UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di un'infrastruttura per la compravendita e la lavorazione di materie prime - EcoChain

Professore:
Luca Spalazzi

Gruppo 1:
David Ceka
Ciro Maccarone
Melissa Proietti
Riccardo Schiavoni
Davide Tomo

Anno Accademico 2021-2022

Indice

Capitolo 1

Introduzione

La seguente relazione è stata scritta al fine di mostrare le fasi di progettazione ed implementazione di un software distribuito necessario a tracciare le emissioni generate dai processi di produzione dei prodotti alimentari. Il tutto deve essere realizzato attraverso l'utilizzo di una blockchain e dei relativi smart-contracts.

E' stata suddivisa nelle seguenti fasi:

- **Ingegneria dei requisiti**: viene trattata l'analisi dei requisiti, creando un diagramma I* e individuando gli *attori* principali che si interfacciano con il sistema, gli *asset* coinvolti, le relative *minacce*, i relativi *attacchi* ed i meccanismi di difesa ad essi associati. Inoltre, attraverso l'utilizzo dello *schema di Jacobson* verranno catalogate tutte le possibili azioni che possono effettuare gli attori: *Use Case*, *Abuse Case* e *Misuse Case*.
- **Implementazione**: viene mostrata l'intera fase di implementazione del codice analizzando le scelte tecnologiche effettuate e le linee guida seguite per la progettazione di un software sicuro.
- **Considerazioni finali**

Capitolo 2

Ingegneria dei requisiti

2.1 Analisi dei requisiti

L'analisi dei requisiti si sviluppa in due fasi principali: *Early Requirement Analysis* e *Late Requirement Analysis*.

2.2 Early Requirement Analysis

In questa fase, caratterizzata da un alto livello di astrazione, vengono discusse le linee guida da seguire.

Nel caso di studio preso in considerazione si deve realizzare un software che sia in grado di tracciare le emissioni di CO_2 generate durante le varie fasi di lavorazione dell'intera filiera. L'obiettivo principale sarà quello di tenere traccia di tali emissioni in una struttura che ne impedisca la modifica.

Le interazioni degli attori e/o con il sistema avvengono tramite un'interfaccia web. E' previsto, inoltre, l'utilizzo di smart-contracts e quindi di una relativa blockchain per eseguire alcune delle operazioni fondamentali richieste, come la registrazione dell'impatto ambientale prodotto durante tutte le fasi di lavorazione delle materie prime o la possibilità di visionare l'impatto ambientale totale del prodotto finito, durante la fase di acquisto, da parte del cliente. In fase di acquisto il software produrrà un **NFT** che andrà a certificare il totale delle emissioni generate durante la lavorazione dei prodotti.

2.3 Late Requirement Analysis

In questa fase è stato utilizzato il modello I^* che ci permette di individuare i principali attori, le risorse, i task, i goal ed i softgoal che devono essere portati a termine.

Dopo un'attenta analisi sono stati individuati quattro attori principali:

- **Fornitore**: chi rifornisce il mercato di materie prime
- **Produttore**: chi, attraverso lavorazioni sulle materie prime, genera prodotti finiti
- **Cliente**: chi acquista i prodotti per la consumazione
- **Software**

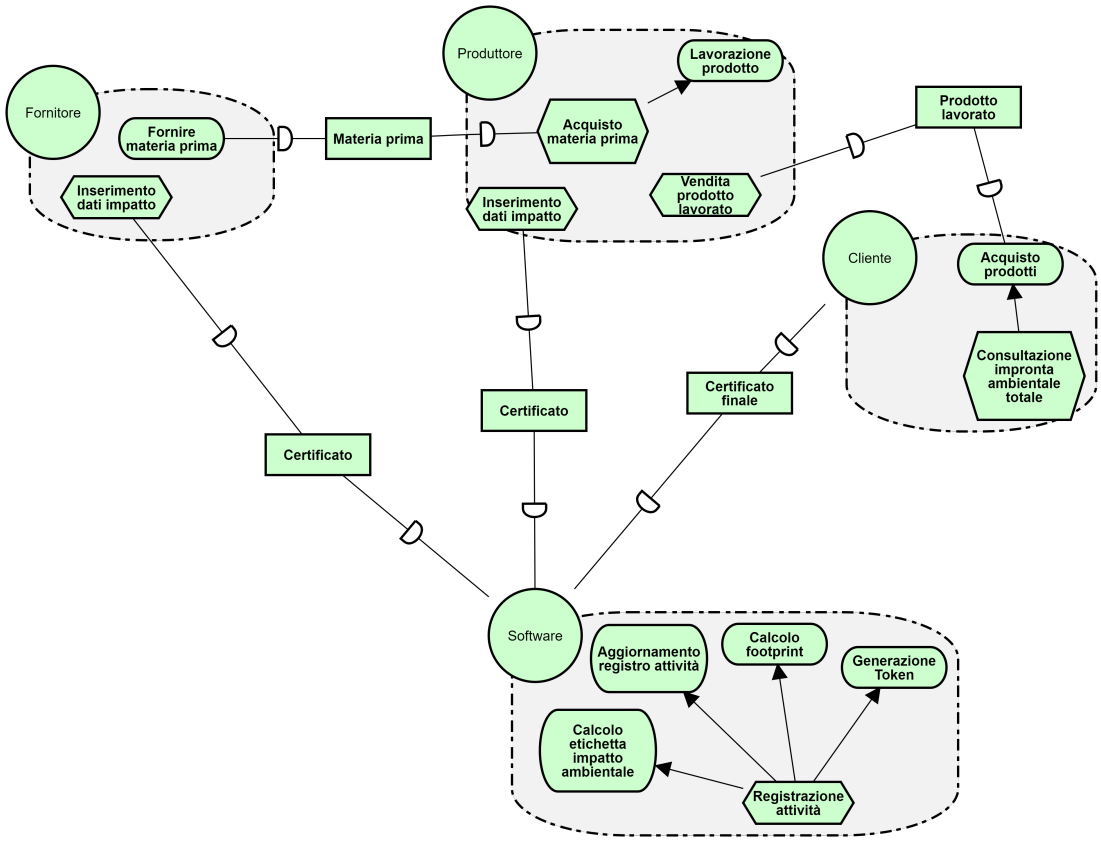

Figura 2.1: Modello I*

2.3.1 Fornitore

Lo scopo principale del fornitore è quello di procurare, a chi è interessato all'acquisto, materie prime con emissioni certificate. Al fornitore è stato assegnato un *task* che permette l'inserimento dei dati relativi alle materie prime all'interno del sistema. Come risposta a tale azione verrà prodotto un certificato che sarà gestito dal software.

2.3.2 Produttore

Il produttore viene identificato come colui che lavorerà le materie prime al fine di ottenere un prodotto finito e pronto per essere messo in vendita. Per questo motivo è stata realizzata una funzione che permette al Produttore di acquisire le informazioni relative alle materie prime, tramite l'utilizzo del software. Anche al Produttore è stato assegnato lo stesso *task* relativo all'inserimento dei dati dei prodotti lavorati e, proprio come accade per il fornitore, verrà generato un certificato che verrà gestito dal software.

2.3.3 Cliente

Il cliente ha la possibilità di acquistare i prodotti lavorati, consultando il valore del footprint finale associato ad ogni prodotto.

2.3.4 Software

L'obiettivo del software è quello di generare i certificati relativi ai prodotti nel momento in cui il cliente si appresta a comprarne uno. Per questo motivo è stato identificato un *task* di registrazione delle attività che consente di manipolare i dati relativi ai prodotti e attraverso cui verrà ricavato il valore della *Carbon Footprint* finale relativo ad ogni prodotto.

2.4 Identificazione degli asset

In questa fase vengono identificati gli **asset**, ovvero quegli elementi del software che è fondamentale proteggere da possibili attacchi. Ad ogni asset viene assegnato un valore e viene individuato un valore relativo al livello di esposizione.
A tale scopo sono state utilizzate due scale di **Likert**:

- Scala di Likert per l'assegnazione di un **valore** all'**asset**: scala da 1 a 3, in cui 1 rappresenta il valore minimo e 3 rappresenta il valore massimo.
- Scala di Likert per l'assegnazione di **valore** relativo al **livello** di **exposure** dell'asset: scala da 1 a 5 in cui 1 rappresenta la pericolosità minima e 5 rappresenta la pericolosità massima.

Per ogni asset sono state individuate anche le **Security Policies** che devono essere rispettate. In particolar modo sono state prese in considerazione le seguenti policies:

- CIA
  - **Confidentiality**: le informazioni possono essere accedute solo da coloro che sono autorizzati;
  - **Integrity**: l'informazione non deve essere alterata/distrutta;

  - **Availability**: l'informazione deve essere disponibile quando serve.

- AAA
  - **Authenticity**: garantire che una determinata entità sia genuina;
  - **Assurance**: garantire che una determinata entità si comporti come ci si aspetti;
  - **Accountability**: essere in grado di attribuire la responsabilità di un'azione ad una determinata entità.

- SRR
  - **Safety**: capacità di operare senza guasti;
  - **Reliability**: garantire che i servizi del sistema siano disponibili nella maniera specificata;
  - **Resilience**: capacità di resistere ad eventi dannosi ed effettuare un ripristino da essi.

Asset	Value	Objective	Exposure
Generazione Token Carbon Footprint	3	Integrità, Autenticità e disponibilità	Autenticità: il cliente consumatore deve avere fiducia nella veridicità dei dati presenti nella footprint calcolata, diversamente potrebbe optare per un prodotto concorrente. (4) Integrità: i dati vanno preservati da attacchi informatici esterni o da errori accidentali, il token potrebbe risultare non valido e impedire il corretto completamento di una transazione o potrebbe essere fuorviante al momento dell'acquisto (5) Non Ripudiabilità: il meccanismo della blockchain deve garantire con certezza la responsabilità di ogni attore nelle modifiche e nella generazione del token. Deve essere sempre possibile per il cliente verificare che il token sia stato effettivamente generato da produttore e fornitore, che non possono non riconoscerlo (3)
Dati delle Transazioni	2	Integrità, Autenticità e disponibilità	Nel caso in cui venga violata la policy di integrità, i dati relativi alle trasazioni (quindi indirizzi di fatturazione , metodi di pagamento, etc) potrebbero essere manipolati e di conseguenza l'attività registrata dal software, relativa alle transazioni, viene valutata rispetto a dati non corretti. (5) Nel caso in cui venga violata la policy di autenticità, i dati delle trasazioni registrati non sono autentici e possono insorgere problemi relativi alla destinazione della merce o all'affidabilità del pagamento. (5) Nel caso in cui venga violata la policy della disponibilità, i dati relativi alla merce disponibile potrebbero essere incorretti e quindi potrebbero insorgere problemi relativi alla quantità di merce ordinata/da spedire. (4)
Aggiornamento Registro Attività	2	Disponibile, Autenticità, Non Ripudiabilità	Disponibilità:Se il registro non è disponibile, potremmo perdere traccia di alcune azioni intraprese nel sistema(3) Autenticità: Se i dati dl registro non sono autentici o sono manipolati, si potrebbero verificare problemi riguardo qualsiasi tipo di storico relativo alle azioni intraprese nel sistema, come ad esempio un pagamento diverso da quello effettivo o quantità diversa di merce (5) Non Ripudiabilità: Se viene meno la non ripudiabilità, si avrebbe che un qualsiasi attore potrebbe negare che un'azione sia effettivamente successa, come ad esempio un pagamento, od una transazione mai effettuata.(5)
Consultazione del Token	1	Disponibilità e response time limitata	La mancata consultazione del token, sia per tempi di attesa per i quali i clienti non sono disposti ad aspettare, sia per mancanza di risposta da parte del software, non consente agli utenti di visualizzare il token dell'acquisto effettuato(1). (impedirebbe agli attori intermedi della catena di distribuzione di poter procedere con la rivendita dei semilavorati(5)).
Produzione e Lavorazione Materie Prime	2	Integrità, Autenticità, disponibilità	Nel caso in cui venga violata la policy di integrità, i dati relativi alla lavorazione o alla produzione potrebbero essere manipolati sia da utenti non autorizzati sia a causa di eventi casuali.(5) Nel caso in cui venga violata l'autenticità i dati relativi all'operazione potrebbero essere trasmessi da fonti non attendibili. (5) In entrambi i casi, la violazione renderebbe non corrette le successive operazioni di lettura o di scrittura dei registri contenuti nel software; i dati utilizzati, infatti, potrebbero essere stati alterati oppure non avere alcun valore di. veridicità. Le violazioni
Autenticazione	2	Integrità, Riservatezza, disponibilità	Integrità: I dati di login inseriti per l'autenticazione devono essere integri e protette da , altrimenti è possibile incorrere in errori di autenticazione (2) Riservatezza: In caso venisse violata, i dati di autenticazione potrebbero essere intercettati (4) Disponibilità: In caso di violazione, l'utente non potrebbe accedere per effettuare operazioni, rallentando i processi produttivi (3)

Figura 2.2: Tabella degli asset

Terminata la fase di definizione delle policy da utilizzare e degli asset da considerare di valore si procede con l'analisi dei rischi.

