Università Politecnica delle Marche
Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Progettazione e implementazione di algoritmi per la generazione di Smart Contract integrati nel sistema IntegrityKey

Design and implementation of algorithms for the generation of Smart Contracts integrated in the IntegrityKey system

Relatore:
Prof. Simone Fiori

Candidato:
Nicola Picciafuoco

Tesi di laurea Triennale
A.A. 2022/2023

Sommario

La presente tesi è stata sviluppata a partire dall'attività di tirocinio svolto presso una Startup innovativa, IntegrityKEY s.r.l, spin off dell'Università Politecnica delle Marche, che opera all'interno del comparto agroalimentare del centro Italia fornendo servizi di tracciabilità e di rintracciabilità dei prodotti per le piccole e medie imprese. L'attività, di tipo progettuale, è consistita nella definizione e sviluppo di un algoritmo per la generazione di contratti intelligenti (Smart Contract), cioè contratti auto-esecutivi non modificabili con delle regole di accordo tra le due parti direttamente scritte in codice informatico, la cui sicurezza è garantita grazie all'utilizzo di tecnologia "blockchain".
Il lavoro è stato articolato in due distinte fasi:

- la prima fase si è concentrata sulla selezione della blockchain più idonea per la realizzazione del progetto. In questa fase si è svolta un'analisi dettagliata dei costi e della fattibilità del progetto, prendendo in esame diversi tipi di blockchain tra cui Etherum, Cardano, Polkadot, Algorand, blockchain europea.

- la seconda fase ha riguardato lo sviluppo effettivo di uno Smart Contract, implementato sulla piattaforma blockchain Algorand con l'obiettivo di tracciare l'intero percorso dei prodotti agroalimentari lungo la filiera, dal produttore al consumatore finale. La generazione dello Smart Contract è poi stata resa semplice tramite lo sviluppo di una specifica App tramite Django, un framework web open-source ad alto livello, scritto in Python, adatto allo sviluppo veloce di applicazioni web con un design pulito e pragmatico.

Questo progetto è stato concepito come un Prototipo di Concetto (P.O.C.) ed è stato intrapreso con l'obiettivo primario di indagare e valutare la sua fattibilità. Come si vedrà nei vari capitoli dell'elaborato, la programmazione di uno Smart Contract ha richiesto conoscenze e abilità variegate, l'assimilazione di nuovi concetti e l'addestramento all'utilizzo di strumenti informatici totalmente nuovi, non risultando sufficiente la conoscenza del solo linguaggio di programmazione. Pertanto, la ricerca approfondita e l'assimilazione delle modalità d'uso delle sofisticate tecnologie informatiche che sono state utilizzate ha assorbito larga parte del lavoro di tesi. Il cuore dell'attività non è quindi risultato tanto nella mera implementazione dell'applicazione web, quanto piuttosto nella complessa ricerca e selezione delle componenti adeguate ad ottenere un funzionamento sinergico e coeso.

Indice

CAPITOLO 1

Premesse Teoriche

In questo capitolo vengono spiegate in modo dettagliato le tecnologie utilizzate, con il fine di rendere chiaro il lavoro svolto. Inoltre, vengono discussi in modo teorico gli elementi trattati e vengono mostrati i riferimenti alle fonti da cui sono state tratte le informazioni principali. In un mondo in cui la tecnologia avanza a ritmi vertiginosi, è essenziale fermarsi un attimo e analizzare attentamente gli strumenti che usiamo, non solo dal punto di vista pratico, ma anche teorico. Per ogni tecnologia o strumento menzionato, verrà fornita una spiegazione complessiva, che comprende sia il funzionamento che le applicazioni principali. Ciò contribuirà a creare una visione di insieme e comprensiva del contesto in cui sono state adottate le scelte tecniche.

1.1 Blockchain

La blockchain può essere paragonata a un registro contabile digitale distribuito. Consiste in una serie di blocchi, ognuno dei quali contiene un gruppo di transazioni. Queste catene di blocchi sono concatenate tra loro in ordine cronologico, assicurando che ogni blocco successivo contenga un riferimento criptografico, chiamato hash, al blocco precedente. Questa struttura, insieme alla crittografia, rende estremamente difficile, se non impossibile, modificare retroattivamente le informazioni registrate sulla blockchain. La natura distribuita della blockchain significa che molteplici copie del registro sono mantenute da vari nodi in una rete, garantendo trasparenza e resistenza alla censura.

Le blockchain possono essere pubbliche, come Bitcoin o Ethereum, o algorand, dove chiunque può partecipare alla rete e vedere tutte le transazioni, oppure private, dove l'accesso è limitato a un gruppo specifico di persone o organizzazioni.

1.1.1 Tipi di tecnologie

Proof of Work

Il Proof of Work è un meccanismo utilizzato da molte blockchain per garantire la sicurezza delle transazioni e prevenire attacchi malevoli. I nodi della rete, chiamati minatori, risolvono problemi matematici complessi per convalidare e registrare le transazioni su un nuovo blocco. Questo processo, pur essendo fondamentale per la sicurezza della rete, è altamente energivoro poiché richiede considerevole potenza di calcolo. Il primo minatore che risolve il problema ottiene il diritto di aggiungere il nuovo blocco alla blockchain e viene ricompensato con la criptovaluta corrispondente.

Proof of Stake

Il Proof of Stake è un'altra metodologia per garantire la sicurezza e la convalida delle transazioni in una blockchain. Invece di risolvere enigmi matematici, come nel PoW, i validatori nel PoS sono scelti in base alla quantità di criptovaluta che "mettono in gioco" o bloccano come garanzia. Se un validatore propone un blocco malevolo, perde la sua posta. Ciò riduce notevolmente il bisogno di potenza di calcolo, rendendo il PoS molto più efficiente dal punto di vista energetico rispetto al PoW.

In sintesi, mentre il PoW e il PoS sono entrambi meccanismi per garantire l'integrità delle blockchain, differiscono nel loro approccio e impatto. Il PoW si basa sulla potenza di calcolo per convalidare transazioni, con un notevole consumo energetico. D'altro canto, il PoS si basa sulla fiducia e sulle risorse monetarie messe in gioco dai validatori, risultando molto più efficiente dal punto di vista energetico.

1.2 Smart Contract

Uno Smart Contract è un contratto auto-esecutivo con delle regole di accordo tra due parti direttamente scritte in codice informatico. Esso risiede e opera all'interno di una blockchain. Il punto di forza degli Smart Contracts è che sono inalterabili e decentralizzati, il che significa che una volta impostati, nessuna parte può modificarli unilateralmente, e non c'è bisogno di un intermediario per garantire o eseguire il contratto. Questo riduce i rischi di frodi, manomissioni e interruzioni.

1.2.1 Funzionamento

Alla base, il funzionamento di uno Smart Contract è simile a un programma tradizionale: ha variabili, funzioni e può avere condizioni come "se-questo-allora-quello"[1]. Tuttavia, a differenza di un programma normale, una volta che uno Smart Contract è stato implementato[2] su una blockchain, non può più essere modificato. Se esistessero eventuali errori nel codice, questi potrebbero causare gravi problemi, come perdite di fondi, rendendo cruciale la fase di verifica e test prima della pubblicazione.

1.2.2 Utilizzi

Gli Smart Contracts hanno una vasta gamma di applicazioni che vanno ben oltre le semplici transazioni di criptovaluta, tutti gli esempi riportati esistono realmente e vengono già utilizzate:

- Gestione della Supply Chain: possono tracciare prodotti lungo tutta la catena di fornitura, assicurando l'autenticità e l'origine dei prodotti.[3]
- Immobiliare: automatizzazione delle vendite e degli affitti, rendendo il processo più veloce e trasparente.
- Voto: creazione di sistemi di voto sicuri e trasparenti.
- Assicurazioni: automatizzazione dei pagamenti delle polizze in base a eventi predefiniti.
- Giochi e Applicazioni Decentralizzate (DApp): creazione di giochi basati su blockchain o piattaforme social con regole definite dagli Smart Contracts.

[1] più comunemente conosciuto con if/else

[2] Spesso viene usato il termine deployato. Il termine "deployare" è un anglicismo derivato dal verbo inglese "to deploy", che in contesti informatici significa mettere in funzione un'applicazione, un sistema o un nuovo servizio, solitamente trasferendolo da un ambiente di sviluppo/test a un ambiente di produzione dove può essere accessibile agli utenti finali.

