eventid,
            nome, cognome);
        emit TicketBought(eventid, biglietto,customer, nome, cognome);
        return biglietto;
    }
```

Figure 21- Buy Ticket method code

- **Validate_ticket:** This method is only accessible by the ticket validator thanks a modifier similar at the *only_owner* of the event smart contract. It takes as input the ticket id and calls the relative event contract method in order to modify the specificized ticket state of validation at true value.

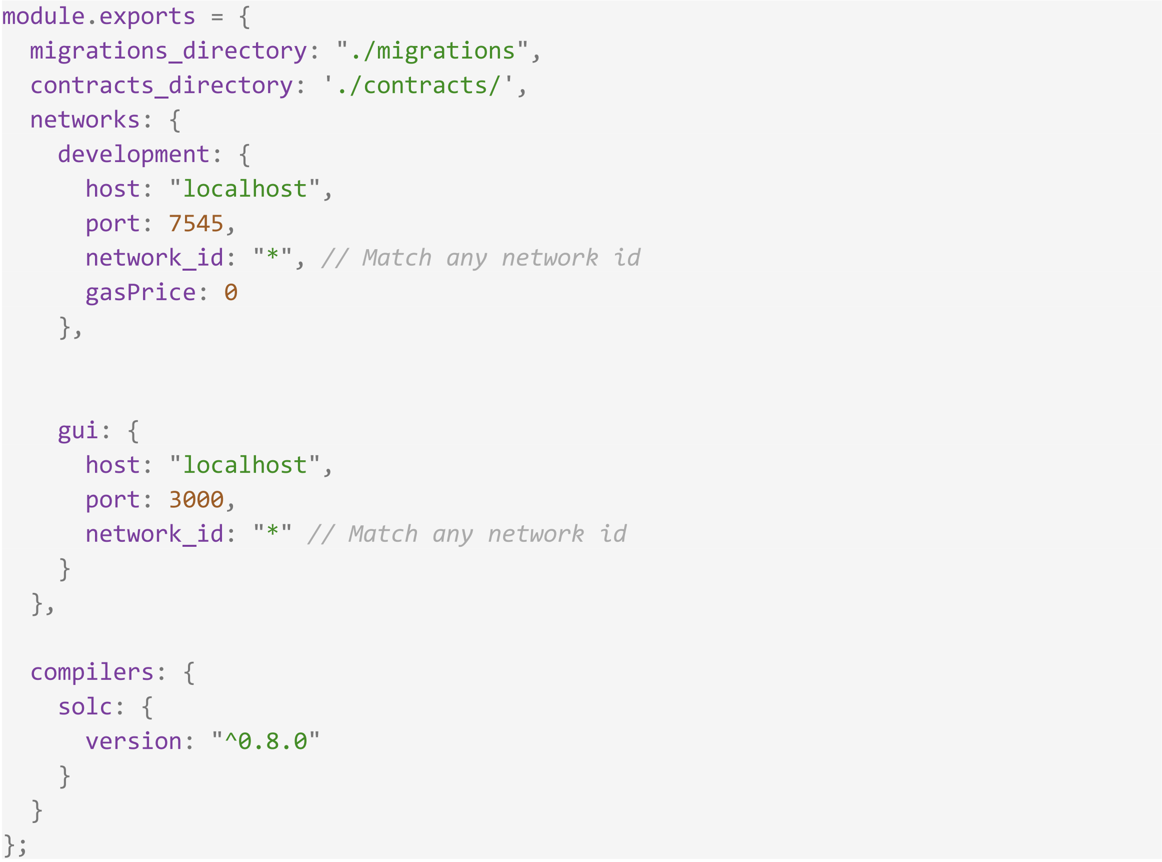
```
function validate_ticket(uint ticketid) public only_owner returns(bool) {
        Event ev= Event(eventAddress);
        bool flag= ev.validate_ticket(ticketid, msg.sender);
        emit TicketValidated(ticketid, msg.sender);
        return flag;
 }
```

Figure 22 - Validate Ticket method code

Like event smart contract the ticket contract defines emitted events after successful execution of the two functions that will be used to register different actions on log files.

6.2 Contract Deployment

The smart contracts developed with solidity are automatically compiled and deployed with the truffle suite command *truffle migrate.* This command uses the *truffleconfig.js* file which contains information about compiler version to use, the network where the Ganache blockchain is running and the port where the web Dapp will run. The command using the compiler version specified first compiles the relative contracts defined and generates their artifacts (which contains contract ABI and other useful information).



```
module.exports = {
  migrations_directory: "./migrations",
  contracts_directory: './contracts/',
  networks: {
    development: {
      host: "localhost",
      port: 7545,
      network_id: "*", // Match any network id
      gasPrice: 0
    },


    gui: {
      host: "localhost",
      port: 3000,
      network_id: "*" // Match any network id
    }
  },

  compilers: {
    solc: {
      version: "^0.8.0"
    }
  }
};
```

Figure 23 - TruffleConfig.js

Then it executes the *deploycontract.js* file to deploy contracts; this file takes the artifacts generated after the compilation and connecting with the Ganache blockchain accounts deploys first the event contract, then takes it's address and use it as argument for ticket contract deployment.

```
var Ticket = artifacts.require("Ticket");
var Event = artifacts.require("Event");
module.exports = function(deployer, network, accounts) {
  deployer.deploy(Event, {from: accounts[0]}).then((event) => {
    return deployer.deploy(Ticket , event.address,{from: accounts[1]})
        .then(async () => {
          const eventInstance = await Event.deployed();
          const ticketInstance = await Ticket.deployed();


        console.log('Event contract owner is ', accounts[0]);
        console.log('Ticket contract owner is ', accounts[1]);
        });
    });
}
```

Figure 24- DeployContracts.js

6.3. Test

The truffle suite permits to test the contract deployed using Solidity or a JavaScript approach. In the project we used the JavaScript approach: the *testEvent.test.js* file contains a function which will executes and test most of the methods of the smart contract deployed. The return value of the methods is compared with the expected results using assert statement which visualize in the command line if the test is successful. In the command line the test will also visualize the data structures generated with the methods (like the event and ticket data arrays after the creation of an event or the purchase/validation of a ticket).

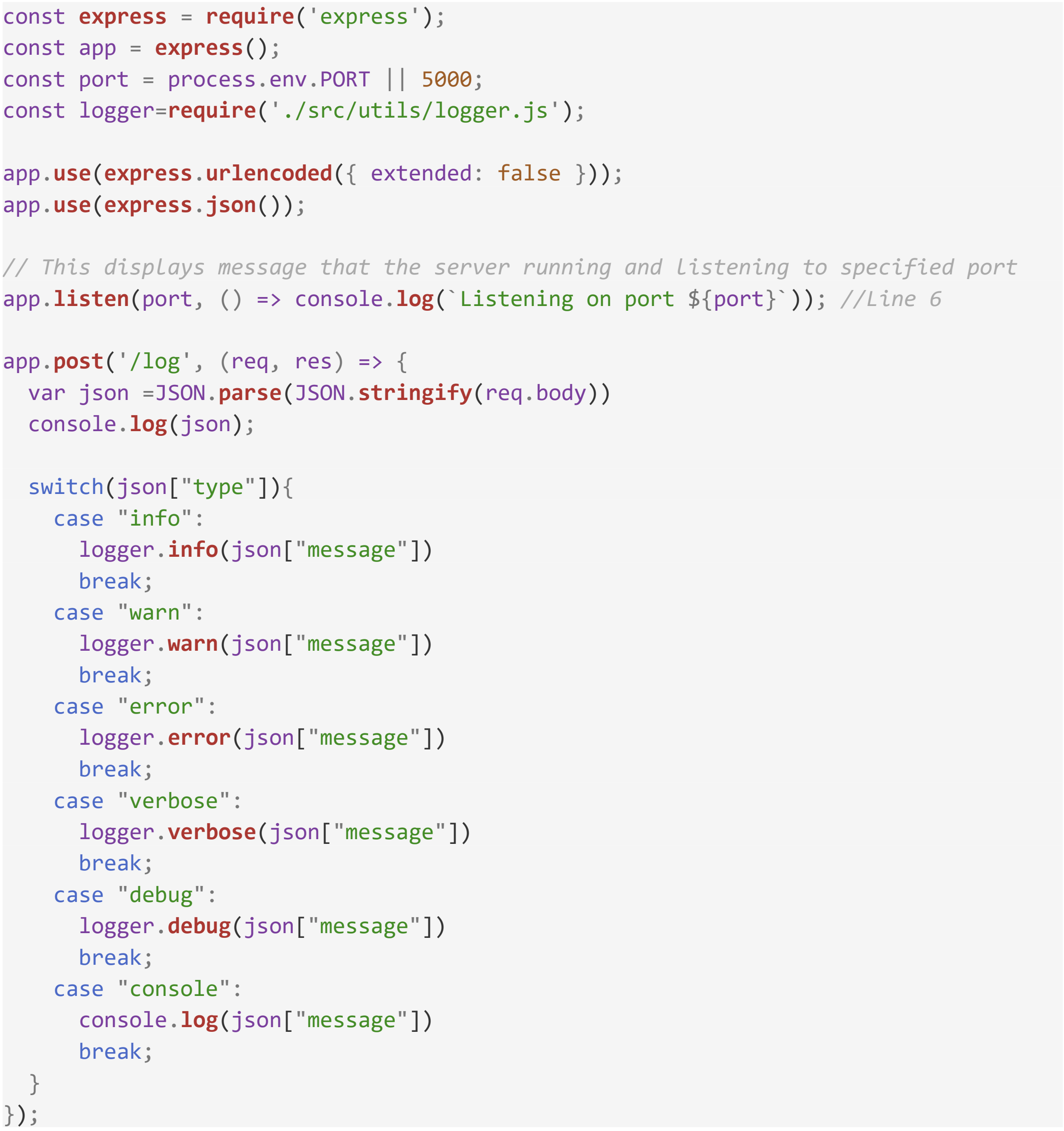
```
contract('Event Ticketing', accounts=> {
    const eventmanager = accounts[0];
    const ticketvalidator = accounts[1];
    const customer = accounts[2];
    let eventInstance;
    let ticketInstance;