2.5 Risk Analysis

2.5.1 Threat Identification

Dopo aver identificato ed analizzato gli asset sono state individuate le possibili minacce a cui potrebbero essere esposti. A tale scopo si è fatto uso della metodologia **STRIDE-DUA**, nella quale le minacce vengono classificate in sei tipologie, ognuna delle quali viola un requisito:

- **S**poofing: utilizzare l'identità di qualcun altro (violazione dell'Authenticity)
- **T**ampering: modifica dei dati (violazione dell'Integrity)
- **R**epudiation: dichiarare di non aver compiuto una determinata azione (violazione del Non-Repudiation)
- **I**nformation Disclosure: esporre le informazioni ad un soggetto non autorizzato (violazione della Confidentiality)
- **D**enial of Service: non rendere disponibile un servizio (violazione dell'Availability)
- **E**levation of Privilege: ottenere permessi senza un'autorizzazione (violazione dell'Authorization)
- **D**anger: la compromissione dell'asset può far scaturire un danno (violazione della Safety)
- **U**nreliability: impossibilità di fare affidamento non garantendo il servizio come specificato (violazione dell'Affidabilità)
- **A**bsence of Resilience: impossibilità di riprendersi in maniera rapida da incidenti informatici (violazione della Resilience)

Le minacce riportate nella tabella STRIDE-DUA sono state estrapolate da **CAPEC**, consentendo di avere informazioni più dettagliate relative ad ogni attacco preso in considerazione e permettendo, perciò, di ricavare le possibili soluzioni.

Asset	Spoofing	Tampering	Repudiation	Information disclosure	DOS	Evation of privilege	Danger	Unreliability	Absence of Resilience	Exposure	Attack	Inherent Probability	Inherent Risk
Generazione Token Carbon Footprint			X	X		X		X		4	CAPEC-39: Manipulating Opaque Client-based Data Tokens	3	12
				X		X				3	CAPEC-573: Process Footprinting	3	9
		X						X		3	Tampering	4	12
					X			X		4	CAPEC-25: Forced Deadlock	4	12
Dati delle Transazioni	X	X		X		X				5	CAPEC-123: Buffer Manipulation	4	20
	X	X		X		X				4	CAPEC-385: Transaction or Event Tampering via Application API Manip	3	12
Aggiornamento Registro Attività		X	X						X	5	Accidental Manipulation/Deletion	1	5
		X	X	X		X				5	CAPEC-94 AiTM Adversary in The Middle	4	20
		X			X			X	X	3	CAPEC-125 Flooding	4	12
Consultazione del Token					X			X		3	Capec-125 Flooding	5	15
	X			X						4	Capec 117: Interception	4	16
Produzione e Lavorazione Materie Prime	X	X		X		X		X		4	Capec 22: Exploiting Trust in Client	3	12
	X	X		X				X		4	CAPEC-162: Manipulating Hidden Fields	3	12
			X		X					3	CAPEC-582: Route Disabling	2	6
Autenticazione		X		X		X				5	CAPEC-115 Authentication Bypass	2	10
						X				5	CAPEC-233 Privilege Escalation	2	10
					X					4	CAPEC-151 Identity Spoofing	3	12
	X				X	X		X		3	CAPEC-125 Flooding	4	12

Figura 2.3: Tabella delle minacce

2.5.2 Attack Assessment

Nel modello STRIDE soprastante ad ogni rischio è stata associata una valutazione numerica. Tale valutazione viene calcolata con la formula:

$$Rischio = Probabilitá \times Impatto$$

Infine, è stata definita una matrice per classificare i risultati ottenuti. Tale matrice prende il nome di **Matrice del Rischio**.

Tutta questa fase è stata di notevole importanza perchè ci ha permesso di evidenziare quali fossero i possibili rischi ed i relativi legami di questi ultimi ad ogni asset.

	Negligible (1)	Minor (2)	Moderate (3)	Significant (4)	Severe (5)
Very Likely (5)	5	10	15	20	25
Likely (4)	4	8	12	16	20
Possible (3)	3	6	9	12	15
Unlikely (2)	2	4	6	8	10
Very Unlikely (1)	1	2	3	4	5

Figura 2.4: Matrice del Rischio

2.6 Use, Abuse e Misuse Case in I*

Terminata la fase di identificazione delle minacce l'attenzione si è spostata su tutte quelle azioni che possono interagire con il sistema. Queste azioni vengono divise in tre categorie:

- **Use Case**: rappresentano le interazioni tra un attore ed il sistema con lo scopo di raggiungere un obiettivo;
- **Abuse Case**: azioni intenzionalmente dannose per il sistema, effettuate da parte di un attaccante esterno;
- **Misuse Case**: azioni dannose per il sistema causate da atti involontari da parte di qualche attore.

Al diagramma I* vengono quindi aggiunte tutte le possibili violazioni dovute agli Abuse Case ed ai Misuse Case, ottenendo quindi un diagramma aggiornato:

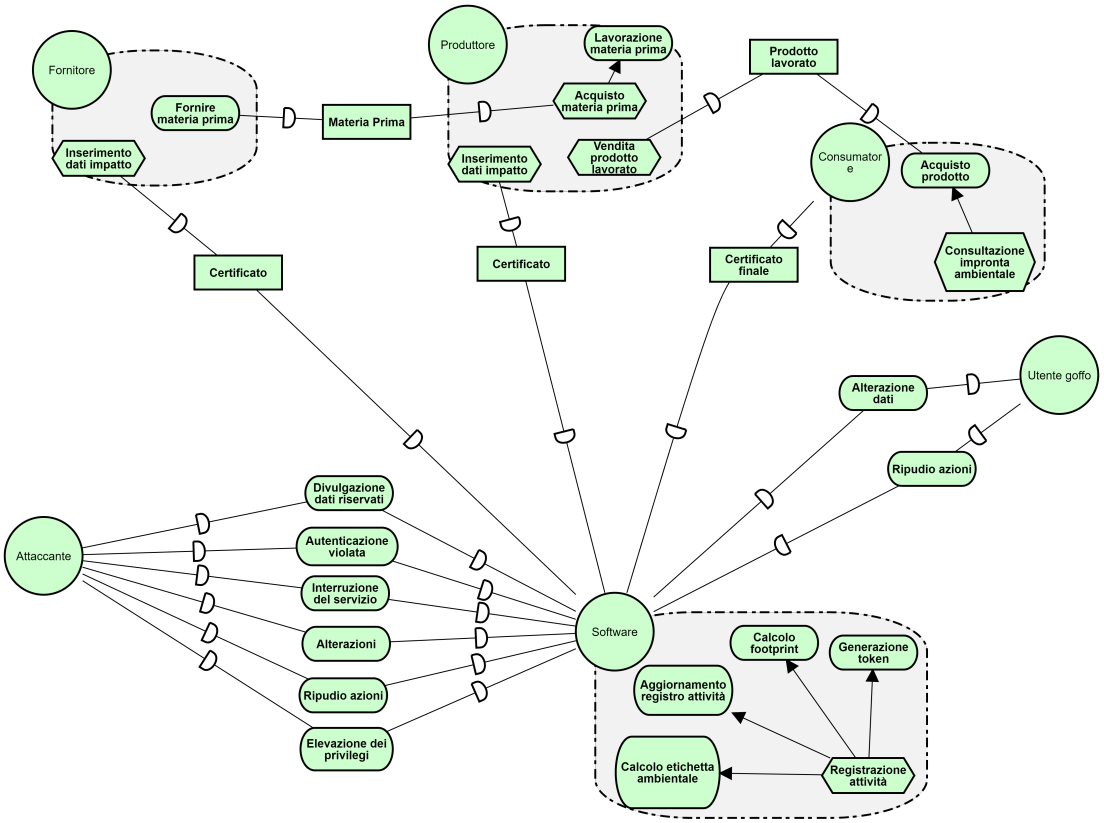

Figura 2.5: Diagramma I* con aggiunta delle possibili violazioni dovute agli Abuse Case ed ai Misuse Case.

Utilizzando lo **schema di Jacobson** sono state riportate le informazioni relative ad ognuna di queste azioni.

2.6.1 Use Case

Case Type	Use Case	Case ID	U-TKN-1
Case Name	Generazione Token Carbon Footprint		
Actors	Software, Produttore, Fornitore, Cliente		
Description	Software calculates footprint from Produttore processing data and Fornitore Processing data, delivering a token to the Produttore who sells the final product. This activity is also labeled and figures in the activity register		
Data	Label, activity register, Produttore processing data, Fornitore processing data		
Stimulus and preconditions	Any working process starts the flow, which ends when the product is sold		
Basic Flow	Fornitore delivers raw material and sends data to the software for the footprint calculation, while Produttore processes the material and sends his own data to the software as well. Software gives back a token to the Produttore representing the carbon footprint, which will be included in the transaction data and received by the final Client as well		
Alternative Flow			
Exception Flow	Software services are unavailable; Produttore (Fornitore) refuses to give information about footprint; software receives invalid data;		
Response and Postconditions	Footprint is saved in the activity register; an immutable label is generated; the footprint is available for the client;		
Non Functional Requirements			
Comments			

Figura 2.6: Use Case 1

Case Type	Use Case	Case ID	U-TRN-1
Case Name	Transazione		
Actors	Produttore, Cliente, Software, Fornitore		
Description	A buys materials from B		
Data	Id Product, Token Produttore		
Stimulus and preconditions	A can acquire raw materials from B Raw material available in B's wallet		
Basic Flow	1) A purchases raw material from B 2) B sends raw material and its relative token 3) A receives raw material and its relative token 4) Software records the operation on the system registry		
Alternative Flow			
Exception Flow	1) Produttore purchases raw material from Fornitore 2) Request from Produttore exceeds Fornitore's product availability		
Response and Postconditions	Confirmation of successful transaction and wallet update		
Non Functional Requirements			
Comments			

Figura 2.7: Use Case 2

Case Type	Use Case	Case ID	U-PROD-2
Case Name	Aggiornamento Registro Attività		
Actors	Produttore, Fornitore, Cliente, Software		
Description	The software logs in the system registry all the operations made		
Data	Multiple data depending on actors involved: Transaction data, raw material producing data, raw material manipulation data, carbon footprint token, article ID		
Stimulus and preconditions	Any actor does any kind of action		
Basic Flow	1)Actor makes any kind of action, for example A buys an article from B 2)System takes care of and approves the transaction 3)Software logs the operation in the system registry		
Alternative Flow			
Exception Flow	Registry unavailable		
Response and Postconditions	1)Transaction is logged both on the system registry (log) and on the blockchain.		
Non Functional Requirements			
Comments			

Figura 2.8: Use Case 3

Case Type	Use Case	Case ID	U-PLMP-1
Case Name	Produzione/Lavorazione Materie Prime		
Actors	Authorized User, (Software)		
Description	Authorized user process raw material A and produces product B; then inputs the data into the software		
Data	Dati Lavorazione Authorized User, Token		
Stimulus and preconditions	1)Product A in user's wallet		
Basic Flow	1) A new product B is produced 2)User inputs the processing data into the software 3)A new Carbon Footprint is Created 4)Software records the operation		
Alternative Flow			
Exception Flow	Fields in input incorrect or incomplete		
Response and Postconditions	1) Activity registered 2) Carbon Footprint returned to user 3) User's wallet updated		
Non Functional Requirements	System checks user's permission		
Comments	U-PLMP-1 generalizes two differents use cases: 1) Produzione materia prima -Authorized User = Fornitore -Product A = natural resources -Product B = raw material 2) Lavorazione materia prima -Authorized User = Produttore -Product A = raw material -Product B = processed product		

Figura 2.9: Use Case 4

Case Type	Use Case	Case ID	U-CLI-3
Case Name	Consultazione Token		
Actors	Lavoratore, Cliente		
Description	After the purchase, at every level of the chain supply, the Cliente receives the Lavoratore's token and purchase data. Thanks to the Lavoratore's token received, Cliente will be able to consult the environmental footprint or use it for its own purposes.		
Data	Articles Id, Lavoratore's Token		
Stimulus and preconditions	1) Products purchase went successfully: the order was finalized and the products ordered have been received		
Basic Flow	1)Cliente checks the lavoratore's Token		
Alternative Flow			
Exception Flow			
Response and Postconditions	1)Software shows the token to Cliente as the latter requested.		
Non Functional Requirements	1) Availability, reasonable response time		
Comments			