[3] approfondimento su realtà che già esistono: `https://cryptonomist.ch/2023/08/23/parmigiano-reggiano-certificato-blockchain/` anche sull'alta moda `https://cryptonomist.ch/2020/01/28/textilechain-moda-blockchain-made-in-italy/`

1.2.3 Vantaggi

Questa tecnologia vanta di numerosi vantaggi:

- Trasparenza: tutte le parti possono vedere i dettagli e l'esecuzione dello Smart Contract sulla blockchain.
- Sicurezza: la blockchain è sicura e resistente alle manomissioni.
- Risparmio: eliminazione degli intermediari e automatizzazione dei processi.
- Velocità: i processi automatici sono generalmente più rapidi dei processi manuali.

1.2.4 Svantaggi

Tuttavia, ci sono anche delle sfide. La principale è che gli Smart Contracts sono "immutabili", il che significa che, una volta pubblicati, non possono essere facilmente modificati o annullati. Questo ha portato a incidenti nei quali sono stati scoperti errori nei contratti, con conseguente perdita di milioni di dollari. Inoltre, la programmazione degli Smart Contracts richiede competenze specializzate, poiché gli errori possono avere conseguenze finanziarie significative.

In conclusione, gli Smart Contracts rappresentano una potente innovazione che ha il potere di rivoluzionare molte industrie, automatizzando processi e riducendo la necessità di intermediari. Tuttavia, come con ogni nuova tecnologia, vengono con le loro sfide e richiedono una profonda competenza per essere implementati correttamente.

1.3 Algorand

Algorand rappresenta una significativa evoluzione nel panorama delle tecnologie blockchain. Fondata dal rinomato criptografo Silvio Micali nel giugno del 2019, professore presso il MIT e vincitore del prestigioso premio Turing, questa piattaforma è stata ideata con l'intento di risolvere alcune delle più pressanti sfide affrontate dalle blockchain.

1.3.1 Storia e Fondazione

Lanciata nel 2019, la visione di Algorand nasce dalla percezione delle limitazioni delle attuali blockchain, in particolare in termini di scalabilità, sicurezza e decentralizzazione. Con un approccio innovativo al consenso basato su Proof of Stake Puro (PPoS), Algorand si propone come una soluzione robusta a questi problemi.

1.3.2 Caratteristiche Tecniche

- Proof of Stake Puro (PPoS): a differenza del tradizionale Proof of Stake, il PPoS di Algorand opera secondo un principio di parità: ogni token ha lo stesso valore degli altri e il detentore di ciascun token possiede le medesime probabilità di essere selezionato per l'aggiunta di un nuovo blocco alla blockchain. Questo sistema, eliminando la necessità del Proof of Work, permette ad Algorand di ridurre notevolmente l'energia impiegata, assicurando al contempo una rapida elaborazione delle transazioni. Più nel dettaglio, per l'aggiunta di un blocco è necessario il consenso di due terzi dei token: una volta che tale consenso viene raggiunto dalla maggioranza qualificata, il nuovo blocco viene aggiunto in maniera univoca alla catena, prevenendo così eventuali biforcazioni.

- Velocità e Scalabilità: con capacità di elaborare di oltre 1000 transazioni al secondo, Algorand si posiziona tra le blockchain più performanti, offrendo un throughput comparabile a quello di grandi società di pagamento, ma mantenendo i vantaggi della decentralizzazione.

- Sicurezza Avanzata: l'architettura di Algorand pone una notevole enfasi sulla sicurezza delle transazioni, sfruttando le ultime ricerche ed iniziative nel campo della crittografia.

- Decentralizzazione Autentica: algorand opera in modo che non vi siano nodi o miner dominanti, garantendo una vera democratizzazione del processo di consenso.

La scelta dei validatori

Silvio Micali illustra il funzionamento del sistema di validazione di Algorand evidenziando un approccio rivoluzionario: ogni volta, un nuovo set di validatori viene determinato in maniera aleatoria da un algoritmo tra tutti i detentori di token. Questo metodo assicura un alto grado di decentralizzazione, poichè ogni possessore di token ha la stessa probabilità di essere selezionato come validatore. Allo stesso tempo, il sistema garantisce sicurezza: non è possibile anticipare o prevedere chi saranno i prossimi validatori, impedendo così potenziali manipolazioni o corruzioni.

Quando si tratta di aggiungere un nuovo blocco alla blockchain, il processo diventa ancor più interessante. Tramite un meccanismo di selezione randomizzato, vengono scelti mille token tra tutti quelli esistenti, e, di conseguenza, i loro proprietari, per costituire un comitato di validazione. Micali esemplifica: "Supponiamo di possedere un determinato numero di token e che, in un dato momento, due dei tuoi token vengano selezionati. Ciò significa che avrai due voti all'interno di questo comitato." I membri del comitato decidono poi consensualmente sul contenuto del nuovo blocco. La bellezza di questo sistema risiede nella sua natura dinamica: per ogni blocco da validare, il comitato di validatori cambia, mantenendo così la rete sempre decentralizzata e imprevedibile.

1.3.3 Pyteal

Algorand, come molte piattaforme blockchain moderne, supporta diversi linguaggi di programmazione per permettere agli sviluppatori di interagire con la sua rete e creare soluzioni personalizzate. La versatilità di Algorand in questo senso è notevole e va dalla creazione di smart contracts alla gestione di transazioni.

Uno degli aspetti distintivi di Algorand è l'introduzione di TEAL, un linguaggio assembly-like per scrivere logiche di transazione per la rete. Sebbene TEAL sia potente, la sua natura a basso livello può renderlo meno accessibile o intuitivo per alcuni sviluppatori, specialmente per coloro che provengono da un background di linguaggi di alto livello. Successivamente per ovviare questa problematica è stato introdotto PyTeal.

PyTeal è un compilatore che permette agli sviluppatori di scrivere logica di SmartContract in un linguaggio che assomiglia molto al Python, rendendolo molto più leggibile e familiare. Una volta scritto, il codice PyTeal viene compilato in TEAL, rendendolo pronto per essere eseguito sulla rete Algorand. Questo offre il meglio di entrambi i mondi: la semplicità e la chiarezza di un linguaggio di alto livello, con le potenzialità e l'efficienza di un linguaggio di basso livello come TEAL.

L'adozione di strumenti come PyTeal dimostra l'impegno di Algorand verso l'apertura e l'accessibilità. Gli sviluppatori non devono più compromettere tra la leggibilità del codice e le prestazioni, potendo sfruttare la facilità d'uso di Python e la potenza di TEAL contemporaneamente.[4]

In sintesi, mentre TEAL fornisce una fondazione solida per la creazione di logiche di transazione su Algorand, PyTeal amplia ulteriormente le capacità della piattaforma, offrendo agli sviluppatori un ambiente di programmazione più amichevole e intuitivo, ma senza sacrificare le potenzialità offerte dalla rete.

1.3.4 Beaker

Beaker è un framework Python per la creazione di contratti intelligenti su Algorand utilizzando PyTeal. Framwork molto giovane, infatti il progetto è ancora in fase di sviluppo attivo, e richiede per l'installazione un python di livello 3.10 o superiore.

Quando si scrivono contratti intelligenti e si usa pyteal viene migliorata l'organizzazione del codice rendendola più familiare per gli sviluppatori python. Rende più intuitiva la distribuzione e la creazione dello smart contract utilizzando l'Algorand Sdk, spiegata nel capitolo1.4, inoltre offre diversi pacchetti che estendono Pyteal che forniscono funzionalità per testare e distribuire smart contract.