    before(async function () {
        eventInstance= await Evento.deployed({from: eventmanager});
        ticketInstance = await Ticket.deployed({from: ticketvalidator});
    });

    it('can create a new event', async function () {
    // using .call() does not persist data, but allows us to get the return value
    // in order to validate that it works properly
      const newEventID = await eventInstance.create_event.call('Concerto',
      'Ancona', '22/02/2021', 5, 2000, accounts[3],accounts[1], {from:
      eventmanager});
      assert.equal(newEventID, 0);

     // Call createEvent normally, where we can't get return value, but the state
     // is saved to the blockchain
      await eventInstance.create_event('Concerto', 'Ancona', '22/02/2021', 5, 80,
      accounts[3], accounts[1],{from: eventmanager});
      const newEvent = await eventInstance.get_events.call();
      console.log(newEvent);

      assert.equal(newEvent[0].title, 'Concerto');
      assert.equal(newEvent[0].luogo, 'Ancona');
      assert.equal(newEvent[0].seats, 5);
    });
}
```

Figure 25 - Example of a test

6.4 Web DAPP

The client directory contains all the Web Dapp code developed. In this directory there are different files and directories used to execute the smart contracts methods using a browser client (in the project we tested only Google Chrome):

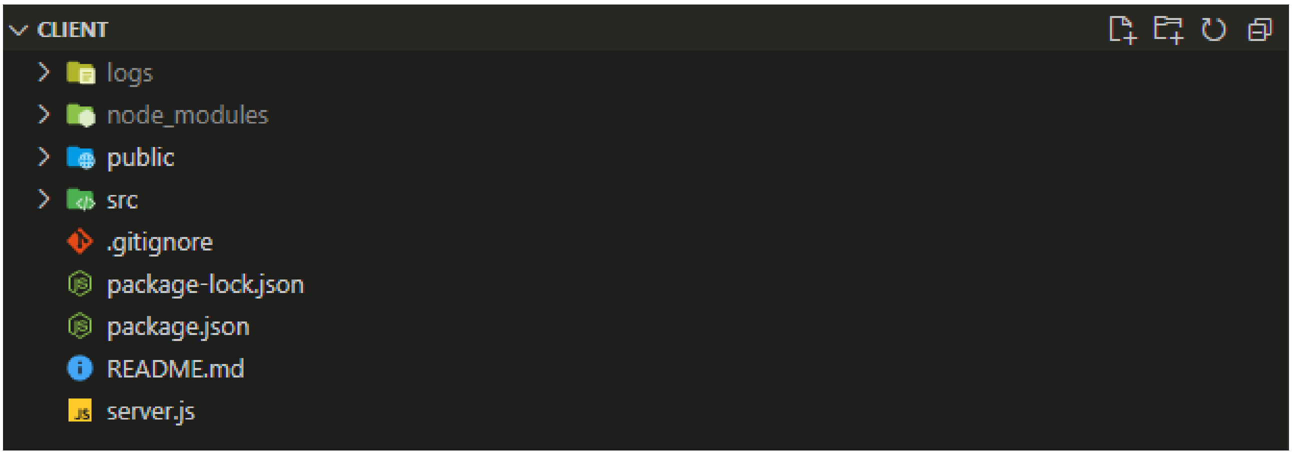
- **Public:** This directory contains the index.html file that will call the script developed for the frontend and other files (like the CSS file and image used).
- **Src:** This directory contains all the scripts and component used for the develop of the frontend.
- **Server.js:** This file creates a backend node at 5000 port using express which will be used to catch the emitted events of the smart contracts and save them in a log file.
  To redirect the events to a file it uses a script described later (logger.js).



```
const express = require('express');
const app = express();
const port = process.env.PORT || 5000;
const logger=require('./src/utils/logger.js');

app.use(express.urlencoded({ extended: false }));
app.use(express.json());

// This displays message that the server running and listening to specified port
app.listen(port, () => console.log(`Listening on port ${port}`)); //Line 6

app.post('/log', (req, res) => {
  var json =JSON.parse(JSON.stringify(req.body))
  console.log(json);

  switch(json["type"]){
    case "info":
      logger.info(json["message"])
      break;
    case "warn":
      logger.warn(json["message"])
      break;
    case "error":
      logger.error(json["message"])
      break;
    case "verbose":
      logger.verbose(json["message"])
      break;
    case "debug":
      logger.debug(json["message"])
      break;
    case "console":
      console.log(json["message"])
      break;
  }
});
```

Figure 26- Server.js

- **Logs:** This directory contains the log files.