Figura 2.10: Use Case 5

Case Type	Use Case	Case ID	U-AUTH-1
Case Name	Autenticazione		
Actors	Produttore, Fornitore, Cliente, Software		
Description	A user tries to log into the system to operate		
Data	Username and Password		
Stimulus and preconditions	User needs to log in to operate		
Basic Flow	1)A inserts his credential information on the log-in page 2)Software verifies if the credentials are valid 3)Software answers back with confirmation of successful log-in		
Alternative Flow			
Exception Flow	1)A inserts an excessively long username or password 2)Software verifies if the credentials are valid 3)Software goes in exception flow (out of bound parameters)		
Response and Postconditions	1) User logs in and can start doing transactions		
Non Functional Requirements			
Comments			

Figura 2.11: Use Case 6

2.6.2 Abuse Case

Abuse Case	
Use case ID:	A-RU-01
Use case name:	Manipulate Registry Information
Actors	Attacker, Software
Description	An attacker exploits a weakness in the software to access and modify the system registry
Data	System Registry
Stimulus and Pre.	The targeted application relies on stored values on the registry. The registry path is public.
Attack 1 flow	The attacker exploits a weakness in the source code The attacker then uses said exploit to edit the data via the software.
Attack 2 flow	The attacker accesses the registry via the public path. The attacker alters/deletes the data to his own gain. The Registry is then updated with the altered data.
Response and Post.	The registry contains the altered data and all the other actors can no longer access the original data, causing problems in transactions between the parts.
Mitigations	Using a blockchain, prevents the modification of the data. Therefore, inserting the transactions into the blockchain solves the altering data problem.
Non Functionals Requirements	

Figura 2.12: Abuse Case 1

Abuse Case	
Use case ID:	A-RU-02
Use case name:	AiTM Attack
Actors	Outsider Attacker, Software, Produttore, Fornitore, Cliente
Description	An attacker intercepts the network traffic between any actor and the software and is able to manipulate that information.
Data	Transaction Data, Token
Stimulus and Pre.	Two actors communicating. The attacker knows the protocols used between the two actors and can decrypt the data.
Attack 1 flow	The attacker inserts himself into the communication channel between the two actors. The attacker can observe, filter, and alter the passing data of its choosing to gain access and manipulate the information for their own advantage.
Attack 2 flow	
Response and Post.	The attacker can store this data for his gain. Furthermore, if the attacker altered some data, the other actors who desire to read that particular data, will read an altered version of it instead.
Mitigations	Encrypting the data guarantees a higher level of security.
Non Functionals Requirements	

Figura 2.13: Abuse Case 2

Abuse Case	
Use case ID:	A-S-01
Use case name:	Buffer Manipulation
Actors	Attacker, Software
Description	An attacker manipulates an application's interaction with a buffer to read or modify data they shouldn't have access to.
Data	Transaction Data
Stimulus and Pre.	The adversary must identify a programmatic means for interacting with a buffer, such as vulnerable C code, and be able to provide input to this interaction.
Attack 1 flow	The attacker exploits a weakness in the source code (such as vulnerable C code) The attacker uses that weakness to manipulate data.
Response and Post.	The software now contain altered data and the actors can no longer carry out secure transactions.
Mitigations	Consider using a code language, like Java, or compiler that limits the ability of developers to act beyond the bounds of a buffer or consider using secure functions instead of those vulnerable to buffer manipulations
Non Functionals Requirements	

Figura 2.14: Abuse Case 3

Abuse Case	
Use case ID:	A-S-02
Use case name:	Transaction or Event Tampering via Application API Manipulation
Actors	Attacker, Software
Description	An attacker hosts or joins an event or transaction within an application framework in order to change the content of messages or items that are being exchanged.
Data	Transaction Data
Stimulus and Pre.	Targeted software is utilizing application framework APIs.
Attack 1 flow	The attacker hosts/joins a transaction The attacker manipulates content in order to produce content that look authentic or substitute one item with another.
Response and Post.	The software now contain altered data and the actors can no longer carry out secure transactions.
Mitigations	Using a blockchain, prevents the modification of the data.
Non Functionals Requirements	

Figura 2.15: Abuse Case 4

Abuse Case	
Use case ID:	A-RO-01
Use case name:	Invalid Product operation Registration
Actors	Outsider Attacker, System
Description	An outsider attacker has the ability to insert invalid data into the system.
Data	Product Processing Data
Stimulus and Pre.	Attacker finds a way to get access the functions of the system without required privileges. System records data without verifying its authenticity
Attack flow	Attacker finds a weakness in the system, by inspecting source code or running tests on specific targets. Attacker sends invalid inputs via a side channel. System records the operation without verifying data's authenticty.
Response and Post.	Future read operations of the generated token and transactions relating to the purchase of the product will be incorrect and must be invalidated.
Mitigations	Monitoring system access by improving authentication and privilege control mechanisms.
Non Functionals Requirements	

Figura 2.16: Abuse Case 5

Abuse Case	
Use case ID:	A-DPR-01
Use case name:	Denial production recording functionality.
Actors	Outsider Attacker, System
Description	An attacker has access to either network route or system resources.
Data	Product Processing Data
Stimulus and Pre.	An attacker has the ability to manipulate one or more resources in order to obstruct the interactions between system components.
Attack flow	Attacker either changes some aspect of a resource's state or availability or to disables the network route between two targets interrupting the communications channel between users and system.
Response and Post.	Authorized users are not able to access the registration functionality of a new process of a product.
Mitigations	Control access to system files.
Non Functionals Requirements	

Figura 2.17: Abuse Case 6

Abuse Case	
Use case ID:	A-TKN-01
Use case name:	Distributed Denial of Service
Actors	Internal/Outsider Attacker, Cliente, Grossista, Software
Description	An attacker has the ability to make tokens unattainable
Data	Token
Stimulus and Pre.	An attacker can get access to a(many) node(s) and wants to make it(them) unable to manage the requests
Attack 1 flow	An attacker makes the resource where the cliente's tokens are hosted unresponsive with a large amount of network requests from a(many) client(s)
Response and Post.	Implement a firewall that can block eccessive amount of request to the software.
Mitigations	Using a blockchain, due to its nature, a DDOS is harder to execute
Non Functionals Requirements	Avaiability, Response time

Figura 2.18: Abuse Case 7

Abuse Case	
Use case ID:	A-TKN-02
Use case name:	Token interception
Actors	Internal/Outsider Attacker, Cliente, Grossista, Software
Description	An attacker has the ability to intercept a token during a token request from a client
Data	Token
Stimulus and Pre.	Messages are not confidential, no obfuscation method implemented
Attack 1 flow	An attacker can use a man in the middle type of attack, or a sniffing software to intercept unprotected messages
Response and Post.	Implement a system of authentication and obfuscation of messages
Mitigations	Token are bound to the correct actors, due to the architecture of the smart contracts
Non Functionals Requirements	Confidentiality

Figura 2.19: Abuse Case 8

Use case ID:	A-TKN-02
Use case name:	Token alteration
Actors	Internal/Outsider Attacker
Description	An attacker can be: - Silently hearing the communication between Fornitore/Produttore and Software, with the purpose of taking the identity of one of the actors and deliver fake data to the software - If already having access to the software, even with low privilege level, he can escalate his access level, trying to alterate the software flow with attacks such as buffer overflow or sql injection to take control of the stream, maybe using global offset table to take advantage of little pieces of code (gadget) which can be called unproperly with malicious intentions.
Data	Token footprint
Stimulus and Pre.	An attacker may be interested in damaging a manifactury company business by altering the footprint data
Attack 1 flow	An attacker can be hearing during the handshake or whatever authentication method is used
Attack 2 flow	An attacker gets hold of some portion of code belonging to libraries imported in the main source, maybe not even used but still present in the offset table, then he can take control of the execution flow
Response and Post.	Keep a report with all the accesses executed to the application. Use sandboxed software
Mitigations	Guarantee a strong authentication by, for example, multi-factor auth. Avoid importing unnecessary libraries. Using the blockchain to generate the token
Non Functionals Requirements	

Figura 2.20: Abuse Case 9

Abuse Case	
Use case ID:	A-AUTH-01
Use case name:	Authentication Bypass
Actors	Produttore, Fornitore, Cliente, Software
Description	An attacker gains access to application, service, or device with the privileges of an authorized or privileged user by evading or circumventing an authentication mechanism. The attacker is therefore able to access protected data without authentication ever having taken place.
Data	
Stimulus and Pre.	An authentication mechanism or subsystem implementing some form of authentication such as passwords, digest authentication, security certificates, etc.
Attack 1 flow	1)Attacker exploits a weakness in the log-in system to avoid authenticating 2)Attacker infiltrates the software enabling him restricted actions
Attack 2 flow	
Response and Post.	Attacker can temper with the registry adding new, false transactions
Mitigations	Strenghten log-in process, adding a multi-auth tool
Non Functionals Requirements	

Figura 2.21: Abuse Case 10

Abuse Case	
Use case ID:	A-AUTH-02
Use case name:	Privilege Escalation
Actors	Produttore, Fornitore, Cliente, Software
Description	An adversary exploits a weakness enabling them to elevate their privilege and perform an action that they are not supposed to be authorized to perform.
Data	Log-In Information
Stimulus and Pre.	An attacker gets ahold of the log-in information about another user
Attack 1 flow	1)Attacker uses the data of a high privilege user 2)Attacker then acts with said user privileges,for example creating false data regarding transactions
Attack 2 flow	
Response and Post.	Attacker gets ahold of all the information about user, and can temper with the registry adding new, false transactions
Mitigations	Log-In data should always be encrypted, and not disclosed; adding a multi auth tool
Non Functionals Requirements	

Figura 2.22: Abuse Case 11

2.6.3 Misuse Case

Misuse Case Spefication	
Use case ID:	M-TKN-01
Use case name:	Carbon Footprint tampering
Actors	Clumsy user
Description	A clumsy user might modify the environmental certificate attesting a transaction
Data	Carbon Footprint
Stimulus and Preconditions	No measures to prevent the carbon footprint tampering
Attack 1 Flow	Accidental modification and/or deletion of the certificate
Response and Postconditions	Carbon Footprints are not reliable, they are subjects to users actions
Mitigations	The use of a blockchain and the usage of a token as a representation of the Carbon Footprint can prevent its tampering and guarantees the environmental impact
Non functionals Requirements	Integrity

Figura 2.23: Misuse Case 1

Misuse Case Spefication	
Use case ID:	M-AUTH-01
Use case name:	Accidental Deletion
Actors	Clumsy User
Description	A user accidentally deletes an entry in the application register
Data	Transaction Data, User Data
Stimulus and Preconditions	User can log in and has the right privileges, no backups stored and irreversible actions, data saved only in local
Attack 1 Flow	1)User logs Into the system 2)Managing the application register, the user accidentally deletes an entry
Response and Postconditions	The register informations become invalid, obsolete or corrupted, hence the data might be inconsistent
Mitigations	Using the blockchain, storing the transaction on the blockchain in addition of storing them in the application register, helps keeping the register updated and unmodified.
Non functionals Requirements	Redundacy

Figura 2.24: Misuse Case 2

Use case ID:	M-ERR-01
Use case name:	Insertion of erroneous values during transactions
Actors	Cliente, Grossista
Description	Users mistakenly sends incorrect values for the transaction. Ie: wrong amount, product
Data	Transaction data
Stimulus and Preconditions	No input validation and no means to update the transaction with the correct values
Attack 1 Flow	The actor sends a request with erroneous values for the transaction
Response and Postconditions	Software registers the request with wrong data
Mitigations	Insert input validation mechanisms, allow the user to create an update for the transaction
Non functionals Requirements	Integrity

Figura 2.25: Misuse Case 3

2.6.4 Attack Trees

Dovendo identificare i vettori di attacco e le relative superfici di impatto, per ogni Abuse e Misuse Case, è stato realizzato l'**Attack Tree**, uno strumento concettuale che ci permette di rappresentare i possibili scenari di attacco e, quindi, di identificare le ragioni che hanno portato ad un guasto del sistema.

Nelle successive immagini vengono riportati gli Attack Tree individuati, ognuno relativo alla policy che viene violata.

L'attaccante, attraverso tecniche di manipolazione del buffer/campi nascosti o l'invio continuo di una serie di pacchetti, potrebbe provocare una violazione dell'integrità.