[4] In aggiunta si evidenzia un evidente propensione a formare nuovi programmatori fornendo Tutorial gratuiti in inglese in live o anche registrazioni del precedentemente es: `bit.ly/3YAecm7`

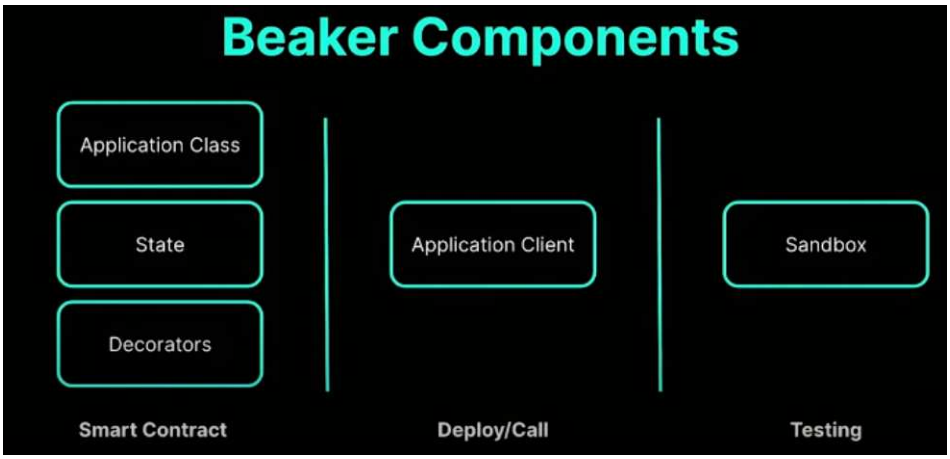

Figura 1.1: Componenti Beaker

In altre parole Beaker rende l'organizzazione degli smart contract più familiare, aiutandoti a definire metodi per te; così da evitare di inserirli manualmente, rendendo più semplice distribuire e creare smart contract[5]

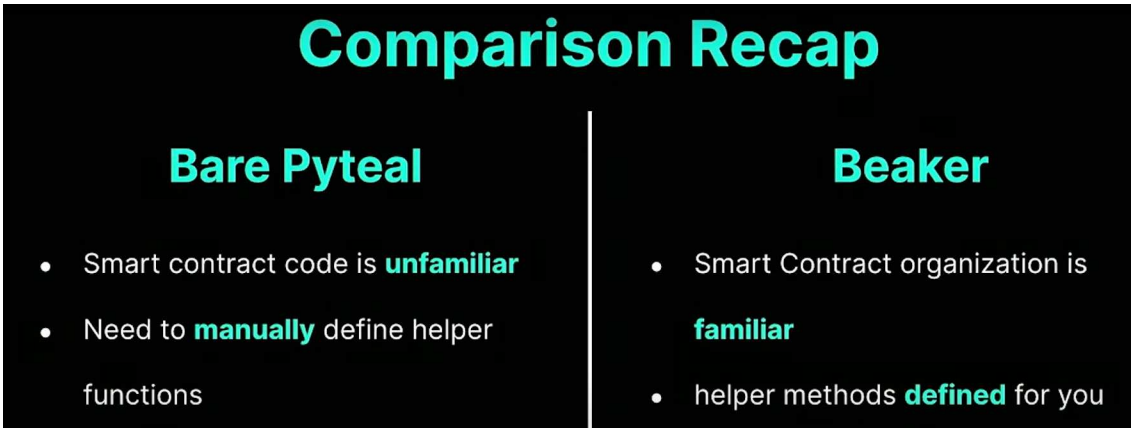

Figura 1.2: Differenze fondamentali

1.4 AlgoSDk

Per facilitare lo sviluppo di applicazioni e smart contract che interagiscono con la rete Algorand, è disponibile un Software Development Kit (SDK), ossia un insieme di strumenti e librerie che agevolano la programmazione di applicazioni.

Algorand offre SDK in vari linguaggi di programmazione come Python, Go, JavaScript, Java e altri. Questi SDK permettono di fare una vasta gamma di operazioni come:

[5]Per approfondire, presentazione di Beaker dalla Algorand Foundation `https://github.com/algorand-devrel/beaker`

- Creare nuovi account e gestire chiavi crittografiche.
- Costruire, firmare e inviare transazioni.
- Interrogare il ledger[6] per ottenere informazioni sulle transazioni, i blocchi, ecc.
- Creare e gestire Asset Standard Algorand (ASA), che sono token personalizzati sulla blockchain Algorand.
- Creare e interagire con gli smart contract.

L'utilizzo dell'SDK semplifica notevolmente lo sviluppo su Algorand, rendendo molto più accessibile la blockchain per sviluppatori con diversi livelli di esperienza.

A questo punto abbiamo visto tutti i componenti necessari per l'ambiente di sviluppo.

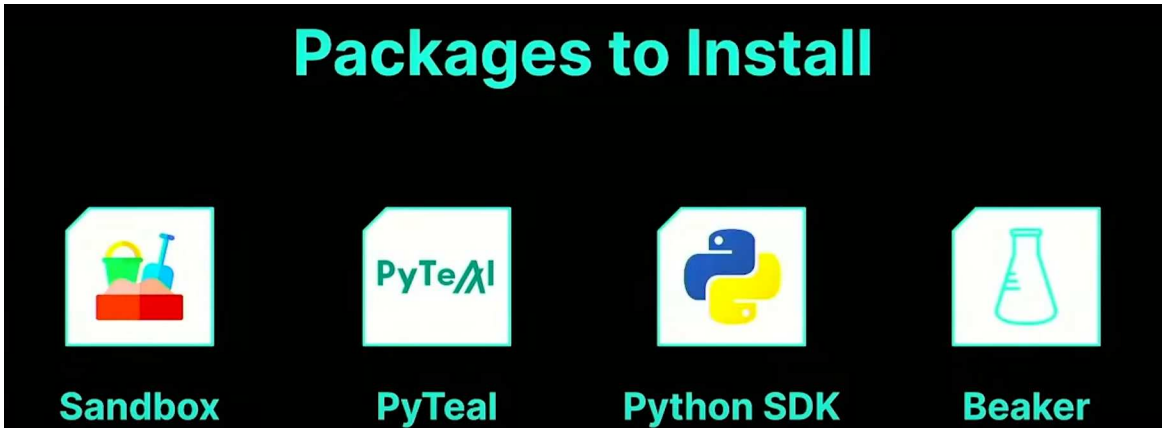

Figura 1.3: Packages da installare

1.4.1 Applicazioni

Algorand non è soltanto una piattaforma per transazioni di criptovaluta. La sua architettura permette lo sviluppo di applicazioni decentralizzate (DApps) sofisticate, tokenizzazione di asset, SmartContract e soluzioni di identità digitale. Queste funzionalità possono avere implicazioni rivoluzionarie in settori come la finanza, la supply chain, il settore pubblico e molti altri.

1.4.2 Costi

Algorand è leader per efficienza economica e velocità, gli utenti beneficiano di tariffe a transazione ridotte, rendendo possibile condurre un maggior numero di operazioni a una frazione del costo che si avrebbe su altre piattaforme blockchain. Questa economicità è fondamentale

[6] o altri cold wallet

per le piccole e medie imprese, per le startup e per chiunque desideri utilizzare la blockchain in modo intensivo senza doversi preoccupare di costi proibitivi.

Inoltre, un costo di transazione ridotto non significa compromessi in termini di velocità o sicurezza. Algorand è stato progettato per garantire transazioni veloci, sicure e a basso costo, fornendo comunque un equilibrio ottimale tra efficienza economica e performance tecnica.

Per fare un esempio si può portare un immagine presa direttamente da Explorer ufficiale Da

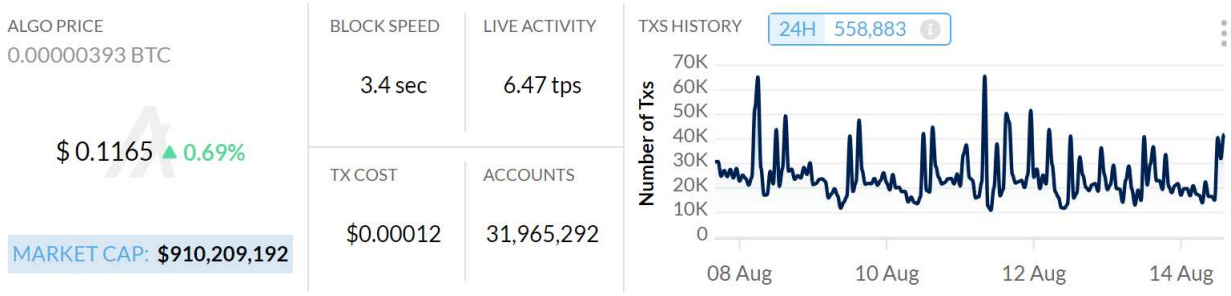

Figura 1.4: fotogramma preso il 14/08/2023

questa si evince come il costo medio per ogni singola transazione sia irrisorio e che ha velocità media dei blocchi(calcolata negli ultimi 10 blocchi) sia di 3.4 secondi.