```
CLIENT
> logs
> node_modules
> public
> src
  .gitignore
  package-lock.json
  package.json
  README.md
  server.js
```

Figure 27- Client directory structure

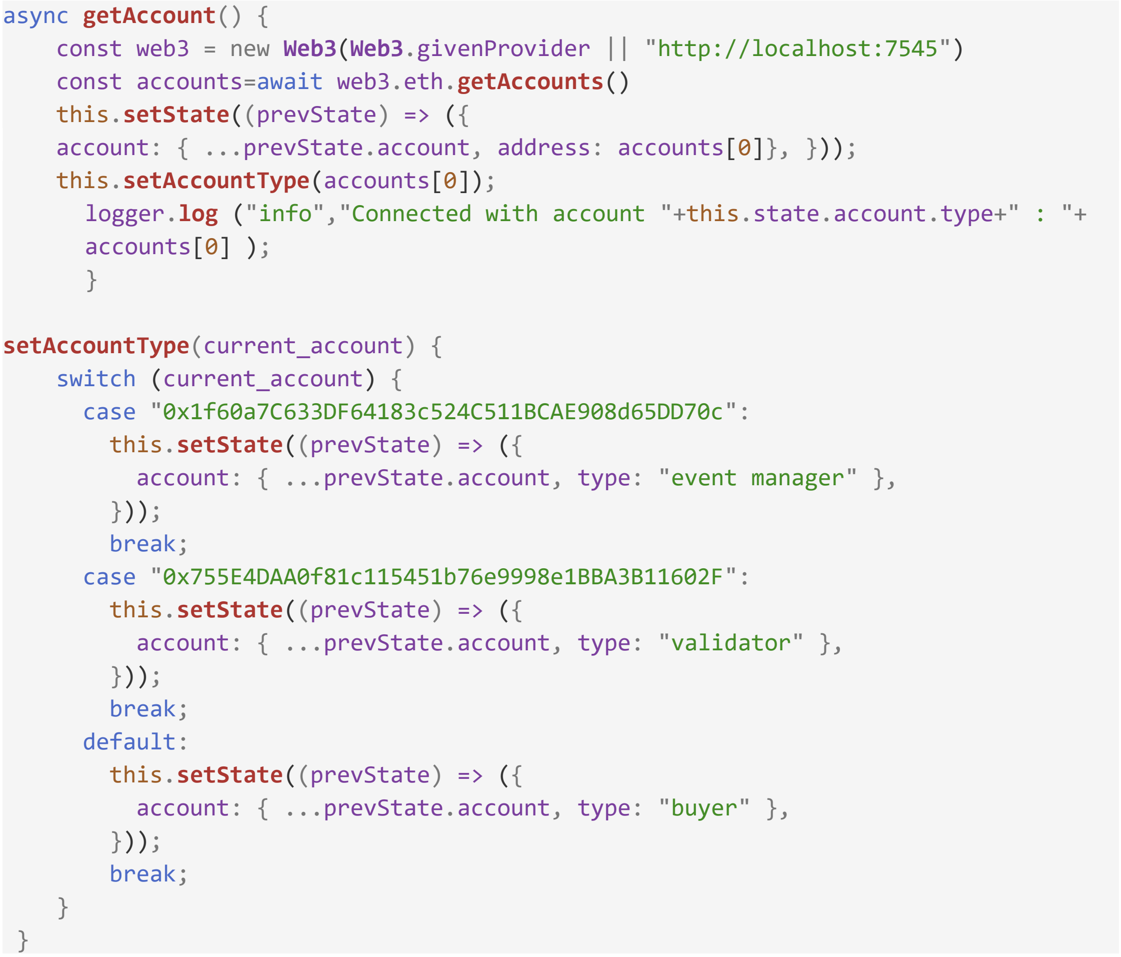
6.4.1 Source

The source directory contains all the scripts used to interact with the smart contract deployed.

1. **App.js**

   The App.js is the main scripts of the frontend. It defines a new Web3 instance and uses React module function Route to redirect at different pages accessible with the nav bar. The script using the web3 functions takes the current Metamask account connected at our WebApp and changes the components visualized at the nav bar. The WebApp recognizes three different types of accounts:

   - **Event Manager:** It's the address associated with the entity who deployed the event smart contract. He can create new events and access admin options (like event ending/invalidation).
   - **Ticket Validator:** It's the address associated with the entity who deployed the ticket smart contract. He can validate purchased tickets.
   - **Buyer:** All other addresses have basic authorization and can only visualize event created, buy a ticket and visualize personal tickets purchased (All these functions are also available for the other two types of accounts).



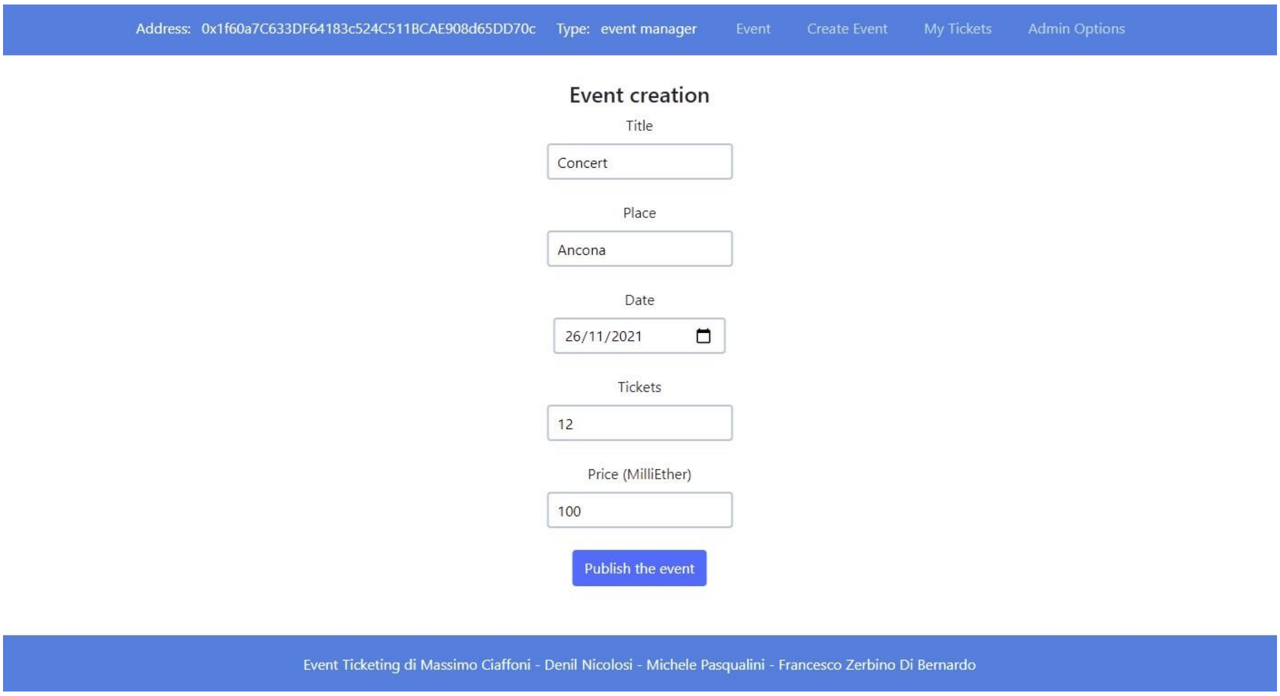
```
async getAccount() {
    const web3 = new Web3(Web3.givenProvider || "http://localhost:7545")
    const accounts=await web3.eth.getAccounts()
    this.setState((prevState) => ({
    account: { ...prevState.account, address: accounts[0]}, }));
    this.setAccountType(accounts[0]);
      logger.log ("info","Connected with account "+this.state.account.type+" : "+
      accounts[0] );
      }