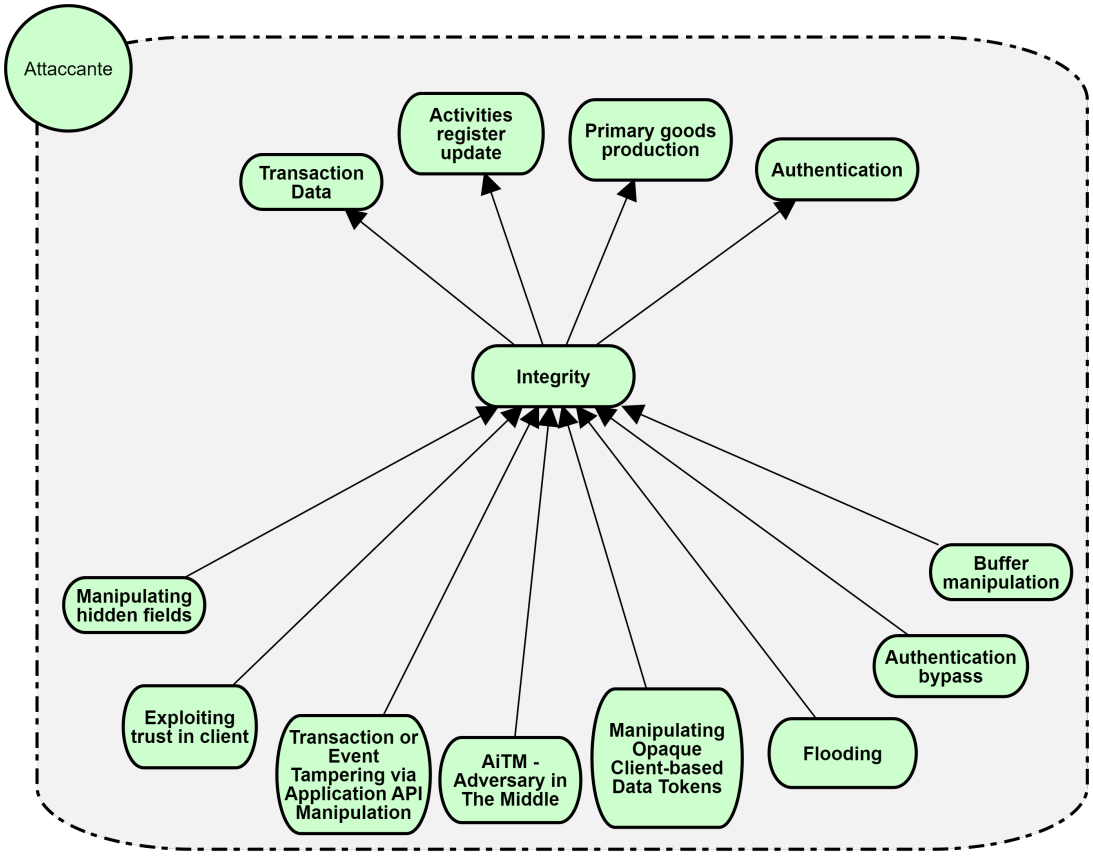

Figura 2.26: Attack Tree - Violazione dell'Integrità

Tramite l'utilizzo di tecniche di intercettazione o l'utilizzo di framework l'attaccante ospita un evento/transazione con il fine di cambiare il contenuto dei messaggi/il destinatario di una transazione con conseguente violazione dell'autenticità.

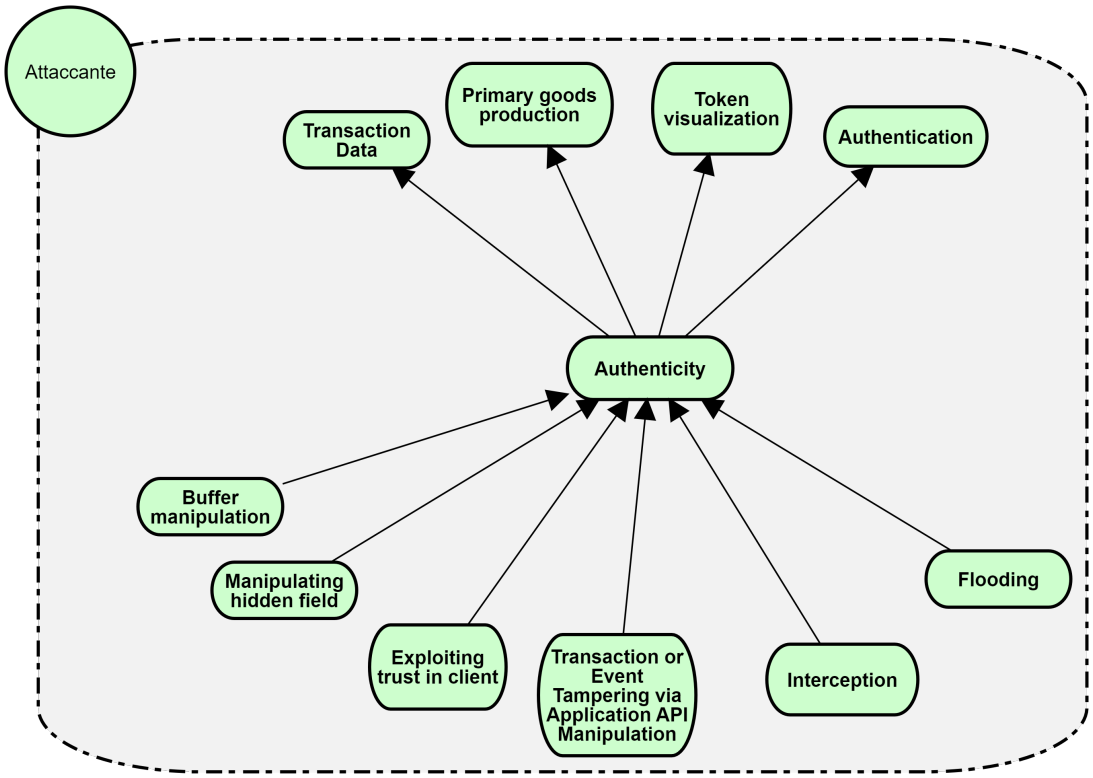

Figura 2.27: Attack Tree - Violazione dell'Autenticità

Con l'utilizzo del *Footprinting* (e.g.) l'attaccante raccoglie quanti più dati possibili identificando le opportunità di penetrazione, andando a violare la policy dell'Autorizzazione.

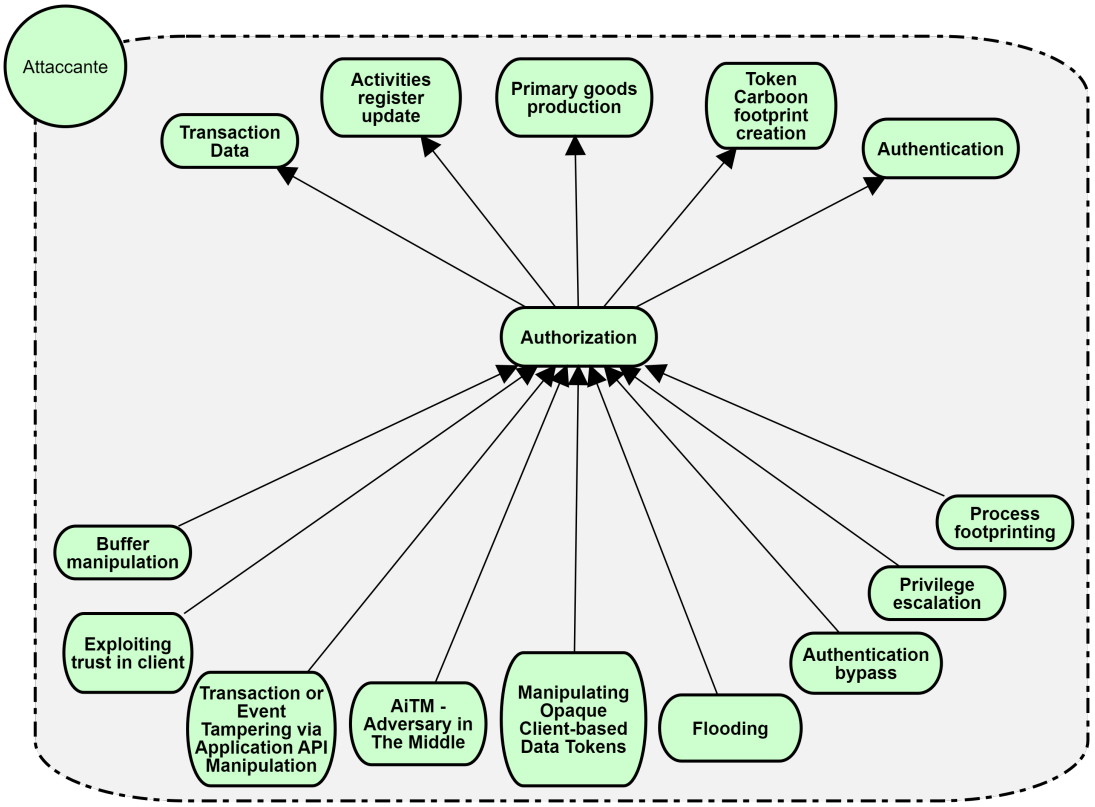

Figura 2.28: Attack Tree - Violazione dell'Autorizzazione

Con varie tecniche di manipolazione dei buffer, di intercettazione o dell'utilizzo di una procedura di accesso inusuale un attaccante provoca la violazione della Confidenzialità.

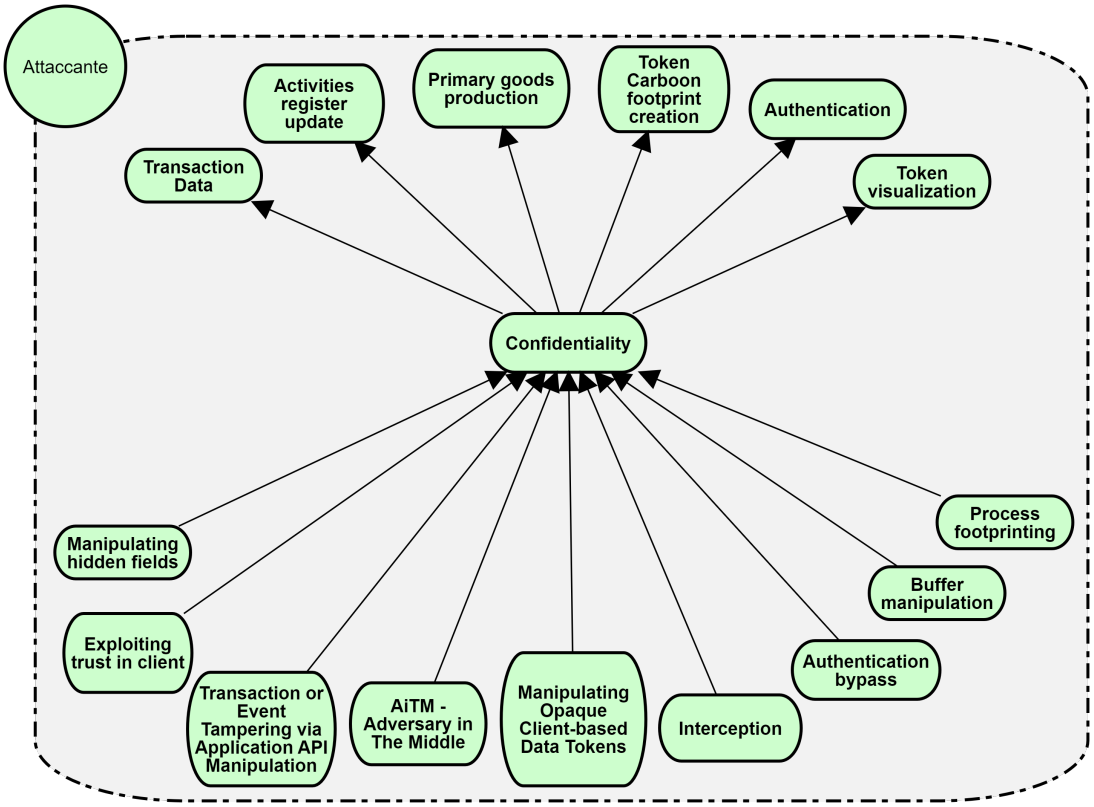

Figura 2.29: Attack Tree - Violazione della Confidenzialità

Un attaccante potrebbe provocare malfunzionamenti esaurendo le risorse del web server fino al punto di renderlo non più disponibile ai clienti e provocando perciò una violazione della disponibilità.