Link utili

- Repository GitHub: `https://github.com/algorand`
- Sito ufficiale: `https://www.algorand.com`
- Explorer: `https://algoexplorer.io`

1.5 Docker

Il prototipo richiesto dall'azienda è stato sviluppato completamente in locale, essendo un P.O.C [7], quindi è stato utilizzato Docker che in sostanza è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Docker raccoglie il software in unità standardizzate chiamate container che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito. In questo caso è stato installato nella macchina locale, non su un server esterno, con appositi pachetti per realizzare una testnet locale, cosi da essere in grado di deployare su tesnet i vari smart contract creati.

[7]Si utilizza questa dicitura per intendere una realizzazione incompleta o abbozzata di un progetto, con il file di provarne la fattibilità e dimostrarne la fondatezza.

Come spiegato nella sezione 1.3.3, vi sono in vari tutorial realizzati dal team algorand, così da rendere lo sviluppo più semplice, anche se non sono mancati vari problemi di compatibilità con il sistema locale. [8]

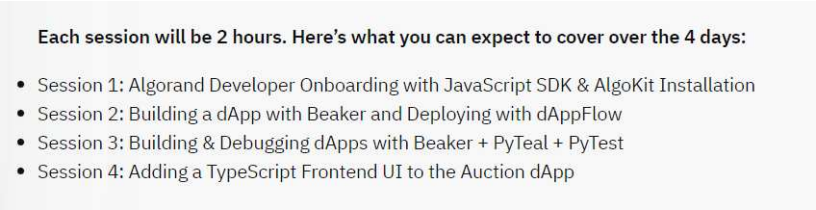

Figura 1.5: Programma seguito per Algorand Developer Bootcamp

1.6 Algorand Sandbox

Al fine di sviluppare questo P.O.C è stato necessario avvalersi di un ambiente dove poter testare la creazione di smart contract, per far questo è stato necessario usare docker. Docker è già stato ampiamente spiegato precedentemente dal punto di vista teorico, in questo capitolo verrà introdotta una spiegazione per il settaggio di Algorand Sandbox. Tutte le informazioni di settaggio sono state prese da `https://github.com/algorand/sandbox`, e dal bootcamp ufficiale di Algorand [9].

Cos'è il Sandbox Algorand?

Il sandbox di Algorand è un ambiente preconfigurato che consente di eseguire una rete Algorand su un computer locale (Testnet locale). È uno strumento utile per sviluppatori e tester che desiderano sperimentare le funzionalità di Algorand senza dover configurare una rete da zero, inoltre la rete principale (Mainet) ha alcuni costi per transazioni e sviluppo di smart contract, cosa che si può evitare utilizzato una localnet o la tesnet[10].

Installazione con Docker

Docker è uno strumento che permette di containerizzare applicazioni, e il sandbox di Algorand sfrutta Docker per eseguire tutti i servizi necessari. Di seguito verrà presentata una

[8] Risulta possibile vedere alcune registrazioni delle live anche su youtube `https://www.youtube.com/watch?v=5O1-mqCGcc0`, in questa prima lezione viene spiegato come settare l'ambiente di sviluppo e scaricare la repository ufficiale, viene consigliato Gitpod, mentre in questo progetto è stato utilizzato inizialmente Vs-code per poi passare a PyCharm per integrare tutto il sistema con Django

[9] Argomento trattato in 1.5, dove si possono trovare i vari link, anche alle lezioni bootcamp ufficiali registrate.

[10] Sono state svolte delle prove su testnet ufficiale di Algorand, con deploy di smart contract, ma è stato deciso per un maggior controllo di adoperare in rete locale, di seguito link di uno smart contrat deployato, visibile a tutti, anche se privo di senso `https://testnet.algoexplorer.io/tx/Q5RUEJQ4ELTOMPBRUW27DV56PWN2ATIFFACA46HSYVQ2U4IBNDPA`

piccola guida:

- Installare Docker, possibile trovare una guida al sito ufficiale `https://www.docker.com/get-started/`

- Clonare il repository: Una volta installato Docker, apri una shell e clona il repository del sandbox di Algorand da GitHub con il comando:
  `git clone https://github.com/algorand/sandbox.git`
  Entrare nella directory del sandbox:
  `cd sandbox`
  Avviare il Sandbox:
  `./sandbox up`
  Questo comando avvierà i container Docker necessari per eseguire il sandbox di Algorand. Puoi anche specificare una versione particolare se necessario.

- Interazione con il sandbox: Una volta che il sandbox è in esecuzione, puoi utilizzare diversi comandi per interagire con esso, come invio di transazioni, deploy di smart contract, ecc.

Questi sono i passaggi di base che sono stati effettuati per lo sviluppo del progetto, è bene ricordare che le versioni possono essere aggiornate e mutate, di conseguenza leggere e seguire le istruzioni del Readme è sicuramente la scelta migliore.

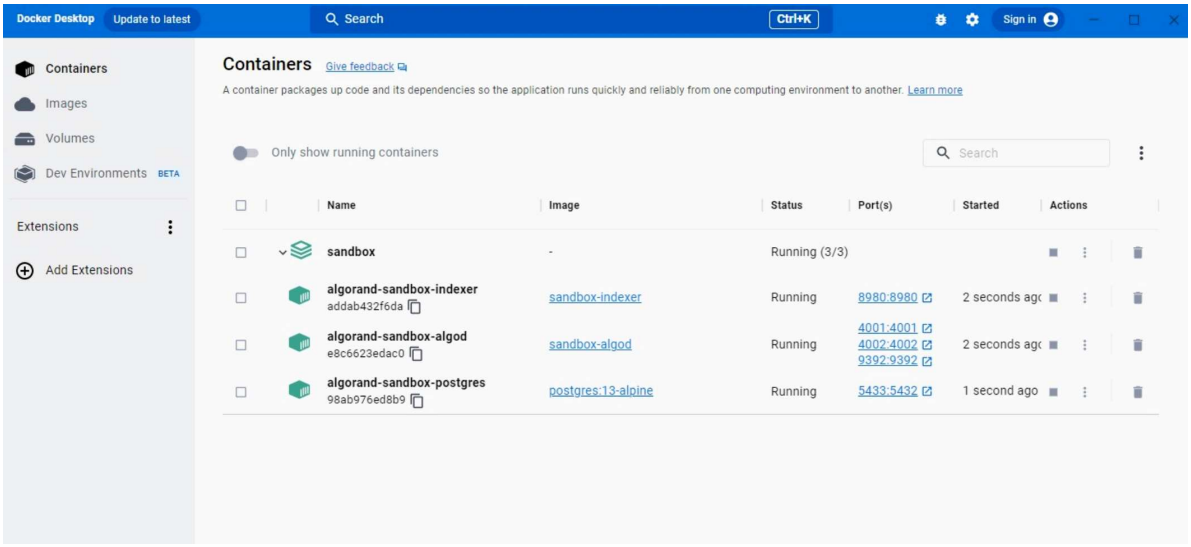

Figura 1.6: Pacchetti installati su Docker

1.7 Dappflow

Dappflow è un set di strumenti di sviluppo per la piattaforma Algorand, tramite il quale è possibile lo sviluppo su Algorand, fornendo strumenti potenti per costruire, distribuire e testare le tue applicazioni decentralizzate (dApp) con facilità.