setAccountType(current_account) {
    switch (current_account) {
      case "0x1f60a7C633DF64183c524C511BCAE908d65DD70c":
        this.setState((prevState) => ({
          account: { ...prevState.account, type: "event manager" },
        }));
        break;
      case "0x755E4DAA0f81c115451b76e9998e1BBA3B11602F":
        this.setState((prevState) => ({
          account: { ...prevState.account, type: "validator" },
        }));
        break;
      default:
        this.setState((prevState) => ({
          account: { ...prevState.account, type: "buyer" },
        }));
        break;
    }
 }
```

Figure 28- Account get and type definition

2. Components

The components directory contains all the scripts which uses the smart contract methods:

- **CreateEvent.jsx**: This script interacts with the event smart contract to create a new event. The component is only accessible to the event manager and render a simple form where the manager can specify info about the event (title, place, date, seats, and price) which will be used to create a new event in the blockchain. The code also prevents (in case an unauthorized account success to access the create event page) event creation from unauthorized entities reverting the transaction.



Address: 0x1f60a7C633DF64183c524C511BCAE908d65DD70c Type: event manager

Event creation

Title: Concert

Place: Ancona

Date: 26/11/2021

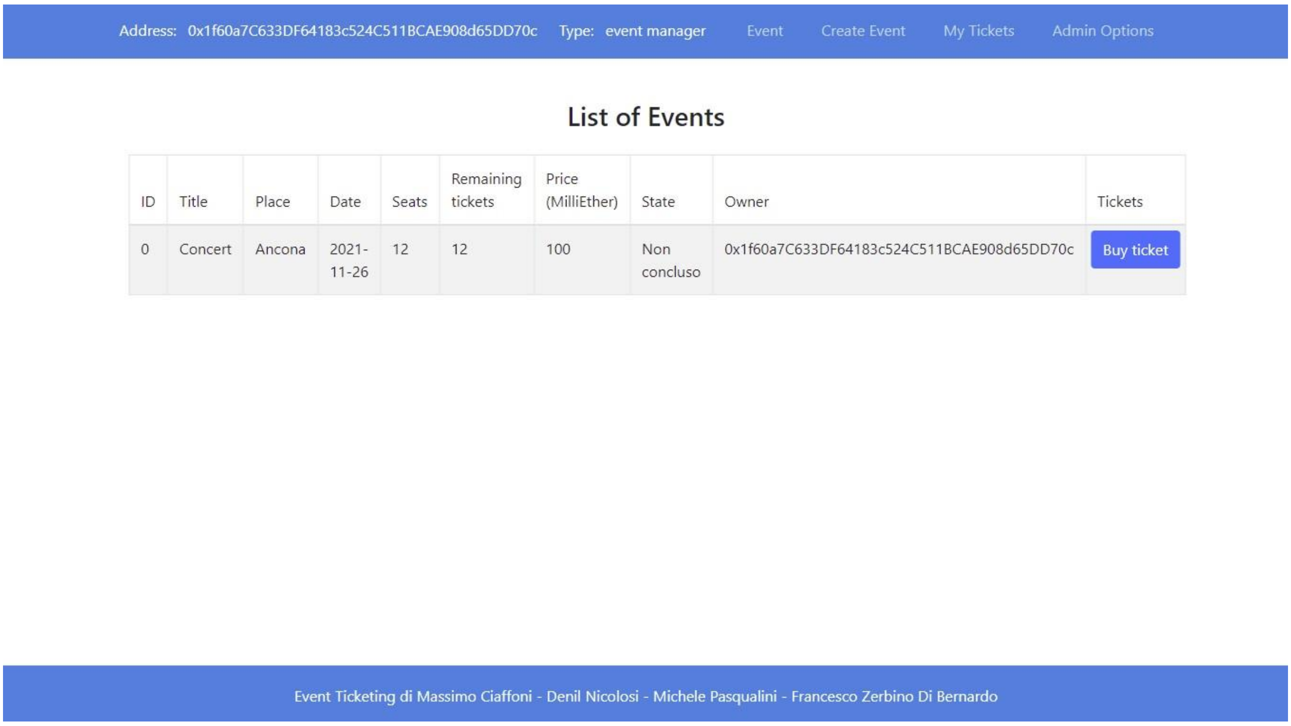
Tickets: 12

Price (MilliEther): 100

Event Ticketing di Massimo Ciaffoni - Denil Nicolosi - Michele Pasqualini - Francesco Zerbino Di Bernardo

Figure 29 - Create Event page

- **GetEvent.jsx:** This script interacts with the event smart contract callin the get_events method which will return all the information about event created and visualizes them in a dynamic table. Here the user can buy the ticket of the choosed event.



Address: 0x1f60a7C633DF64183c524C511BCAE908d65DD70c Type: event manager

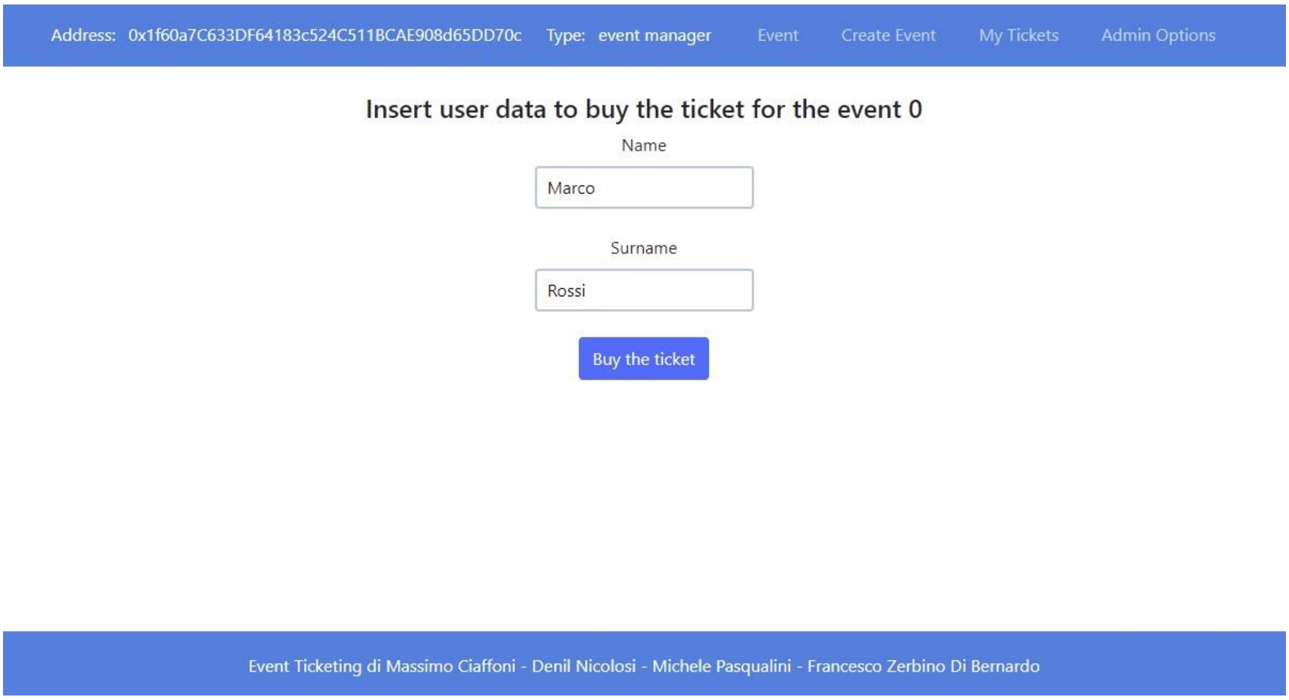
List of Events

ID	Title	Place	Date	Seats	Remaining tickets	Price (MilliEther)	State	Owner	Tickets