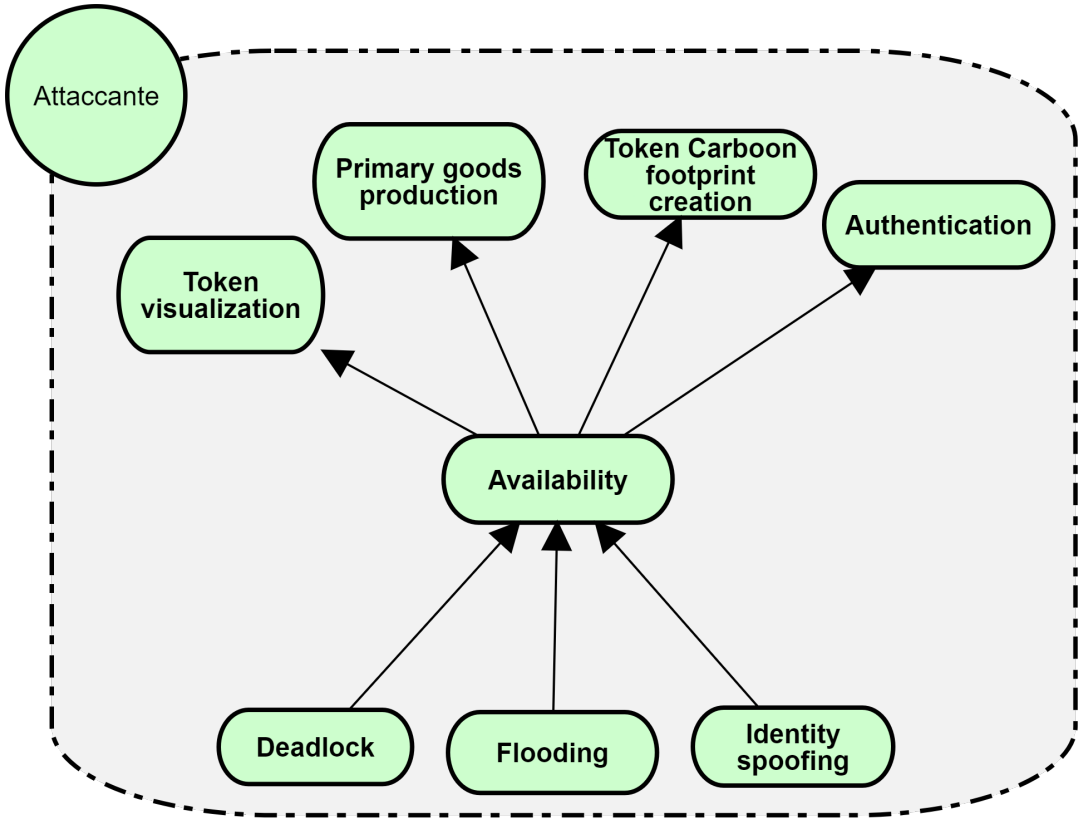

Figura 2.30: Attack Tree - Violazione della Disponibilità

Tramite la disabilitazione del router tra due bersagli o il posizionamento dell'attaccante tra i canali di comunicazione dei due target, un attaccante provoca una violazione del Non Ripudio.

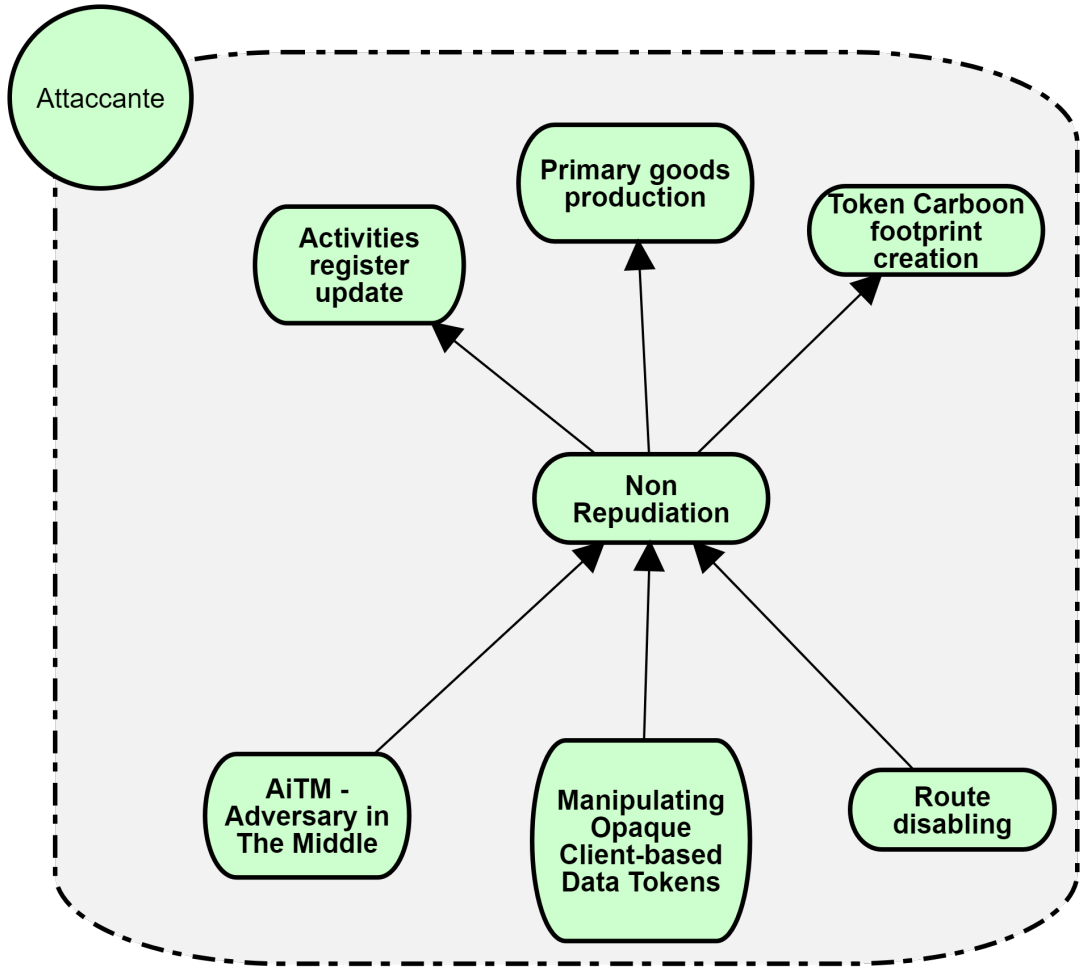

Figura 2.31: Attack Tree - Violazione del Non Ripudio

Effettuando azioni involontarie un utente provoca la violazione dell'Integrità e del Non Ripudio.

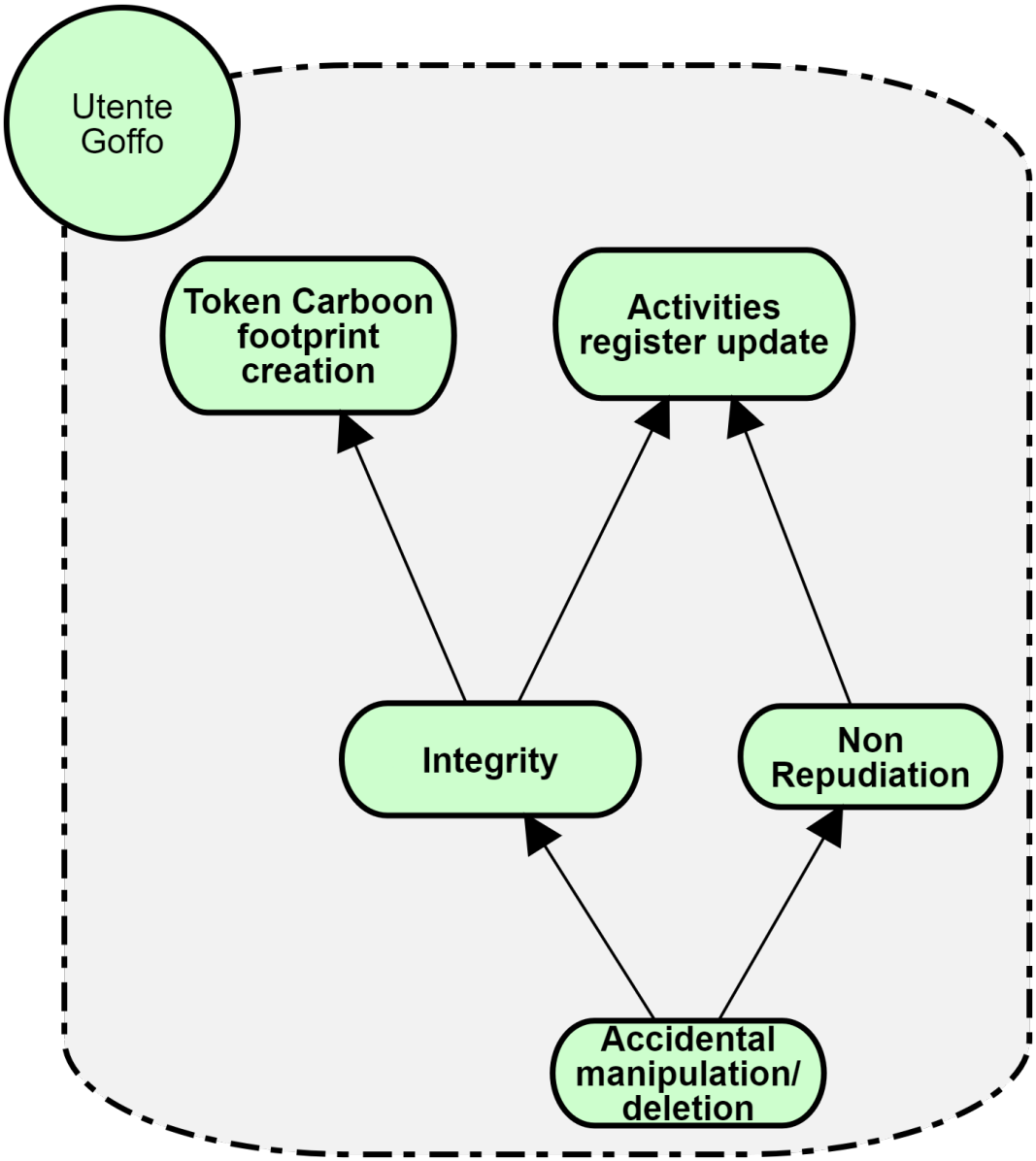

Figura 2.32: Attack Tree- Violazione del Non Ripudio e dell'Integrità

2.6.5 Risk Reduction

In questa fase è necessario decidere quali strumenti possono essere utilizzati per ridurre la probabilità di attacco.
Viene seguito un iter ben preciso, composto da sei passaggi:

1. Utilizzando il modello *STRIDE*, per ogni minaccia elenchiamo una serie di tecnologie che servono a ridurre la minaccia, e quindi la probabilità di attacco, presa in considerazione.

2. Per ognuna di queste tecnologie elencate si va a valutare il livello di fattibilità, ossia quanto è applicabile il tipo di tecnica di mitigazione relativo ai vari vettori di attacco.

3. Se la soluzione tecnologica è fattibile/applicabile allora si va a valutarne il costo.

4. Per ognuna delle soluzioni tecnologiche si va a valutare la probabilità di attacco (dopo averla introdotta nel sistema). In questo passaggio ci si aspetta che la probabilità attacco diminuisca con conseguente diminuzione/annullamento dei rischi.

5. Valutazione del costo complessivo.

6. Calcolo del rischio residuo.

Nella successiva tabella sono riportati i risultati ottenuti in seguito all'applicazione dei passaggi descritti in precedenza.