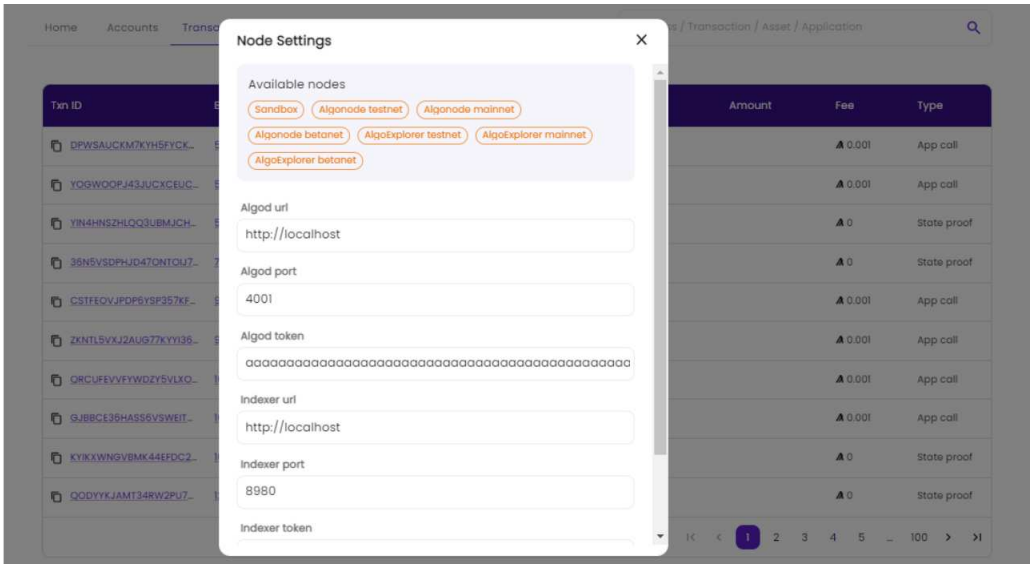

Figura 1.7: Settaggi per il nodo di sandbox

Grazie a questo potente strumento è possibile vedere cosa accade alla tua Blockchain locale, andando a vedere transazioni, e Smart Contract sviluppati sulla rete.

Figura 1.8: Esempio di output dello smart contract visibile tramite dappflow

Questo strumento è fondamentale, dandoti la piena visione di quello che accade e di quello che lo sviluppatore sta realmente facendo, tramite semplici passaggi, illustrati nei tutorial, Dappflow riconosce l'ambiente Sandbox creato nel tuo dispositivo e va a collegarsi permettendo

una chiara e semplice visione di quello che accade, tutto in maniera gratuita. Per ulteriori informazioni è possibile consultare il loro sito `https://dappflow.org/`.

1.8 Django

Per lo sviluppo dell'app è stato utilizzato il framework usato dall'azienda IntegrityKey, in modo da rendere il codice integrabile, per questo è stato sviluppato l'algoritmo che può essere utilizzato tramite un' interazione nel sistema locale, con l'interfaccia di base generata da Django. Andando nel dettaglio Django è un framework web ad alto livello, scritto in Python, che promuove lo sviluppo veloce di applicazioni web, privilegiando un design pulito e pragmatico. La sua filosofia si basa sul principio del "Don't repeat yourself" (DRY), che incoraggia la riutilizzazione del codice e riduce la duplicazione attraverso una struttura modulare e un sistema di "app" riusabili.

1.8.1 Origini

Django è nato nel 2003, sviluppato in origine dal team web del "Lawrence Journal-World", un giornale del Kansas, come soluzione interna per gestire le sfide dello sviluppo web. Fu rilasciato al pubblico come software open-source nel 2005.

1.8.2 Architettura

La sua architettura segue il pattern Model-View-Controller (MVC), sebbene nella documentazione ufficiale venga spesso fatto riferimento al pattern Model-View-Template (MVT) per sottolineare la differenza nell'implementazione del layer di presentazione.

Nel dettaglio possiamo vedere come Django interpreti e implementi il tradizionale pattern architetturale MVC (Model-View-Controller) in un modo leggermente diverso, che la sua documentazione chiama Model-View-Template (MVT). Per comprendere meglio la differenza, analizziamo entrambi i pattern:

Model-View-Controller (MVC)

Il pattern MVC è un design pattern comune nell'architettura del software utilizzato per separare un'applicazione in tre componenti interconnesse:

- Model: rappresenta i dati e le regole di business. È responsabile dell'accesso ai dati, solitamente interagendo con un database.

- View: si occupa della presentazione e visualizzazione dei dati. In una applicazione web, corrisponde spesso alla parte front-end e alla presentazione della UI.[11]

- Controller: agisce come un intermediario tra Model e View. Gestisce le richieste dell'utente, elabora i dati attraverso il Model e restituisce la risposta appropriata utilizzando la View.

Model-View-Template (MVT)

Mentre Django segue l'approccio generale MVC, lo implementa con alcune differenze chiave, pertanto ha coniato il termine Model-View-Template (MVT):

- Model: analogamente al MVC, rappresenta la struttura dei dati e le regole di business. Django fornisce un ORM[12] integrato per facilitare le interazioni con il database.

- View: in Django, la View gestisce altre funzionalità oltre che la mera presentazione dei dati. Si occupa di gestire la logica di controllo, ovvero quale informazione mostrare e come rispondere alle azioni dell'utente. Questa è una deviazione significativa dal MVC tradizionale, dove queste responsabilità sono delegate al Controller

- Template: questo componente si occupa esclusivamente della presentazione dei dati. In pratica, è il layer di presentazione in Django, dove i dati vengono iniettati e visualizzati all'utente finale. A differenza della "View" nel MVC tradizionale che può gestire logica di presentazione, il "Template" in Django è strettamente orientato alla definizione di come i dati vengono presentati.

Risulta evidente che mentre entrambi i pattern mirano a separare le preoccupazioni in un'applicazione web, Django differisce nella sua implementazione specifica, mettendo una maggiore enfasi sulla separazione tra la logica di controllo e la presentazione dei dati, risultando in quello che chiama il pattern Model-View-Template.

1.8.3 Punti di forza

- ORM (Object-Relational Mapping): permette agli sviluppatori di interagire con il database utilizzando il linguaggio Python, senza la necessità di scrivere query SQL direttamente, permette anche di associare le tabelle del database ad un oggetto dove risulta possibile manipolarlo

[11] Durante l'esperienza di tirocinio di 150 ore, non è stato possibile andare a personalizzare questa sezione per mancanza di tempo, si è preferito lavorare sull'efficacia vera e propria dell'applicazione, avendo riscontrato alcune problematiche nel far interagire tutte queste tecnologie tra di loro simultaneamente

[12] spiegato nel paragrafo 1.8.3

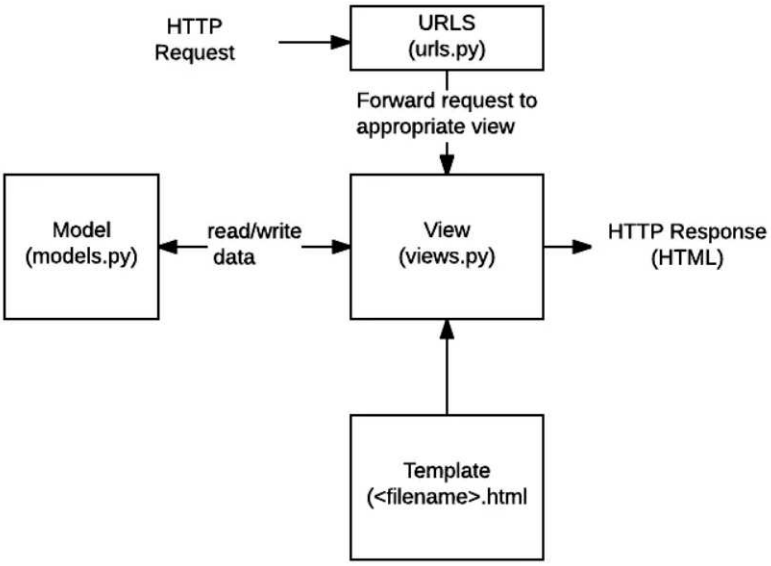

Figura 1.9: Diagramma di funzionamento

- Sistema di autenticazione: django viene fornito con un sistema di autenticazione integrato che gestisce la creazione di account utente, l'accesso, la disconnessione e altre funzionalità correlate.
- Sistema di template: offre un linguaggio di template potente e flessibile per definire la struttura e il design delle pagine web.
- Amministrazione: una delle caratteristiche più apprezzate di Django è l'interfaccia di amministrazione autogenerata. Questa fornisce un pannello di controllo per gestire il contenuto e gli utenti del sito.
- Sicurezza: django pone una grande enfasi sulla sicurezza, fornendo protezione contro molti dei vettori di attacco comuni come Cross-site Scripting (XSS) e l'Injection SQL

1.9 Conclusioni

Algorand si presenta come una delle soluzioni blockchain più avanzate e promettenti attualmente disponibili. La sua combinazione di rapidità, sicurezza e decentralizzazione lo rende particolarmente adatto a una vasta gamma di applicazioni, dalle semplici transazioni finanziarie alle più complesse soluzioni aziendali. L'adozione crescente e la comunità in espansione sono testimonianza del suo potenziale di trasformare l'ecosistema delle tecnologie blockchain. Inoltre il capitolo si dimostra essenziale per capire tutto il processo di sviluppo dell'app, sono state analizzate le tecnologie principali che poi verranno utilizzate, per far si che tutta l'applicazione lavori coordinata.