0	Concert	Ancona	2021-11-26	12	12	100	Non concluso	0x1f60a7C633DF64183c524C511BCAE908d65DD70c	Buy ticket

Event Ticketing di Massimo Ciaffoni - Denil Nicolosi - Michele Pasqualini - Francesco Zerbino Di Bernardo

Figure 30 - Get Event page

The buy ticket button redirect to another page that uses the InsertUserData.jsx component. This script takes the price of the relative event selected and render a form where the customer can specify his name and surname and then buy a ticket interacting with the ticket smart contract relative method.



Address: 0x1f60a7C633DF64183c524C511BCAE908d65DD70c Type: event manager

Insert user data to buy the ticket for the event 0

Name: Marco

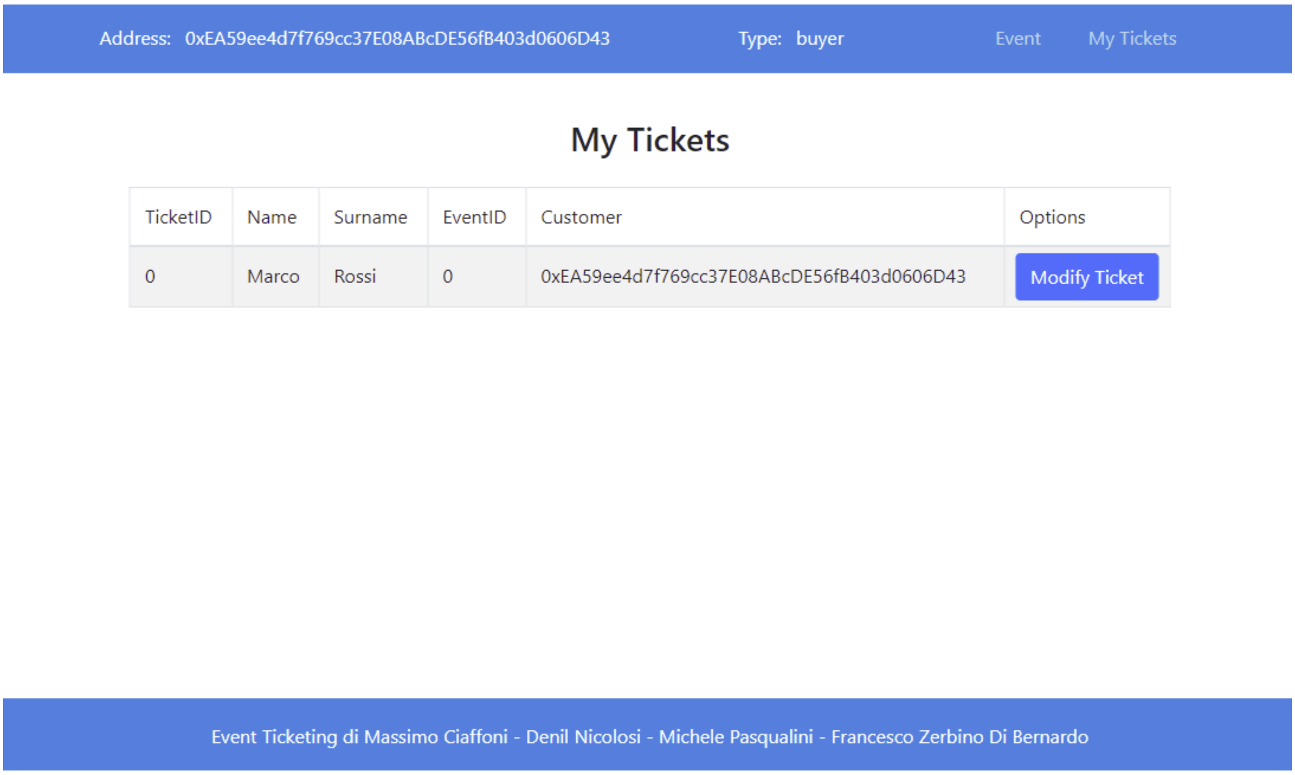
Surname: Rossi

Event Ticketing di Massimo Ciaffoni - Denil Nicolosi - Michele Pasqualini - Francesco Zerbino Di Bernardo

Figure 31 - Buy Ticket page

The button is accessible only when the ticket is available, or the event isn't finished or invalidated.

- **GetTicket.jsx:** This component interacts with the event smart contract calling get_personal_tickets method which returns only the tickets owned by a particular address specified whom will be the current Metamask account connect at the WebApp. The tickets are visualizable in a dynamic table. Here the user can modify his ticket.


Address: 0xEA59ee4d7f769cc37E08ABcDE56fB403d0606D43 Type: buyer

My Tickets

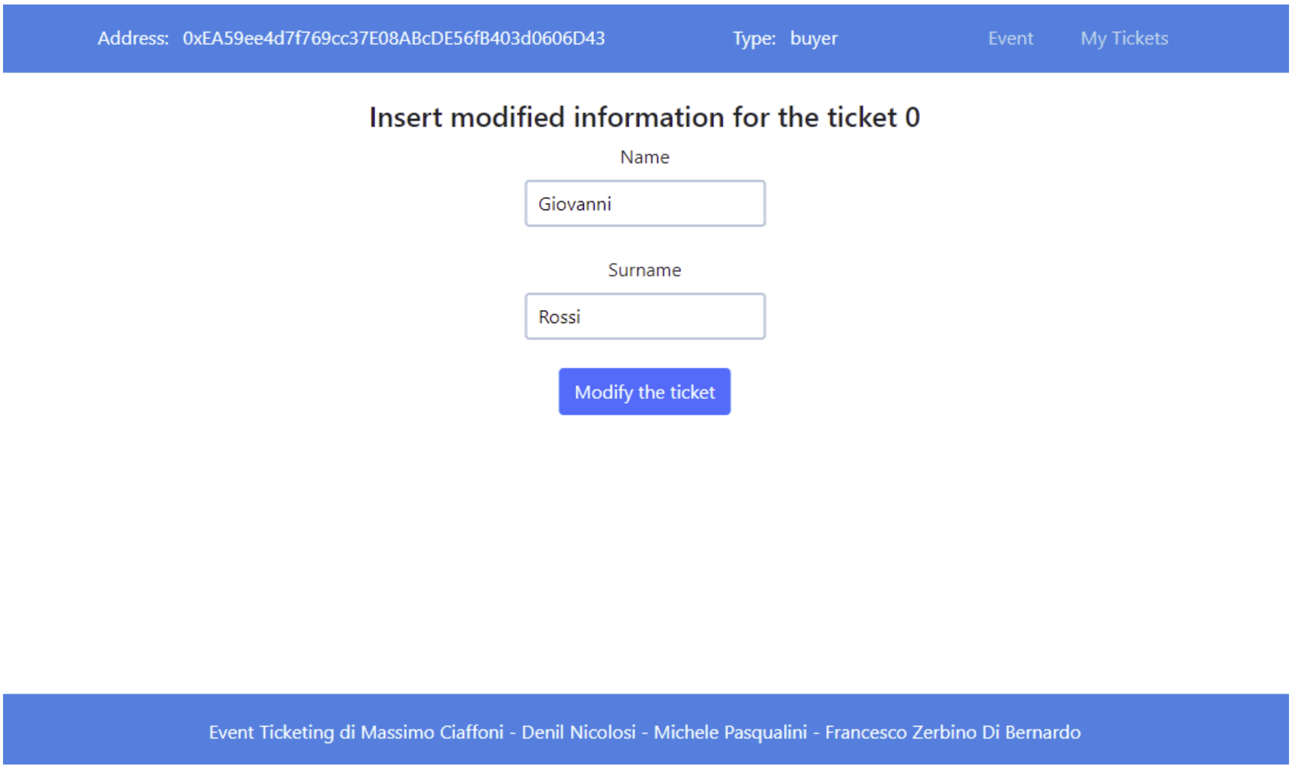
TicketID	Name	Surname	EventID	Customer	Options
0	Marco	Rossi	0	0xEA59ee4d7f769cc37E08ABcDE56fB403d0606D43	Modify Ticket

Event Ticketing di Massimo Ciaffoni - Denil Nicolosi - Michele Pasqualini - Francesco Zerbino Di Bernardo

Figure 32 - Tickets page

The "modify ticket" button redirect to another page that uses the ModifyTicket.jsx component.

- **ModifyTicket.jsx:** This script renders a form where the customer can modify his name and surname. The button "Modify the ticket" allows you to make and modify informations only if in accordance with the specified criteria.