Asset	Spoofing	Tampering	Repudiation	Information disclosure	DoS	Elevation of privilege	Danger	Unreliability	Denial of Service	Exposure	Attack	Inherent Probability	Inherent Risk	Control	Cost	Feasibility	Residual Probability	Residual Impact	Residual Risk
Generazione Token Carbon Footprint			X	X		X		X		4	CAPEC-39: Manipulating Opaque Client-based Data Tokens	3	12	1)Using CRC 2) encryption or signed hash	3	1. Technically feasible 2. Technically feasible	2	4	6
				X		X				3	CAPEC-573: Process Footprinting	3	9	1)Using a software restriction policy	2	1. Technically feasible	1	3	3
		X						X		3	Tampering	4	12	1)Blockchain	2	1. Technically feasible	2	3	6
					X			X		4	CAPEC-25: Forced Deadlock	4	12	1)Improving synchronization algorithms	3	1. Technically feasible	2	3	6
Dati delle Transazioni	X	X		X		X				5	CAPEC-123: Buffer Manipulation	4	20	1)Traffic Encryption	3	1. Technically feasible	2	5	10
	X	X		X		X				4	CAPEC-385: Transaction or Event Tampering via Application API Manip	3	12	1) Blockchain	2	1. Technically feasible	2	4	8
Aggiornamento Registro Attività		X	X						X	5	Accidental Manipulation/Deletion	1	5	1)ACL 2)Store the data on multiple devices	3	1. Technically feasible 2. Technically feasible, implemented with the blockchain	1	2	2
		X	X	X		X				5	CAPEC-94 AiTM Adversary in The Middle	4	20	1)Data Encryption	2	1. Technically feasable	3	2	6
		X			X			X	X	3	CAPEC-125 Flooding	4	12	1)ACL2)Throttling between executions	2	1. Technically feasible 2. Feasable, but introduces delays in the system	1	4	4
Consultazione del Token					X			X		3	Capec-125 Flooding	5	15	1)ACL2)Throttling between executions	1	1. Technically feasible 2. Feasable, but introduces delays in the system	3	3	9
	X			X						4	Capec 117: Interception	4	16	1)Traffic Encryption	2	1.Technically feasible	2	4	8
Produzione e Lavorazione Materie Prime	X	X		X		X		X		4	Capec 22: Exploiting Trust in Client	3	12	Perform input validation for all remote conte	1	1. Technically feasible	2	4	8
	X	X		X				X		4	CAPEC-162: Manipulating Hidden Fields	3	12	Perform input validation for all remote conte	1	1. Technically feasible	1	4	4
			X		X					2	CAPEC-582: Route Disabling	2	6	Access protection to system files.	2	1. Technically feasible	1	4	4
Autenticazione		X		X		X				5	CAPEC-115 Authentication Bypass	2	10	1)ACL2)Strong authentication system	2	1. Technically feasible 2. Needs user intervention or approval	1	5	5
						X				5	CAPEC-233 Privilege Escalation	2	10	2)Implementing strong authentication system 3)Data execution prevention	3	2. Needs user intervention or approval 3. Feasable, and already part of the blockchain	1	5	5
					X					4	CAPEC-151 Identity Spoofing	3	12	1)ACL 2)Strong Authentication system	3	1. Technically feasible 2. Needs user intervention or approval	1	4	4
	X				X	X		X		3	CAPEC-125 Flooding	4	12	1)ACL2)Throttling continuous requests	3	1. Technically feasible 2. Feasable, but introduces delays in the system	3	3	9

Figura 2.33: Riduzione del Rischio

Capitolo 3

Implementazione

3.1 Sistema Operativo e Virtual Machine

Per l'implementazione, abbiamo sviluppato un applicativo web per la più facile fruizione dei servizi da parte dell'utente (semplificando l'interfaccia andiamo anche a ridurre rischi legati all'incorretto utilizzo).

Al fine di rendere più facile lo sviluppo cross-platform tra i vari componenti del gruppo abbiamo optato per l'utilizzo di una macchina virtuale; ed abbiamo scelto il sistema operativo basandoci su tre criteri:

- **facilità nell'utilizzo**
- **consumo ridotto di risorse**
- **sicurezza**

Questi criteri ci hanno portato a scegliere **Lubuntu 20.04**[1], una versione lightweight della classica Ubuntu Linux che però non ne compromette le prestazioni.

Per l'installazione della macchina virtuale abbiamo scelto **Oracle VM Virtualbox**[2], una soluzione Open-Source che ci permette l'emulazione di una macchina in tutti i suoi aspetti ed in modo totalmente gratuito.
La scelta di un sistema operativo come Linux ci fa ottenere il massimo controllo dal punto di vista del rispetto delle buone norme di programmazione sicura che sono riassunte dalle principali linee guida di **OWASP**, **Sommerville**, **Saltzer** e **Schroeder**.
Linux permette, ad esempio, l'assegnazione dei privilegi di *RWX* (lettura, scrittura, esecuzione) dei file per ogni utente, andando a soddisfare il principio dei **Privilegi Minimi** presente nelle linee guida sopracitate.

[1] Reperibile all'indirizzo: `https://www.ubuntu-it.org/derivate/lubuntu`

[2] Reperibile all'indirizzo: `https://www.virtualbox.org/`. La macchina virtuale in allegato alla relazione è stata realizzata con versione 6.1.34 di VirtualBox, unito all'extension pack 6.1.34, ed è reperibile a questo indirizzo `https://download.virtualbox.org/virtualbox/6.1.34/Oracle_VM_VirtualBox_Extension_Pack-6.1.34.vbox-extpack`.Senza extension pack è possibile che la VM non riesca ad avviarsi.

3.2 Strumenti Software (Blockchain)

Al fine di ovviare a molti degli attacchi ed ai rischi di sicurezza presenti all'interno dell'analisi dei casi d'uso, di cui si è parlato in precedenza, la soluzione migliore da implementare è stata quella della **Blockchain**.
Nello specifico abbiamo utilizzato **GoQuorum**[3], una blockchain privata basata sullo stesso modello della blockchain Ethereum, con le stesse modalità di gestione e programmazione dei contratti. A differenza di Ethereum, che si basa su un meccanismo di consenso *Proof-of-Work* (al momento della stesura della seguente, Ethereum è tuttavia in fase di passaggio a *Proof-of-State*), GoQuorum utilizza il meccanismo di consenso **Proof-of-Authority** rendendo la nostra rete di tipo *permissioned*.
Questo tipo di rete si addice perfettamente al nostro case-study per il tracking dell'impatto energetico di un processo produttivo, permettendo a pochi nodi, rappresentanti i vari esercenti, di effettuare transazioni all'interno della blockchain.
La scelta di questo metodo di consenso è legata anche alla rimozione dell'onerosità che i metodi PoS e PoW portano, andando ad abbattere i costi di gestione e di rete quali, ad esempio, gas fee.

Per l'installazione della rete GoQuorum ci siamo avvalsi di uno script del node package manager: *GoQuorum-Wizard*.
Lo script in questione permette la creazione di una blockchain del tipo sopraindicato, attraverso una procedura step by step, andando a selezionare specifiche impostazioni, che nel nostro caso risultano essere:

- **network configuration**: Bash, che permette l'avvio della blockchain attraverso uno script bash,
- **consensus mode**: Istanbul (resistente ai fault di tipo bizantino),
- **number of nodes**: 3 (per simulare le varie aziende esercenti),
- **Quorum version**: 21.0.4 (Blockchain),
- **Tessera version**: 21.1.1 (gestore delle transazioni),
- **additional tools**: Cakeshop (l'ufficiale block explorer di Quorum).

Il consensus mode Istanbul **(IBFT)** è stato scelto per la sua capacità di resistere ai fallimenti di tipo bizantino. Questa scelta è stata la migliore anche dal punto di vista dell'implementazione a causa della presenza di alcuni problemi con il **metodo RAFT** nella sottoscrizione agli eventi.

Per potersi interfacciare con la blockchain, abbiamo sfruttato le **API** di **Web3** per l'interfacciamento tra codice offchain ed onchain.

Gli smart contract sviluppati all'interno della blockchain di progetto sono stati scritti in linguaggio **Solidity 0.8.10**, sviluppando e testando attraverso l'utilizzo

[3]Reperibile all'indirizzo: `https://github.com/ConsenSys/quorum-wizard#readme`

dell'IDE **RemixIDE**[4], che consente di debuggare all'interno di una blockchain emulata dalla JVM o di collegarsi ad una blockchain personale.

Il software è stato programmato per funzionare con dei contratti già esistenti; per questo motivo, a livello di client, l'applicativo dovrà solo connettersi ad essi, senza andarli a generare né modificare.

3.3 Strumenti Software (Client Server)

L'architettura Client-Server è quella più adatta per realizzare il nostro applicativo. Si è scelto di utilizzare **NodeJS**[5], un ambiente di sviluppo javascript che negli ultimi anni si è affermato come ottimo strumento di sviluppo per applicativi web. Per lo sviluppo di un'applicazione Client-Server basata su NodeJS, abbiamo scelto **ExpressJS**, un potente framework per la gestione del back-end per web applications. ExpressJS, però, non riesce da solo a gestire tutte le funzionalità di cui abbiamo bisogno e per questo NodeJS mette a disposizione una grande libreria di pacchetti, meglio noti come *middleware*: ogni middleware si occupa di gestire una determinata funzionalità in modo da rendere modulare ed ordinata la realizzazione. Tale libreria prende il nome di *npm*[6] *Registry*: un registro open source costantemente aggiornato per permettere agli utenti di verificare quali pacchetti siano i migliori per le proprie esigenze.

I moduli importanti all'interno della nostra applicazione sono numerosi ma, in particolare, possiamo distinguerne due tipologie:

- **packages critici**
- **packages complementari**

3.3.1 Packages Critici

Tra i packages critici troviamo quelli di configurazione per generare l'infrastruttura che connetterà l'offchain con l'onchain:

- **web3js**[7]: costituisce l'API che permette di gestire la connessione con gli smart contract e più in generale la rete blockchain,
- **mysqljs**[8]: costituisce il driver di collegamento al DBMS MariaDB all'interno del quale abbiamo inserito informazioni relative all'autenticazione degli utenti.

[4]Reperibile all'indirizzo: `https://remix.ethereum.org/`
[5]Reperibile all'indirizzo: `https://nodejs.org/it/`
[6]Reperibile all'indirizzo: `https://www.npmjs.com/`
[7]Reperibile all'indirizzo: `https://web3js.readthedocs.io/`
[8]Maggiori informazioni: `https://github.com/mysqljs/mysql`

3.3.2 Packages Complementari

I packages ai quali ci riferiamo come complementari sono dei packages che aggiungono funzionalità al codice, senza bisogno di andare a scrivere del nuovo codice. Utilizzando questi pacchetti è possibile risparmiare molto tempo, ma non solo: tutti i pacchetti di npm sono costantemente controllati e, se qualcuno nella community si accorge di un problema che mette a rischio la sicurezza di un qualsiasi package, può segnalarlo; in tal modo lo sviluppatore, o chiunque ne sia in grado, può rilasciare una patch di sicurezza che si occupi di risolvere il problema insorto. Se ci sono problemi critici in qualche pacchetto verrà mostrato un warning riportante il grado di criticità del problema e l'eventuale disponibilità di patch. La funzione di audit ci permette di applicare queste eventuali patch.
Al momento di rilascio della nostra applicazione tutti i nostri packages sono esenti da problemi o sono patchati.I pacchetti utilizzati sono i seguenti:

- **@openzeppelin/contracts**[9]: libreria per l'utilizzo degli standard ERC721 per la generazione dei token non fungibili;
- **bcrypt**[10]: libreria per effettuare l'hash delle password;
- **crypto-js**[11]: libreria per effettuale la crittografia, nel nostro caso quella degli indirizzi dei portafogli;
- **cookie-parser**[12]: : libreria per il parsing dei cookie e la loro gestione;
- **dotenv**[13]: libreria che permette la gestione e l'aggiunta di variabili d'ambiente ad un processo;
- **express-handlebars**[14]: libreria per l'utilizzo dell'estensione hbs per l'aggiunta di funzionalità al codice HTML;
- **express-session**[15]: libreria per la gestione delle sessioni in javascript;
- **nodemon**[16]: Strumento di aiuto nello sviluppo dell'applicazione, consente di non dover rilanciare l'applicazione NodeJS in caso di modifica di un file;
- **Winston**[17]: libreria per l'utilizzo di funzioni di log a vari livelli.

[9]Maggiori informazioni: `https://github.com/OpenZeppelin/openzeppelin-contracts`
[10]Maggiori informazioni: `https://github.com/dcodeIO/bcrypt.js#readme`
[11]Maggiori informazioni: `https://github.com/brix/crypto-js#readme`
[12]Maggiori informazioni: `https://github.com/expressjs/cookie-parser#readme`
[13]Maggiori informazioni: `https://github.com/motdotla/dotenv#readme`
[14]Maggiori informazioni: `https://github.com/express-handlebars/express-handlebars`
[15]Maggiori informazioni: `https://github.com/expressjs/session#readme`
[16]Maggiori informazioni: `https://nodemon.io/`
[17]Maggiori informazioni: `https://github.com/winstonjs/winston#readme`

La libreria MySQL è stata utilizzata per la gestione di un piccolo database contenente tutte le informazioni anagrafiche d'interesse riguardo gli esercenti ed i clienti che si registrano al sito. Ciascuno di questi, inoltre, sarà munito di un indirizzo che rappresenterà il proprio wallet all'interno della blockchain. Per la gestione del servizio MySQL/mariaDB è stato installato all'interno della macchina l'applicativo *xampp*, il quale gestisce il servizio di mariaDB in automatico senza dover effettuare configurazioni particolari. Inoltre, comprende il tool *phpMyAdmin* il quale, in fase di testing, è stato fondamentale per risolvere alcuni problemi attraverso un'interfaccia grafica.