CAPITOLO 2

Ambito Applicativo

Il progetto è realizzato (P.O.C) su incarico di IntegrityKEY s.r.l., una società che si posiziona all'incrocio tra il settore agroalimentare e la supply chain nella zona dell'Italia centrale, con un focus particolare sulla regione delle Marche, La mission dell'azienda è di offrire servizi alle piccole e medie imprese (PMI) che aspirano a lanciare prodotti innovativi sul mercato, non solo italiano, ma anche a livello europeo ed internazionale. Di seguito, delineeremo il dominio operativo del progetto, analizzando ciascuna macro area coinvolta, ponendo il focus sulla tracciabilità trasparente dei prodotti alimentari.

2.1 Committente Del Progetto

Il progetto viene realizzato per conto di IntegrityKEY che pur essendo una start-up di recente formazione, nata nell'inizio del 2022, questa giovane realtà ha già ottenuto un riconoscimento significativo nel suo campo. Definita come una start-up innovativa e spin-off accademico, si è distinta per i numerosi riconoscimenti ricevuti, tra cui il premio ecapital 2020, la startcup Marche, e le finali del Premio Nazionale Innovazione (PNI) nel 2021 e 2022.

Nel 2022, IntegrityKEY ha lanciato la sua prima piattaforma, una web app progettata per offrire soluzioni diverse e di grande valore ai suoi clienti. Questa applicazione consente alle aziende di evidenziare e gestire efficacemente la tracciabilità dei loro prodotti, sia a livello interno sia nei confronti dei consumatori esterni.

Questa tesi riflette la parte di ricerca e innovazione per la tracciabilità dei dati su IntegrityKEY.

2.2 Supply Chain

La supply chain, o catena di approvvigionamento, abbraccia ogni aspetto e fase della produzione, dal concepimento del prodotto alla sua distribuzione e vendita all'utente finale. In questa complessa rete, le aziende devono essere capaci di accedere rapidamente e in modo efficiente alle risorse necessarie, per soddisfare le mutevoli esigenze del mercato.

Fondamentale distinguerla dalla logistica, dato che i due termini, pur strettamente correlati, vengono spesso impropriamente utilizzati come sinonimi. La logistica è, infatti, una componente specifica della supply chain: essa si occupa dell'organizzazione, gestione e trasporto di prodotti e materie prime lungo la catena di produzione. Mentre la logistica si focalizza sul trasferimento fisico e sulla distribuzione dei beni, la supply chain, in senso lato, comprende tutti i processi, risorse e attori coinvolti nella filiera produttiva, dalla produzione al punto vendita.

2.3 Tracciabilità

Per tracciabilità si intende la capacità di seguire un prodotto attraverso tutti i passaggi della sua filiera produttiva, dalla sua origine al consumatore finale. Ad esempio, nella produzione del succo di mela, il processo di tracciabilità parte dal campo in cui il l'albero viene coltivato, passando per tutte le fasi di raccolta e lavorazione , fino a giungere sugli scaffali dei negozi. Il termine inglese "tracking" si riferisce a questo concetto. Tuttavia, in italiano, è essenziale distinguerla dal concetto di rintracciabilità che verrà esaminato più avanti nel capitolo 2.4 .

La tracciabilità può essere classificata in due principali sotto categorie: tracciabilità interna ed esterna.

- Tracciabilità interna: capacità di un'azienda di monitorare l'origine delle sue materie prime, i processi di trasformazione e la distribuzione dei prodotti finiti. Questo tipo di tracciabilità è gestita singolarmente da ogni azienda, indipendentemente dalle azioni di altre entità nella supply chain.

- Tracciabilità esterna: coinvolge l'intera filiera, richiede la collaborazione di tutte le aziende coinvolte. In settori come l'agroalimentare, la raccolta e l'integrazione di dati da diverse aziende è una sfida notevole.

È importante sottolineare che la tracciabilità non si limita semplicemente alla raccolta di dati, ma fungono da ruolo fondamentale per quelle che sono le aziende responsabili di ogni singolo lotto di prodotto.

Dal punto di vista normativo, tracciabilità ed etichettatura sono strettamente interconnesse. Le leggi vigenti impongono l'inserimento di specifiche informazioni sulla tracciabilità direttamente sull'etichetta del prodotto, aumentando la trasparenza per il consumatore e garantendo l'accesso a informazioni essenziali. Questa pratica assicura non solo che le aziende siano in grado di identificare e affrontare prontamente eventuali rischi per la salute, ma anche di assumersi la piena responsabilità in caso di problemi. In ultima analisi, la tracciabilità manda un duplice messaggio: il processo di produzione è trasparente e le aziende si assumono una responsabilità chiara e definita per i loro prodotti.

2.4 Rintracciabilità

Per rintracciabilità si intende la capacità di ricostruire a ritroso l'intero percorso di un prodotto, risalendo fino alle sue materie prime originarie. Mentre la tracciabilità segue un prodotto dal suo inizio, cioè dalla produzione delle materie prime, fino al suo arrivo sul mercato, la rintracciabilità opera in direzione opposta: inizia dal prodotto finito e risale fino alla sua origine.

Il processo ha diverse finalità fondamentali:

- Ricostruzione dettagliata del prodotto: accedere a informazioni chiave sul prodotto, quali componenti, lotti di provenienza e metodologie di produzione.

- Creazione di una cronologia produttiva: identificare le fasi di trasferimento di proprietà e il percorso che il prodotto ha attraversato.

- Gestione delle emergenze: qualora emergessero rischi legati al prodotto, sia per l'essere umano che per l'ambiente, la rintracciabilità permette di effettuare richiami mirati.

- Monitoraggio a lungo termine: facilitare la valutazione degli impatti di certi prodotti sull'ambiente e sulla salute nel lungo periodo.

- Convalida delle etichette: assicurare che le informazioni fornite al consumatore attraverso l'etichettatura siano veritiere.

La rintracciabilità rappresenta uno strumento marketing di inestimabile valore. Essa infatti permette alle aziende di valorizzare la trasparenza del proprio processo produttivo, mettendo in evidenza le caratteristiche uniche del prodotto, le specificità legate all'origine geografica o alla metodologia di lavorazione, e promuovendo i valori distintivi del brand.

Dal punto di vista pratico, se ogni entità all'interno della supply chain ha attuato con successo procedure di tracciabilità interna, si avranno a disposizione tutti gli elementi necessari per ricostruire la storia completa del prodotto. La rintracciabilità, in questo contesto, diventa un esercizio di raccolta e interpretazione delle "impronte digitali" che ogni fase di lavorazione ha impresso sul prodotto. Per tutta questa serie di motivi risulta di fondamentale importanza essere in grado di garantire strumenti per avere una tracciabilità e rintracciabilità efficienti e sicuri.

2.5 Conclusioni

Questo capitolo serve per comprendere l'ambiente di lavoro e dove il progetto in esame è stato inserito, andando cosi a capire le scelte fatte successivamente in merito allo smart contract realizzato al fine di garantire la tracciabilità e la rintracciabilità per i motivi spiegati in questa sezione.