Address: 0xEA59ee4d7f769cc37E08ABcDE56fB403d0606D43 Type: buyer

Insert modified information for the ticket 0

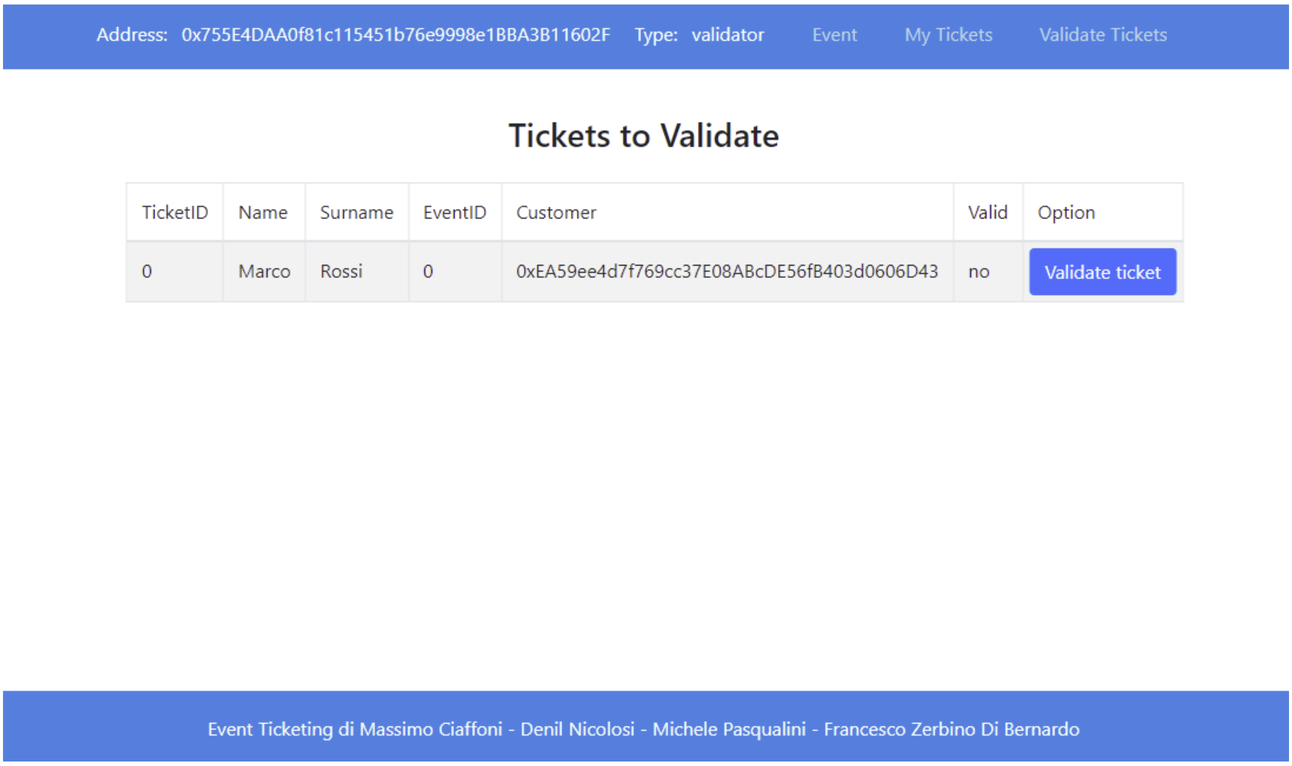
Name: Giovanni

Surname: Rossi

Event Ticketing di Massimo Ciaffoni - Denil Nicolosi - Michele Pasqualini - Francesco Zerbino Di Bernardo

Figure 33 – Modification ticket page

- **ValidateTicket.jsx:** This component is like the previous one, but it renders all the tickets purchased by a customer (tickets with the sell state value at true). The table also contains a button which will interact with the ticket smart contract's method used to validate tickets. This button is accessible only if the ticket isn't yet validated. Furthermore, like the create event component the page is only accessible by the ticket validator, but it still prevents ticket validation by unauthorized entities.



Address: 0x755E4DAA0f81c115451b76e9998e1BBA3B11602F Type: validator
Event
My Tickets
Validate Tickets

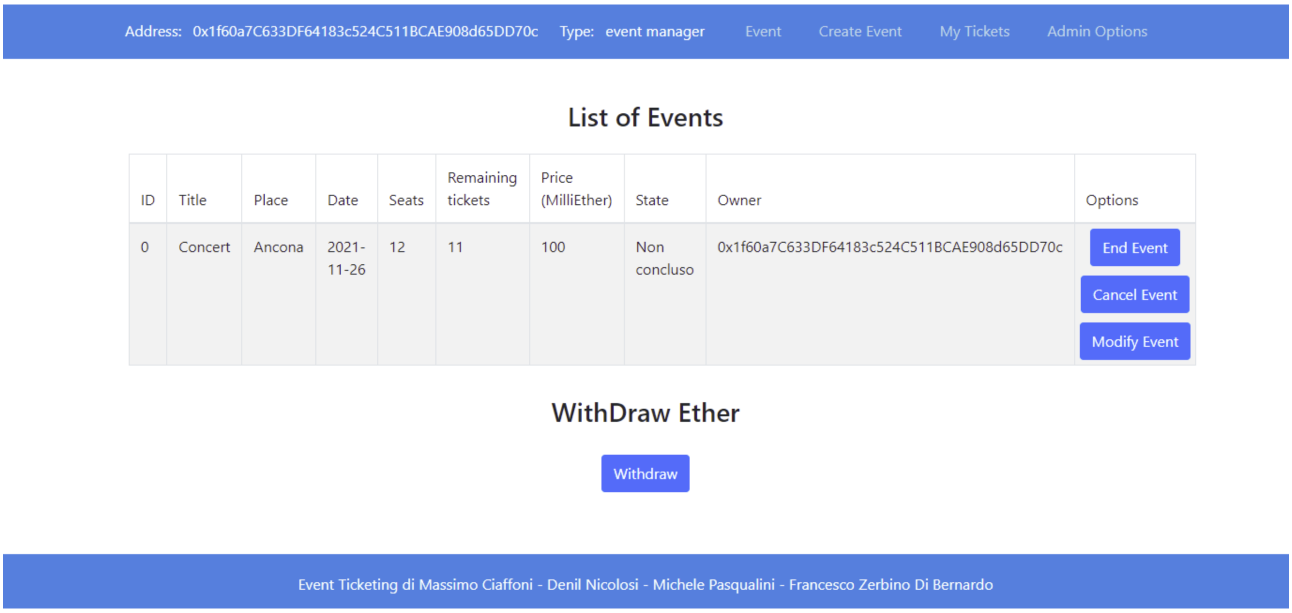
Tickets to Validate

TicketID	Name	Surname	EventID	Customer	Valid	Option
0	Marco	Rossi	0	0xEA59ee4d7f769cc37E08ABcDE56fB403d0606D43	no	Validate ticket

Event Ticketing di Massimo Ciaffoni - Denil Nicolosi - Michele Pasqualini - Francesco Zerbino Di Bernardo

Figure 34 - Validation page

- **Admin.jsx:** This script interacts with the event smart contract, and it's used by the event manager to finish or invalidate an event. It renders a dynamic table with all the events using the same function used in the GetEvent component. For each event in the table there are three buttons used for invalidate/finishing or modify the relative event. These buttons are accessible only if the event is still in the not ended state. Like all other sensitive pages, the code prevents method execution by unauthorized entities. The page also renders a button which can be used by the event manager to withdraw ether transferred into event smart contract during buy ticket phase (the button is available to transfer ether only if contract balance is not zero). When the event manager chooses to modify event information using the relative button, he'll be redirected to another page which uses the ModiftEvent.jsx component.