Per motivi di semplicità, rispettando il principio **Keep It Security Simple** *(KISS, OWASP)*, abbiamo optato per un'interfaccia grafica anche per l'utilizzo stesso dell'applicazione, andando a sviluppare un frontend intuitivo per una maggiore facilità d'uso. A tal proposito ci siamo serviti di **Handlebars**, un engine grafico per il potenziamento del normale HTML rendendo intelligente il codice potendo, ad esempio, inserire check su variabili direttamente nel codice HTML nel seguente modo:

```
{{#if condition}}
  {{#each object}}
      //some HTML code
  {{/each}}
{{/if}}
```

I suddetti codici vengono detti *hbs helpers*.

3.4 Smart Contract & Blockchain

Dopo un'attenta analisi preliminare su come poter implementare il programma, abbiamo individuato due categorie di oggetti d'interesse:

1. **Transazioni**

2. **Token**

Ogni transazione deve essere salvata e bisogna tenerne traccia all'interno della blockchain: dati come materie prime, prodotti, ed i loro rispettivi proprietari sono quelli di nostro interesse.
Inoltre, per ciascuna di queste materie prime o prodotto bisogna registrarne l'impatto ambientale che il processo produttivo ha generato. Sono quindi stati realizzati due contratti:

- **Transazioni**: si trova nella cartella contracts/Transazioni.sol

- **CarbonFootprint**: si trova nella cartella contracts/CarbonFootprint.sol.

3.4.1 CarbonFootprint.sol

Il contratto CarbonFootprint è il fulcro del progetto: da specifiche ogni prodotto e materia prima durante la lavorazione o produzione è soggetta alla produzione di impatto ambientale.
Tale impatto ambientale deve essere registrato ed eventualmente aggiornato nel caso in cui il prodotto subisca lavorazioni.

Dopo aver effettuato uno studio delle possibili soluzioni, si è optato per la libreria fornita da OpenZeppelin, nello specifico quella relativa allo standard **ERC721**[18].
Lo standard ERC721 è lo standard che definisce i **N**on **F**ungible **T**okens, che a differenza dei fungible token **ERC20**[19] (come possono esserlo ad esempio Ethereum o Bitcoin) fa sì che ciascun token non fungibile (o NFT) sia unico e seppur possa essere della stessa tipologia di un altro NFT, è comunque diverso da tutti gli altri.
Abbiamo utilizzato gli **NFT** per rappresentare una sorta di "scontrino" da attaccare sul prodotto o materia contenente i dati relativi all'impatto ambientale che il processo produttivo ha portato per quella specifica materia prima o prodotto.
Alla produzione di una materia prima, verrà anche eseguito il *mint* di un nuovo token, contenente l'impatto.
Andando avanti nella catena produttiva le materie prime verranno lavorate e trasformate in prodotti, la cui produzione comporta anch'essa un ulteriore impatto, e quest'ultimo dovrà essere sommato all'impatto fino ad ora registrato: per la natura immutabile dei token nella blockchain però, non possiamo modificare i token già esistenti, ma dobbiamo crearne uno nuovo contenente la somma degli impatti totali.

Per quanto riguarda le funzioni del contratto, abbiamo effettuato l'override della funzione *safeMint* già contenuta all'interno della libreria ERC721 ed, inoltre, è stata inserita la funzione *getCarbonFootprint* per leggere la variabile contenente il valore dell'impatto del token.

3.4.2 Transazioni.sol

Il contratto Transazioni si occuperà di gestire tutte le operazioni sulle materie prime e sui prodotti:

- creazione di una materia prima
- creazione di un prodotto
- acquisto di un prodotto
- acquisto di una materia prima

[18]Maggiori informazioni: `https://ethereum.org/it/developers/docs/standards/tokens/erc-721/`
[19]Maggiori informazioni: `https://ethereum.org/it/developers/docs/standards/tokens/erc-20/`

Per rappresentare i prodotti e le materie prime abbiamo realizzato delle semplici strutture dati contenenti varie informazioni relative ad essi.
Per la materia prima:

- ID lotto materia prima
- nome
- ID token impatto ambientale associato
- quantità
- flag di lotto disponibile
- percentuale di materia prima rimanente rispetto alla quantità iniziale

Le linee guida di programmazione sicura, in particolare quelle di OWASP, ci dicono che è bene mantenere i privilegi che ogni utente possiede al minimo ed implementare la separazione dei compiti: ciascuno degli utenti deve poter effettuare modifiche solo ai dati che gli competono e per questo motivo ci siamo ingegnati per riuscire a soddisfare le suddette linee guida.

La nostra implementazione dei prodotti e delle materie prime si basa sul mapping annidato: ad ogni indirizzo corrisponde un ulteriore mapping che, ad ogni valore intero, fa corrispondere una struttura prodotto o materia prima.

```
mapping(address=>mapping(uint256=>Product)) products;
mapping(address=> mapping(uint256=>RawMaterial)) raw_materials;
```

Figura 3.1: Esempio di mapping annidato

Tale implementazione consente, per come è stato codificato il contratto, ad ogni utente di effettuare solamente l'accesso senza possibilità di accedere ai suoi prodotti/materie.
Per accedere alle proprie risorse è stata implementata questa soluzione:

```
1 products[msg.sender][id]
2 raw_materials[msg.sender][id]
```

La creazione di una materia prima è molto semplice e comporta il mint di un nuovo token da associarle.

L'acquisto di una materia prima o di un prodotto comporta il trasferimento della suddetta risorsa dal portafoglio dell'utente proprietario all'acquirente. Cambiando di proprietario, però, dobbiamo anche tener conto del token associato alla risorsa: l'acquirente dovrà diventare proprietario anche del token.
Per fare ciò ci siamo serviti della funzione *safeTransferFrom*, contenuta sempre

all'interno dello standard ERC721, che permette di fare il trasferimento della proprietà da un utente ad un altro.

Per la realizzazione di un prodotto, abbiamo seguito i seguenti passi:

1. verificare che ci siano abbastanza materie prime per creare un nuovo prodotto
2. in caso positivo, scalare le materie prime a partire dal lotto più vecchio, in modo tale da ridurre gli sprechi
3. tenere traccia di quali lotti di materia prima compongono un prodotto
4. aggiornare l'impatto ambientale totale in percentuale alla quantità di materia prima utilizzata

Abbiamo posto una particolare attenzione al meccanismo di calcolo del Carbon Footprint cumulativo dei lotti di materie utilizzate, che è stato pensato nel seguente modo: sapendo quante sono le materie prime necessarie per la realizzazione del prodotto il contratto va a controllare tra tutte le materie prime, scartando quelle contrassegnate dal flag **non disponibile=true** in combinazione con **amount=0**. Nel caso in cui la quantità del lotto dell'iterazione corrente sia sufficiente a realizzare il prodotto in questione: in caso positivo, la quantità rimanente viene impostata a zero, mentre alla quantità disponibile nel lotto verrà sottratta la quantità effettivamente utilizzata; in caso negativo, si sottrarrà la quantità del lotto alla quantità rimanente, e successivamente verrà impostata a zero, iterando nuovamente sul prossimo lotto disponibile.

In ciascuno dei casi viene calcolata la percentuale di materia prima utilizzata, ed in rapporto a questa, si è realizzata una stima dell'impatto dato dai soli prodotti utilizzati, ad esempio:

se ho un lotto con 10 pomodori e con impatto 20, se ho necessità di utilizzare 2 pomodori, ricavando la percentuale di 2 rispetto a 10 (20%) applicando questo 20% all'impatto totale andiamo quindi a sommare l'impatto effettivo che è 4. Essendo i prodotti però diminuiti, e l'impatto un'informazione statica e non modificabile all'interno del token, abbiamo necessità di tenere traccia della quantità di prodotto rimanente. In questo modo, se ad un'iterazione successiva abbiamo 8 pomodori (nel lotto che prima era da 10) con impatto 20(rimasto invariato) sappiamo che il prodotto rimasto è l'80%, quindi se vogliamo utilizzare 4 pomodori, andando a moltiplicare la quantità di materia richiesta per la percentuale rimanente, ed infine dividendo per quantità di materia prima rimanente nel lotto, ricaviamo la nostra percentuale (quantità richiesta*percentuale rimanente/quantità rimanente nel lotto): $4 * 80/8 = 40\%$. Questa è proprio la percentuale di materia prima rispetto alla quantità iniziale (che era 10). Per maggiori riscontri, si rimanda alla visione del codice del contratto.

Solidity mette a disposizione due costrutti molto utili per il log delle azioni all'interno della blockchain: l'**emit()** e l'**event()**. Gli eventi non sono altro che

dei messaggi che vogliamo vengano emessi sotto forma di log all'interno della blockchain, ed ogni evento può essere definito nel modo che più si preferisce.
Abbiamo implementato una serie di eventi di cui viene eseguito l'emit:

- materia prima creata
- prodotto creato
- lotto terminato
- quantità sufficiente

Questi eventi sono gestiti dal frontend che li utilizza per il salvataggio all'interno dei log, per soddisfare la linea guida definita da Sommerville relativa all' effettuare il log di ogni azione all'interno del sistema.

Al fine di rendere i contratti failsafe (OWASP), si è fatto ampio utilizzo delle funzioni *require()* con applicate varie condizioni, in modo tale da far fallire in modo sicuro una transazione che potrebbe mandare in crash la blockchain: in questo modo, la transazione viene bloccata ancor prima che questa mandi in blocco o in errore la blockchain.
Questo ci permette di evitare che le policy di sicurezza vengano violate o che alcuni dati vengano corrotti. Ad esempio, è stato applicato il require() nel momento in cui si va a controllare se le materie prime disponibili sono minori o uguali a quelle richieste, in modo tale da effettuare il revert della transazione ed inviare un messaggio d'errore nel caso in cui invece tale condizione non venga soddisfatta.

In ultimo, essendo i due contratti istanziati su due indirizzi diversi all'interno della blockchain, abbiamo bisogno di "connettere" i due contratti, in modo da permettere al contratto Transazione di sapere su che indirizzo il contratto CarbonFootprint è in esecuzione.

L'ordine di deploy individuato è questo:

1. deploy del contratto CarbonFootprint;
2. deploy del contratto Transazioni che, come input costruttore, richiede l'indirizzo del contratto CarbonFootprint (che poi userà per generare i token).

Ad esempio, nella creazione di una nuova materia prima, verrà associata l'istanza del contratto CarbonFootprint all'indirizzo specificato nel deploy del contratto Transazione.

Nella versione che abbiamo preparato per il testing, i contratti non hanno bisogno di deploy avendo progettato l'applicazione in modo tale da funzionare direttamente connettendosi ai contratti già istanziati.

3.5 Implementazione WebApp

L'applicazione parte dallo script *app.js*, all'interno del quale troviamo il codice relativo all'importazione e configurazione dei middleware definiti in precedenza. All'interno dello script troviamo anche variabili di configurazione per l'inizializzazione dell'ambiente in cui risiederà la WebApp e la gestione dei percorsi di routing.

L'app è stata configurata per operare sulla porta 5000 in modo tale da non andare in conflitto con altre possibili porte presenti sulla macchina.
Essendo un ambiente di testing, non è stato implementato il protocollo SSL/TLS per il server, ma nell'ipotesi di un rilascio al pubblico (implementando quindi una trasmissione di tipo HTTPS) sarebbe sufficiente generare le chiavi per la creazione del relativo certificato. Tale soluzione andrebbe a criptare la connessione tra client e server rendendo più difficile la decifrazione del messaggio da un eventuale attaccante.

All'interno di questo file siamo andati anche a configurare il middleware relativo alle sessioni. Tali sessioni ci permettono di salvare i dati ricavati dal database o dalla blockchain che vogliamo persistano, in modo tale da non dover eseguire altre operazioni di interrogazione al database o di ricerca all'interno della blockchain.
Per rendere più sicuro l'accesso alle sessioni viene generata una chiave criptata che viene salvata all'interno dei cookies di navigazione. Tali informazioni vengono utilizzate da *Express* ed *express-sessions* per l'**encrypting** e l'**offuscamento** della comunicazione: al programmatore viene richiesto un *secret* (che non è altro che un seed generato casualmente dal middleware crypto ad ogni avvio del server) il quale verrà utilizzato per generare la sessione. Il cookie di sessione avrà una vita massima prestabilita nel codice dalla variabile *singleSession*.

Al fine di rendere la programmazione ordinata e pulita, nel rispetto delle linee guida sulla programmazione sicura, il gruppo ha deciso di sviluppare l'applicazione secondo il pattern **MVC**.
L'applicazione, infatti, utilizza la suddivisione Model View Controller per la navigazione e la gestione delle richieste al suo interno.
Attraverso handlebars abbiamo quindi realizzato le Views che poi il Controller andrà a popolare con i dati ricavati dal Model.Sono stati predisposti 6 controller:

1. **authController**: per la gestione dell'autenticazione;
2. **blockchainController**: per la gestione e l'interfacciamento con la blockchain;
3. **crypt.js**: non è un vero e proprio controller ma viene visto come un'utility per l'encrypting;
4. **dbController**: controller relativo al model, per l'interrogazione al database;
5. **logController**: per la gestione dei vari log di sistema su file;
6. **session**: per la gestione delle funzioni relative ai dati salvati in sessione.