CAPITOLO 3

Obiettivi Del Progetto

Nel presente capitolo, verranno presi in disamina gli obiettivi fondamentali del progetto che viene presentato. L'obiettivo primario è la progettazione e l'implementazione di un algoritmo, il cui compito sarà quello di generare uno smart contract partendo da specifici dati forniti dall'utente attraverso una dedicata interfaccia di un'applicazione[1]*. Questa P.O.C. ha delle ripercussioni di ampia portata: mira a offrire un sistema robusto e intuitivo che permette a un utente, previamente registrato sulla piattaforma IntegrityKEY, di accedere in modo sicuro e veloce, di registrare il proprio prodotto e, ancor più rilevante, di annotare meticolosamente ogni singolo passaggio che il prodotto ha attraversato. Questo processo non solo amplifica la capacità di tracciare e rintracciare l'origine e la storia di un prodotto, come ampiamente discusso nel capitolo precedente, ma aggiunge anche un significativo valore di trasparenza e fiducia per tutti gli attori coinvolti nella catena di approvvigionamento. Con la crescente richiesta di trasparenza e verificabilità nel mondo moderno, la realizzazione di tale algoritmo e sistema diventa una risorsa inestimabile.*

[1] l'interfaccia rimane quella di base di Django avendo scelto di lavorare maggiormente sul funzionamento che sulla presentazione per il poco tempo a disposizione

3.1 Soluzione Tecnologica

La soluzioni tecnologiche adottate sono state molte per questo P.O.C, in particolare però la sperimentazione sulla fattibilità è basata su Algorand e sul suo possibile utilizzo. Come spiegato precedentemente la Blockchain è una catena di blocchi concatenati in cui vi è possibile leggere all'interno, questa caratteristica la rende molto efficiente per sviluppare app sulla tracciabilità e rintracciabilità. Data la natura dello smart contract spiegata in 1.2, una volta deployato nella main-net non è più modificabile, rendendo questa soluzione tecnologica ottima per garantire la verità in ambito di tracciabilità o rintracciabilità, dove non serve appoggiarsi ad un ente certificatore esterno, perchè la blockchain e lo Smart contract compiono questo lavoro data la loro natura intrinseca.

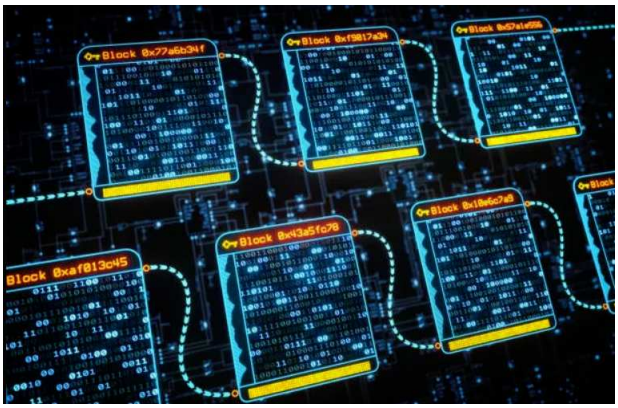

Figura 3.1: Catena di blocchi

3.2 Glossario del progetto

Per garantire la massima chiarezza e trasparenza nel corso di questa trattazione, è di cruciale importanza definire e chiarire alcuni termini chiave che saranno frequentemente utilizzati. Questi termini, formano un quadro che ci accompagnerà passo dopo passo nella presentazione del lavoro svolto. Con questo glossario e le fondamenta teoriche precedentemente stabilite, saremo in grado di navigare l'intero percorso del progetto con una chiara visione. Questo garantirà che ogni fase, decisione e implementazione siano comprese nel loro contesto appropriato, eliminando qualsiasi possibile ambiguità o fraintendimento. Ecco quindi un elenco dei termini salienti, il cui intento non è solo informativo, ma anche formativo, contribuendo così a fornire una solida base di partenza per le sezioni successive.

Termine	Definizione	Sinonimi
Blockchain	struttura dati che consiste in una sequenza di record chiamati blocchi	catena di blocchi
Smart Contract	contratto auto eseguito non modificabile	
Utenti	persone che interagiscono con l'app	utilizzatori
Prodotto	Elemento che viene coltivato o trasportato da un utente	
Deploy	Publicazione di uno Smart Contract su una rete blockchain	pubblicazione
Mainnet	Rete operativa principale, dove avvengono le transazioni, le transazioni sono a pagamento	Rete operativa
Testnet	versione parallela della Blockchain principale utilizzata esclusivamente per testare, le transazioni sono gratuite	Rete di test

3.3 Analisi dei requisiti

L'azienda ha delineato con precisione i requisiti essenziali per la realizzazione dell'applicazione per le funzionalità dell'app, cosa fondamentale per qualsiasi progetto, cosi da avere ben chiaro il lavoro da svolgere. I requisiti possono essere suddivisi in due categorie principali: funzionali e non funzionali.

I requisiti funzionali descrivono le funzioni specifiche che il sistema o l'applicazione dovrebbe essere in grado di eseguire. Essi rappresentano le funzionalità principali, cioè "cosa" il sistema deve fare.

I requisiti non funzionali, invece, si riferiscono alle caratteristiche qualitative del sistema, come le prestazioni, la sicurezza, l'usabilità, ecc. Questi requisiti stabiliscono "come" il sistema deve eseguire le sue funzioni.

3.3.1 Requisiti Funzionali

I requisiti funzionali descrivono le funzioni o le caratteristiche specifiche che il sistema deve implementare. Essi delineano le azioni che il sistema deve eseguire per soddisfare gli obiettivi previsti.

RF1 L'applicazione deve permettere all'utente di essere in grado di creare lo smart contract.

RF2 L'applicazione deve permettere all'utente di inserire un prodotto.

RF3 L'applicazione deve permettere la registrazione di altri utenti.

RF4 L'applicazione deve permettere la visualizzazione del link dello smart contract creato.

RF5 L'applicazione deve suportare da un database

3.3.2 Requisiti non Funzionali

I requisiti non funzionali, al contrario, si focalizzano sulle qualità o gli attributi del sistema piuttosto che sulle specifiche funzionalità. Essi dettano come il sistema deve comportarsi e stabiliscono criteri di performance, sicurezza, usabilità ed altri aspetti qualitativi.

RF1 L'applicazione deve essere sviluppata su Django.

RF2 L'applicazione deve essere scritta in Pyhton.

RF3 L'applicazione deve permettere la creazione di uno smart contract su Algorand.

RF4 L'applicazione deve dialogare tramite docker ad una testnet locale.

3.4 Casi d'uso

I casi d'uso permettono di descrivere le interazioni tra uno o più attori e il sistema in esame, rappresentando le funzioni che il sistema dovrà eseguire in risposta a una particolare richiesta da parte di un attore

3.4.1 Identificazione degli attori

Gli attori rappresentano gli utilizzatori della piattaforma, che dovrebbero svolgere diverse mansioni e all'interno dell'applicazione. In questo caso non è stato definito un diverso accesso per gradi di utente, di conseguenza possiamo identificare un unico utente, che ha a disposizione tutte le funzionalità comprese quelle dell'amministratore.

- Utente: rappresenta chi utilizza l'applicazione, l'attore in questione è l'amministratore che può andare a interagire con il sistema con i casi d'uso.[2]

3.4.2 Identificazione dei casi d'uso

l'identificazione dei casi d'uso fornisce una rappresentazione ad alto livello delle funzionalità offerte dal sistema in termini di sequenze di azioni, mettendo in evidenza le relazioni tra gli utenti e le diverse funzioni del sistema.

RF1 Login: il caso d'uso viene autogestito da Django nel momento che si va a creare un superUser, il login è possibile solo per l'amministratore.

RF2 Registrazione prodotto: il caso d'uso permette la registrazione di un nuovo prodotto.

[2]non è stato definita una registrazione ne un diverso accesso per altri utenti, a causa del limitato tempo a disposizione

RF3 Registrazione utente: il caso d'uso deve permette la registrazione di un nuovo utente e inserirlo nel database.

RF4 Creazione smart contract: il caso d'uso deve permettere la creazione di uno smart contract su rete Algorand, andando a inserire parametri personalizzabili come utente e prodotto.

RF5 Permettere la visualizzazione: il caso d'uso deve permettere la visualizzazione dello smart contract appena creato

3.4.3 Diagrammi dei casi d'uso

I diagrammi dei casi d'uso sono una componente fondamentale della modellazione UML (Unified Modeling Language), utilizzati per rappresentare graficamente le interazioni tra gli attori esterni e un sistema. Essi forniscono una visione ad alto livello delle funzionalità del sistema, evidenziando le relazioni e le interazioni tra le varie componenti.

In un diagramma dei casi d'uso, le funzionalità del sistema sono rappresentate da ellissi, mentre gli attori, che possono essere sia utenti reali che altri sistemi, sono raffigurati come figure stilizzate. Le relazioni tra attori e casi d'uso sono rappresentate mediante linee, che indicano le interazioni e le azioni che un attore può compiere sul sistema.