Address: 0x1f60a7C633DF64183c524C511BCAE908d65DD70c Type: event manager

List of Events

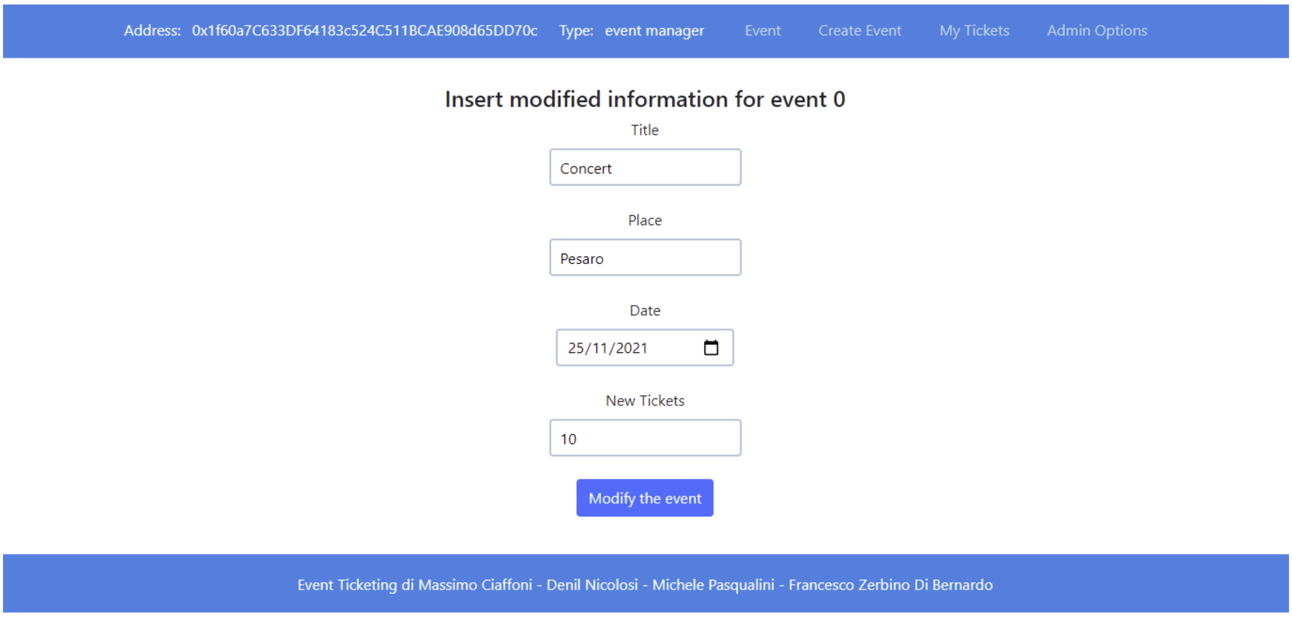
ID	Title	Place	Date	Seats	Remaining tickets	Price (MilliEther)	State	Owner	Options
0	Concert	Ancona	2021-11-26	12	11	100	Non concluso	0x1f60a7C633DF64183c524C511BCAE908d65DD70c	End Event Cancel Event Modify Event

WithDraw Ether

Event Ticketing di Massimo Ciaffoni - Denil Nicolosi - Michele Pasqualini - Francesco Zerbino Di Bernardo

Figure 35 - Admin Page

- **ModifiyEvent.jsx:** This script implements the editing functionality of a created event. The event manager can modify or update the event parameters, can be changed Title, Place and Date with the possibility of adding more seats. The form for date, allows the user to choose a date after the current day. The button "Modify the event" allows you to make and modify informations only if in accordance with the specified criteria.



Address: 0x1f60a7C633DF64183c524C511BCAE908d65DD70c Type: event manager

Insert modified information for event 0

Title: Concert

Place: Pesaro

Date: 25/11/2021

New Tickets: 10

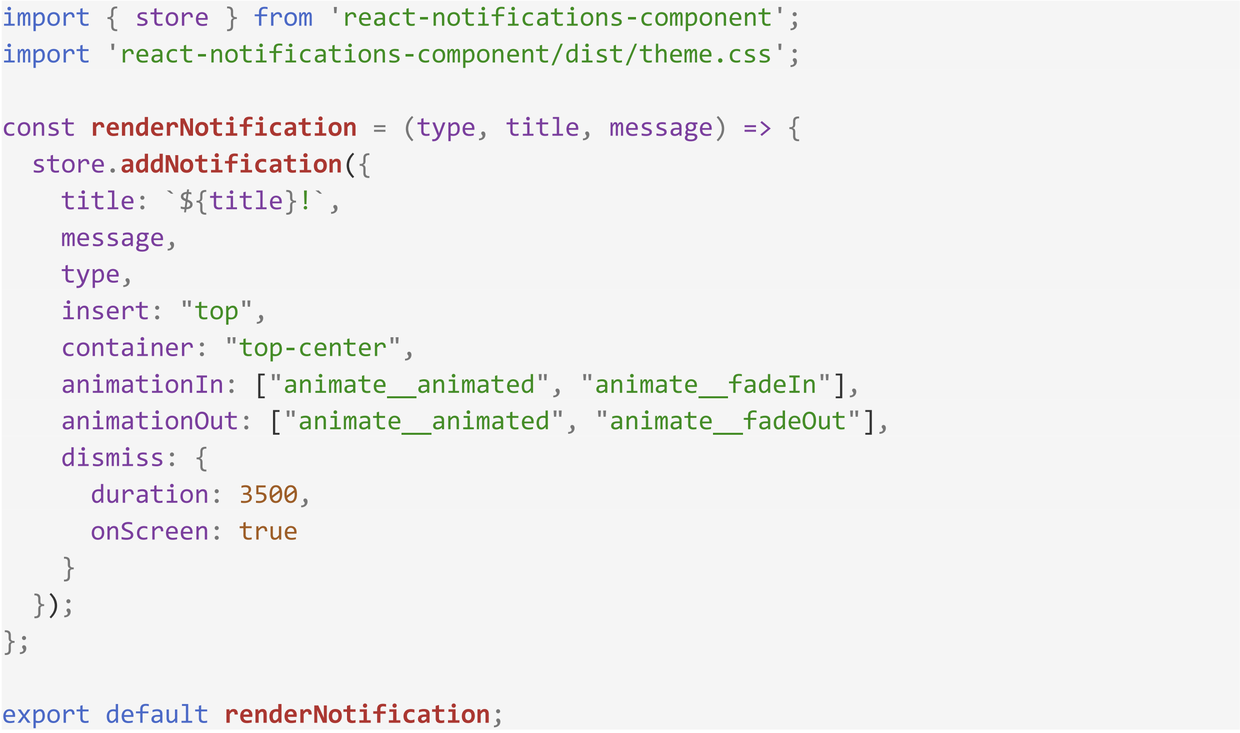
Event Ticketing di Massimo Ciaffoni - Denil Nicolosi - Michele Pasqualini - Francesco Zerbino Di Bernardo

Figure 36 – ModifyEvent.jsx

3. Utils

This directory holds some JavaScript files used in the components:

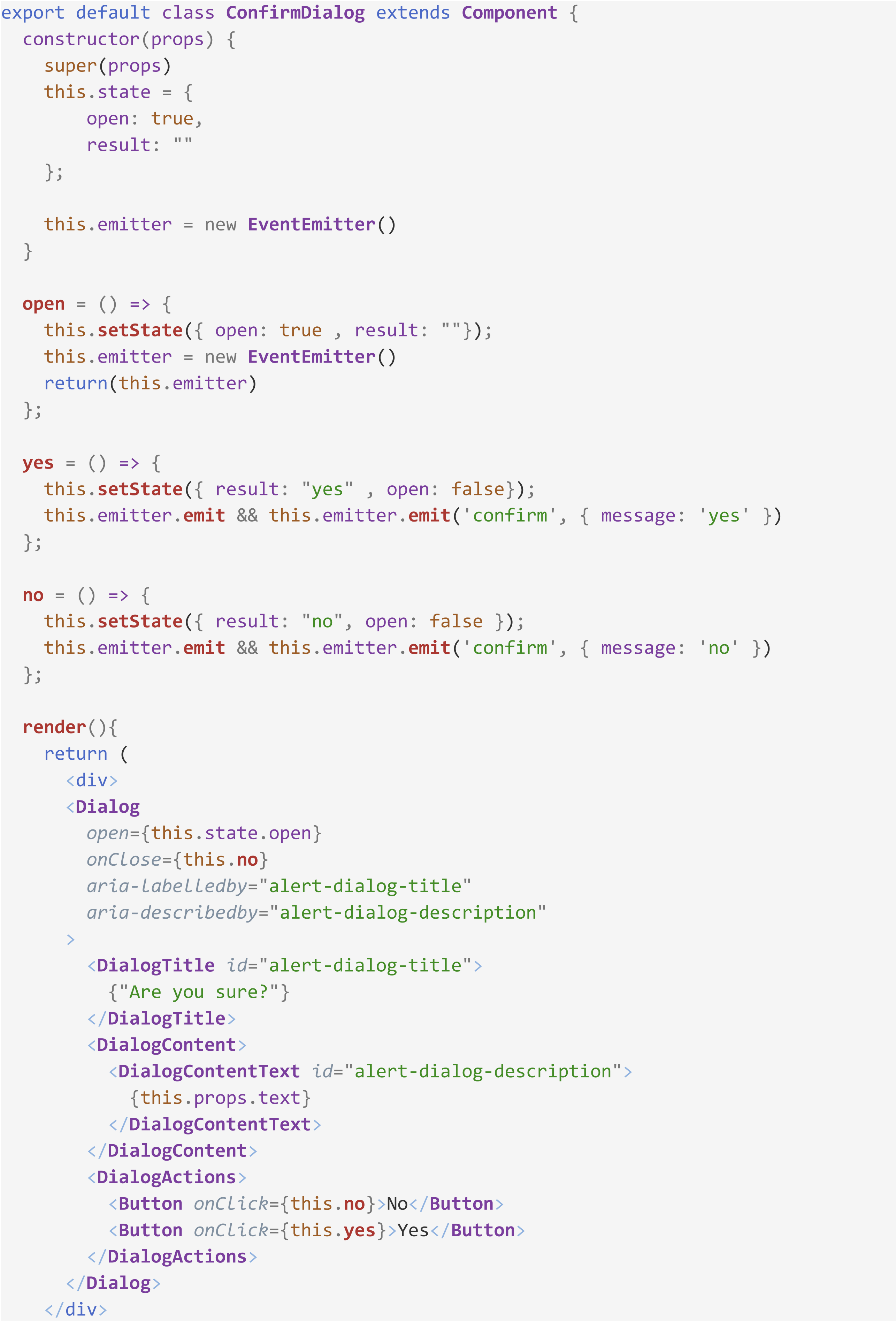
- **NotificationHandler.js**: This script defines a notification created with React components that will be used in the transaction confirmation. If a particular transaction succeeds it will generate a notification coloured in green with a particular message, if instead the transaction fail or is rejected by the user it will generate a notification coloured in red with message like "User unauthorized" or "User denied transaction".