3.6 authController.js

È contenuto in *controllers/authController.js* e mette a disposizione tutte le funzioni utili alla registrazione ed all'autenticazione degli utenti nel portale. Il plugin mysqljs viene configurato attraverso le informazioni riportate nel file *.env* ed il metodo *defineConn* viene chiamato all'inizio del programma impostando la variabile conn in modo da essere sempre disponibile. Di seguito si elencano le funzioni implementate.

3.6.1 Register

La funzione di register permette la registrazione di un nuovo account. Ogni account richiede:

- nome,
- cognome,
- e-mail,
- password,
- conferma della password,
- tipologia dell'account (tipo di prodotti venduti, ad esempio carne o caseari).

Per rendere l'account più sicuro da possibili intrusioni, in fase di registrazione si è aggiunto il vincolo di lunghezza e difficoltà della password.
Difatti la password richiede che sia lunga almeno otto caratteri, in questo modo le possibilità di indovinare la password scendono esponenzialmente. In caso di password non combacianti o campi mancanti, il sistema riporterà un messaggio di errore relativo alla tipologia di errore.

3.6.2 Login

Come si evince dal nome, questa funzione permette l'autenticazione di un utente attraverso le credenziali di registrazione. Le credenziali inserite cercheranno riscontro in quelle salvate nel database e, nel caso in cui la password venga sbagliata troppe volte, il sistema bloccherà l'account per prevenirne attacchi di **brute force** (nel nostro caso, 5 tentativi, 10 minuti di blocco).
Essendo un'operazione di interrogazione ad una base di dati, è necessario andare a prevenire possibili attacchi di **SQL-Injection**: in questo caso si è utilizzata la formattazione della query nel formato specificatamente richiesto da mysqljs, in modo tale da occuparsi del parsing della stessa, riportando errori nel caso in cui si verifichino comportamenti sospetti.

Tra le linee guida di OWASP troviamo anche quella relativa al non fidarsi dei servizi offerti da terze parti: proprio per questo si è optato per non utilizzare servizi

di autenticazione esterni, ma si è realizzato un database sulla macchina che hosta la webapp. Quindi sono stati scartati servizi di login attraverso vari social quali, ad esempio, Google o Facebook.

3.7 BlockChainController.js

Attraverso questo script, andiamo a gestire tutte quelle funzionalità che sono blockchain-related connettendoci ad un'istanza dei contratti di cui è stato fatto il deploy in precedenza. Gli indirizzi delle istanze sono riportati nel file *.env* insieme alle altre informazioni di configurazione.
La connessione viene instaurata andando a connettersi alla blockchain attraverso la porta *WebSocket*[20] del nodo principale della rete. È stata scelta la **WS** rispetto alle RPC[21], dato che queste ultime non permettono la sottoscrizione agli eventi e sono più aggiornate rispetto allo standard RPC. Le informazioni riguardanti gli indirizzi delle porte sono anch'essi contenuti all'interno del file *.env*.
Sono state implementate una serie di funzioni che permettono di richiamare le funzioni associate sullo smart contract di interesse, nello specifico:

- nuovo account (nuovo portafogli);
- sblocco account;
- lista materie prime possedute;
- lista materie prime possedute da un determinato indirizzo (utile in fase di acquisto);
- acquisto/creazione materia prima;
- acquisto/creazione prodotto;
- consultazione Carbon Footprint.

Ciascuna di queste funzioni è stata trattata in precedenza all'interno del contratto, questa è solo l'interfaccia javascript che manipola le informazioni ricavate dal frontend per eseguire e richiamare le funzioni presenti nei contratti.
Tali funzioni sono state anche racchiuse in vari blocchi try-catch, che ci permettono di gestire le eventuali eccezioni generate dal codice o dal revert di una transazione all'interno della blockchain (fallire in modo sicuro, OWASP). Nel caso specifico di una transazione fallita, la gestione dei revert può essere abilitata andando a settare la variabile d'ambiente di Web3 *handleRevert=true*.

Le varie transazioni da parte dell'utente verso la blockchain sono asincrone. Bisognerà quindi attendere che il blocco contenente la transazione venga scritta dal nodo principale e che questo blocco venga approvato. IBFT ha un tempo intrinseco

[20]Maggiori informazioni: `https://datatracker.ietf.org/doc/html/rfc6455`
[21]Maggiori informazioni: `https://datatracker.ietf.org/doc/html/rfc1057`

di generazione dei blocchi, ed è stato calcolato attraverso varie misurazioni che questo tempo è circa 2 sec. Questo tempo sarà quindi il tempo massimo per l'elaborazione della transazione.

In precedenza si è accennato all'implementazione degli eventi. È sempre all'interno di questo script che si è scritto il codice relativo all'intercettazione dei vari eventi che il contratto emette: questi eventi vengono intercettati, ne viene effettuato un parsing e la risultante viene riportata all'interno di alcuni file di log che verranno trattati nella parte relativa al logger.

3.7.1 Crypt.js

Questo script consente di effettuare funzioni di encrypting e decrypting di stringhe. La scelta di utilizzare questo modulo è data dalla necessità di rendere confidenziali gli indirizzi.
Durante la registrazione di un utente, viene creato un nuovo portafoglio con relativo indirizzo, quest'ultimo viene criptato attraverso l'algoritmo **Rabbit** e salvato all'interno del database. Durante l'operazione di login, i dati dell'utente vengono caricati in sessione. Tuttavia si è deciso di mantenere in sessione l'indirizzo criptato andando a decriptarlo nel momento in cui serve, rendendolo **unpersistent**(volatile): così facendo, l'utente malevolo che vuole intercettare i dati di sessione, troverà comunque l'indirizzo criptato, rendendo così più difficile la perdita di dati.

3.7.2 dbController.js

Questo script provvede ad interfacciarsi con il database, per andare a popolare il frontend caricando i dati relativi delle risorse (prodotti, materie prime) sulla base della categoria di utente corrente all'interno della sessione.
Ad esempio, il produttore può solo creare materie prime, il lavoratore può sia comprarle che lavorarle ed il cliente invece può solamente acquistare prodotti. Attraverso la tipologia di risorse dell'utente salvata nella sessione si è andati a cercare nel database la categoria di prodotti e materie prime associate.

3.7.3 logController.js

Il logging viene effettuato dal plugin **Winston**, introdotto in precedenza. Questo script gestisce ogni tipo di log, sia off-chain che on-chain.
Winston mette a disposizione un plugin per generare log a più livelli, facendo riportare l'informazione nel livello scelto ed in tutti quelli inferiori. Sono stati implementati due tipi di logger:

- off-chain;
- on-chain.

Per quanto riguarda l'off-chain il logger prende il nome di *ActionLogger*, il quale prevede due livelli di logging: le azioni(actions, livello 0) e gli errori (errors, livello 1). Per fare un esempio, se un utente effettua il login compie un'azione, che verrà loggata in actions ma non negli errori, e se un utente compie un errore, questo verrà salvato all'interno degli errori ma anche come azione.
Analogamente è stato effettuato per quanto riguarda l'on-chain. Quattro livelli, errore(livello 3) blockchain(livello 0) token (livello 1) transazione (livello 2).

3.7.4 Session.js

Il controller di sessione si occupa di fornire alle pagine i dati che sono stati ricavati dal model(database o blockchain). È lui che comunica con le viste e, di conseguenza, le popola.
All'interno di questo script troviamo tutte le funzioni relative ai *getters* ed ai *setters* per le variabili di sessione relative all'utente al momento loggato. Lo script è di fondamentale importanza in quanto attraverso semplici controlli basati su degli *if-else*, riusciamo ad evitare attacchi del tipo *privilege escalation*.
Difatti, per ogni pagina sono stati aggiunti controlli atti a verificare se l'utente abbia effettivamente il permesso di accedere a quella pagina e se i contenuti di quella pagina sono visibili o meno dalla sua categoria di utenza. Le pagine della web app, quindi, cambiano dinamicamente a seconda della categoria di utente che ne richiede l'accesso ed a seconda della tipologia di risorse che egli tratta.
Soddisfacendo queste policy, andiamo quindi a garantire un corretto information flow all'interno di tutta l'applicazione, garantendo che flussi di alto livello rimangano di alto livello, e quelli di basso livello rimangano di basso livello.
Nel caso in cui un utente provi ad accedere comunque ad una pagina per cui non ha il permesso lo script mette a disposizione una serie di messaggi di **warning** e di **errore** che sono stati opportunamente applicati dal programmatore. In questo modo è possibile avvertire l'utente sbadato o malevolo che l'azione che sta compiendo non è lecita/autorizzata.

3.7.5 Views

Le viste sono state realizzate attraverso **HTML** unito al motore **HBS**, che permette di aggiungere una logica al codice HTML potendo effettuare controlli sulle variabili e consentendo riutilizzo del codice usando i vari *partials* (pezzi di codice incapsulati all'interno di file denominati partials). Le pagine di layout vengono richiamate come scheletro della web app. Lo scheletro cambierà a seconda delle informazioni relative all'utente ed i vari controlli posti nel gestore delle pagine *pages.js* filtreranno i contenuti.

3.8 Testing e Collaudo

Per garantire il funzionamento corretto del software esso è stato sottoposto al testing da parte di tutti i membri del gruppo, al fine di avere un campione di dati eterogeneo da cui eventualmente trovare gli errori.

Sono stati suddivisi compiti all'interno del gruppo per quanto riguarda lo sviluppo del codice, assegnando vari task ai singoli componenti del gruppo. Lo sviluppo è stato completato attraverso una repository github alla quale i componenti del gruppo hanno contribuito scrivendo la loro parte di codice. Ciascuno, singolarmente, ha testato la propria parte effettuando successivamente un push in modo tale da rendere disponibile la propria soluzione anche agli altri membri che a loro volta testeranno la soluzione in cerca di possibili bug.

Tuttavia, per alcuni script non è stato possibile lavorare contemporaneamente (come i contratti). Per questo motivo sono state effettuate alcune riunioni all'interno delle quali si è discusso di come affrontare determinati problemi. Abbiamo delineato quattro collaudi totali riguardanti:

1. frontend;
2. backend;
3. contratti;
4. securing del codice (aggiunta dei controlli ulteriori per evitare errori ed attacchi).

In ciascuno di questi collaudi, il gruppo di progetto si è riunito per testare la soluzione appena realizzata, verificando che tutto fosse in regola ed, eventualmente, sistemando gli eventuali bug. Ciò ci ha permesso di arrivare ad una soluzione funzionante del programma, pronto per il suo utilizzo.

Capitolo 4

Considerazioni finali

4.1 Considerazioni finali

Il lavoro svolto soddisfa appieno le specifiche progettuali assegnateci in fase di definizione del progetto.
Molte sono le funzionalità implementate: la web app è completamente modulare e questo permette la facile aggiunta di middleware aggiuntivi atti ad aggiungere altre funzionalità. Tuttavia alcune funzionalità, come ad esempio l'autenticazione a due fattori, non sono state inserite a causa delle risorse limitate a disposizione.
Le funzionalità inserite permettono un utilizzo ampiamente sicuro ed al riparo da possibili attacchi provenienti entità malevoli o da possibili errori compiuti da utenti sbadati.
Tutti i dati sono presenti in duplice copia sia all'interno della blockchain che all'interno di un registro di sistema composto dai log delle operazioni; potendo accedere alle informazioni anche in caso di sistema irraggiungibile.

Nel codice sono presenti porzioni sicuramente ridondanti di controlli, ma garantiscono una maggiore sicurezza dal punto di vista di possibili intrusioni.
Abbiamo infatti diversi layer:

- layer HTML;
- layer HBS;
- layer javascript;
- layer solidity (per operazioni nella blockchain).

I dati inseriti devono ottenere approvazione da tutti questi layer, rendendo così più difficile un possibile attacco da un utente malintenzionato che, sfondando il layer HTML ed HBS (e. g.), viene bloccato dal layer javascript il quale manda in errore il programma gestendo l'eccezione.

Dove possibile sono state riportate tutte le pratiche di buona programmazione tratte dalle molteplici linee guida fornite; tuttavia non sono state riportate tutte

vista la grandissima mole di codice scritto. Per questo motivo si consiglia di cercarne riscontro all'interno del codice allegato alla relazione.

Infine, allegata al software ed a questa relazione, è presente una breve guida all'utilizzo del software.

Elenco delle figure