```
import { store } from 'react-notifications-component';
import 'react-notifications-component/dist/theme.css';

const renderNotification = (type, title, message) => {
  store.addNotification({
    title: `${title}!`,
    message,
    type,
    insert: "top",
    container: "top-center",
    animationIn: ["animate__animated", "animate__fadeIn"],
    animationOut: ["animate__animated", "animate__fadeOut"],
    dismiss: {
      duration: 3500,
      onScreen: true
    }
  });
};

export default renderNotification;
```

Figure 37 - NotificationHandler.js

- **ConfirmDialog.jsx:** This script define a pop-up dialog where the user can confirm or refuse the operation selected. Using this, we can prevent accidental operation made by clumsy user. This component is generic and it can use everywhere we need it. Indeed, when we call it we can set the display message. When the user click on “Yes” or “No” the script emit two different event that can be catch from the component that called it.



```jsx
export default class ConfirmDialog extends Component {
  constructor(props) {
    super(props)
    this.state = {
        open: true,
        result: ""
    };

    this.emitter = new EventEmitter()
  }

  open = () => {
    this.setState({ open: true , result: ""});
    this.emitter = new EventEmitter()
    return(this.emitter)
  };

  yes = () => {
    this.setState({ result: "yes" , open: false});
    this.emitter.emit && this.emitter.emit('confirm', { message: 'yes' })
  };

  no = () => {
    this.setState({ result: "no", open: false });
    this.emitter.emit && this.emitter.emit('confirm', { message: 'no' })
  };

  render(){
    return (
      <div>
      <Dialog
        open={this.state.open}
        onClose={this.no}
        aria-labelledby="alert-dialog-title"
        aria-describedby="alert-dialog-description"
      >
        <DialogTitle id="alert-dialog-title">
          {"Are you sure?"}
        </DialogTitle>
        <DialogContent>
          <DialogContentText id="alert-dialog-description">
            {this.props.text}
          </DialogContentText>
        </DialogContent>
        <DialogActions>
          <Button onClick={this.no}>No</Button>
          <Button onClick={this.yes}>Yes</Button>
        </DialogActions>
      </Dialog>
    </div>
```

Figure 38- ConfirmDialog.jsx

- **Logger.js:** This script defines a custom logger using Winston module. This custom logger will be saved in the logs directory and creates or append the existed file. The log format takes for each call the level of the message (info, warning, error), a timestamp and the relative message that we want to record (in a json format). This script is used for record emitted events of the smart contracts, report a user connected to the Dapp, and for record error messages in case of failures. Every message also contains the address of the user who success/fails a determinate smart contract method execution in order to track the different actions undertaken by the users in the WebApp.

```
const { createLogger, format, transports } = require('winston');

module.exports = createLogger({
format: format.json(),
transports:
    new transports.File({
    filename: 'logs/server.log',
    format:format.combine(
        format.timestamp({format: 'MMM-DD-YYYY HH:mm:ss'}),
        format.align(),
        format.printf(info => `${info.level}: ${[info.timestamp]}:
      ${info.message}`),
    )}),
});
```

Figure 39 - Logger.js

- **LogAPI.js:** This script uses the node created with the server.js file to post log information (level and message) at localhost:5000/log which will create/append the log file with the timestamp and the information received. The LogAPI is imported and used in most of the components of the WebApp to post emitted events of the smart contract or the error messages.

```
module.exports = {
    log : async(type, message) => {
        const data = {
          type: type,
          message: message
        };

        await fetch('/log' , {
          method: "POST",
          headers: {
            'Content-type': 'application/json'
          },
          body: JSON.stringify(data),
        })
        .then((response) => response.json())
        .then((result) =>{
          console.log(result)
        })
      }
};
```

Figure 40 - LogAPI

6.5. User Guide

A user guide is available at the project repository. In the guide is described all the passages to test the WebApp with all its functionalities.

7. Conclusion and future work

The realization of this project as part of the Software Cybersecurity course has allowed us to learn and test the potential of blockchain technology. In particular, this first development of the DApp leads us to conclude that it is feasible and quite simple to develop a ticketing system based on this technology. Our solution ensures that the safety requirements analysed in the design phase are not violated. Therefore, some control measures such as filtering, encryption, credential recovery system have not been implemented after an accurate assessment of the feasibility, residual risk and cost. Therefore, developing a blockchain-based ticketing system involves a number of advantages including the ability to track the owner of each ticket, the ability to prevent fake ticket scams and all illegal activities related to the world of ticketing.

We decided to develop the project by adopting Ganache and Metamask for accounts management. One of the possible future developments would be to integrate a database that stores all the different types of accounts and implement a login system, using some encryption techniques to safeguard user's personal information in such a way that security requirements continue to be guaranteed. Another future development of the application could be to support multiple event manager accounts and validator accounts, keeping the system as modular as possible. Finally, some safety measures could be implemented in more detail in order to guarantee a high reliability of the system.

References

[1] Jon Martindale. What is a blockchain [Internett] 08.3.18 [Accessed 28.01.2019] Available from: https://www.digitaltrends.com/computing/what-is-a-blockchain/

[2] Chaonian Guo, Shenglan Ma, Hao Wang, Shuhan Cheng, Tongsen Wang. LoC: Poverty Alleviation Loan Management System based on Smart Contracts. Division of Sci. anf Tech, Department of ICT and Natural Sci. [Accessed 28 January 2019]

[3] Ripple [Internett]. [Accessed 2 January 2019]. Available from: https://ripple.com/

[4] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper.

[5] M. Hearn, Corda - a distributed ledger, corda Technical White Paper.

[6] Hyperledger [Internett]. [Accessed 28. January 2019] Available from: https://www.hyperledger.org/

[7] Toggl [Internett]. [22. January 2019] Available from: https://toggl.com/

[8] Bitbucket [Internett]. [28. January 2019] Available from: https://bitbucket.org