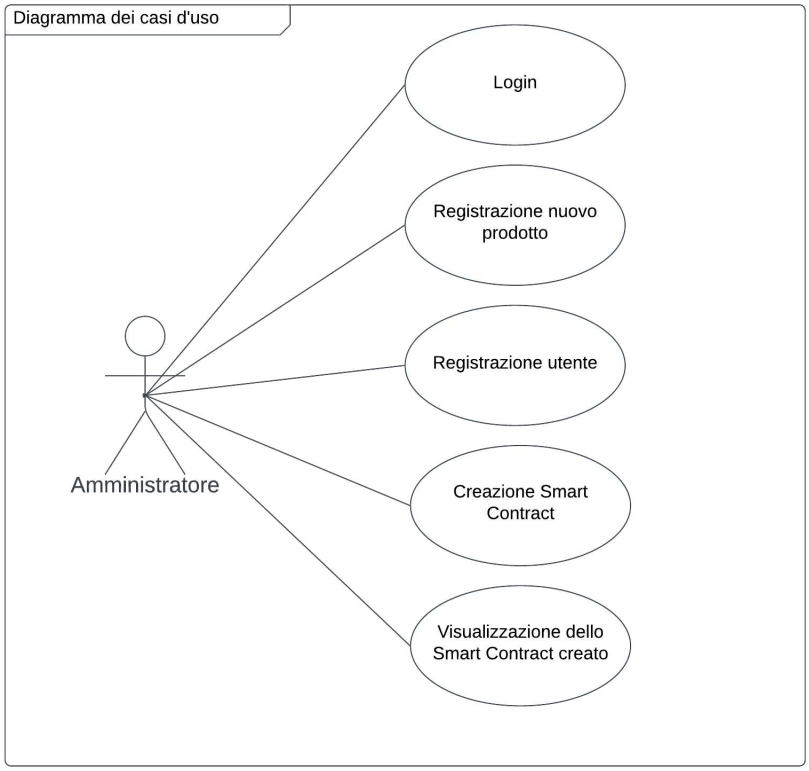

Figura 3.2: Diagramma dei casi d'uso

3.5 Conclusioni

Nel corso di questo capitolo, è stata effettuata un'analisi approfondita della soluzione tecnologica che sarà adottata per il progetto in questione. Questa fase è stata cruciale per la comprensione e la definizione dei vari aspetti che influenzeranno lo sviluppo e l'implementazione del sistema. In particolare, sono stati delineati i requisiti funzionali, che costituiscono le funzionalità e le caratteristiche chiave che il sistema dovrà avere per soddisfare gli obiettivi del progetto. Analogamente, sono stati definiti i requisiti non funzionali, benché meno visibili all'utente finale, risultano essere importanti per garantire il corretto funzionamento, e scalabilità nel sistema dell'azienda. Un glossario è stato anche preparato per fornire definizioni chiare e non ambigue dei termini tecnici e dei concetti utilizzati.

CAPITOLO 4

Applicazione Sviluppata

Nel seguente capitolo verrà presa in disamina l'applicazione sviluppata, verranno quindi portati esempi di funzionamento e verranno spiegate anche alcune parti salienti del codice. All'inizio del capitolo troveremo la spiegazione della base di dati progettata e sviluppata al fine di un corretto funzionamento dell'applicazione, poi passeremo ad una visione della parte view (admin) e per finire un esempio di SmartContract realizzato.

4.1 Base di dati

Per permettere la realizzazione della suddetta applicazione risulta necessario salvare diverse informazioni su un database, per cui è stata necessaria la progettazione e realizzazione di un database capace di salvare i dati necessari. Django risulta essere molto funzionale, aiutando lo sviluppatore tramite le migrazioni che permettono di evitare di scrivere i comandi SQL direttamente.

4.1.1 le migrazioni

La migrazione in Django è una potente caratteristica che permette di gestire le modifiche al modello dei dati (cioè la struttura del database) senza la necessità di intervenire direttamente sul database stesso. Le migrazioni sono come un sistema di versionamento per il database. Esse registrano le modifiche apportate ai modelli di Django e le convertono in comandi SQL, che vengono poi applicati al database.

Andando nel dettaglio possiamo vedere dei comandi per la gestione delle migrazioni in Django:

- Modifiche al Modello: dopo aver creato il tuo modello iniziale[1] (che rappresenta una tabella nel database) in un'app Django, potresti avere la necessità di modificare questo modello in futuro, ad per aggiungere/eliminare/ modificare un campo.

- Creazione della Migrazione: quando modifichi il modello, devi informare Django che desideri creare una nuova migrazione per tener traccia di queste modifiche. Ciò può essere fatto con il comando: `python manage.py makemigrations nome_app` questo comando dirà a Django di generare dei nuovi file di migrazione basati sulle modifiche rilevate nei tuoi modelli. Questi file vengono creati nella directory migrations all'interno della tua app, verranno anche nominati in maniera incrementale.

- Verifica della Migrazione: prima di applicare effettivamente la migrazione, potresti voler vedere gli SQL che Django intende eseguire. Questo può essere fatto con: `python manage.py sqlmigrate nome_app nome_migrazione` dove nome_migrazione è il nome del file di migrazione (senza l'estensione .py).

- Applicazione della Migrazione: una volta che sei soddisfatto delle modifiche e vuoi applicare la migrazione al database, esegui: `python manage.py migrate` questo comando applica tutte le migrazioni pendenti al database. È equivalente all'esecuzione di tutti gli SQL associati alle migrazioni.

[1] poi verranno spiegati i modelli

- Annullare una Migrazione: se hai bisogno di annullare una migrazione (cioè tornare indietro a una versione precedente del database), puoi farlo con: `python manage.py migrate nome_app nome_migrazione_precedente`

La potenza del sistema di migrazione di Django risiede nella sua capacità di creare, applicare e gestire modifiche al database in modo controllato e sistematico. Ciò consente agli sviluppatori di modificare la struttura del database senza dover scrivere manualmente complicate query SQL e, cosa più importante, senza temere di perdere dati o di interrompere le funzionalità esistenti.

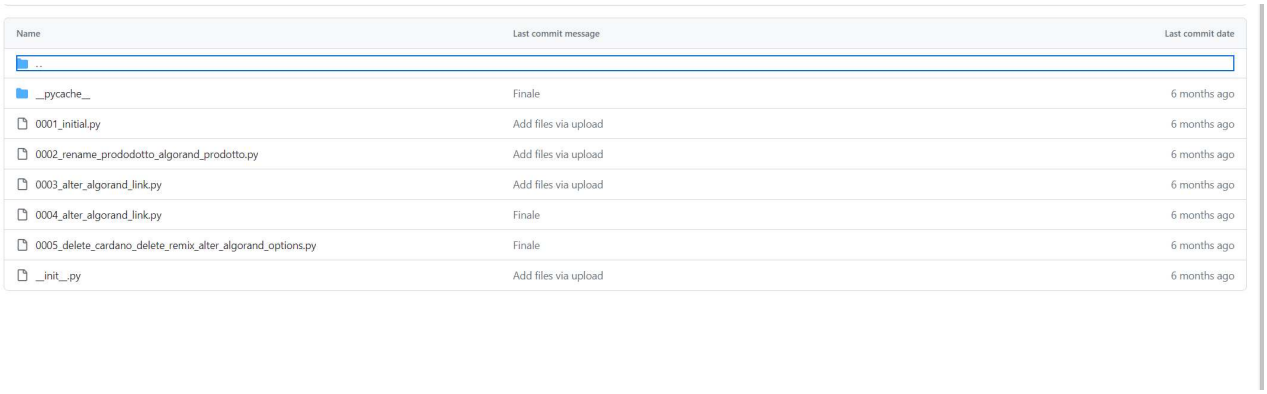

Figura 4.1: Migrazioni database

Esempio di migrazione

Di seguito viene riportato un esempio di migrazione, in questo caso la numero **0003**, di seguito al codice verrà fornita la spiegazione dettagliata del codice autogenerato

```
1     # Generated by Django 4.1.6 on 2023-03-01 15:03
2
3 from django.db import migrations, models
4
5
6 class Migration(migrations.Migration):
7
8     dependencies = [
9         ('Smart_Contract', '0003_alter_algorand_link'),
10    ]
11
12    operations = [
13        migrations.AlterField(
14            model_name='algorand',
15            name='link',
16            field=models.CharField(editable=False, max_length=500),
```

```
17          ),
18      ]
```

- `from django.db import migrations, models`: importazione dei moduli necessari per definire la migrazione.
- `class Migration(migrations.Migration)`class Migration(migrations.Migration): definizione della classe di migrazione, che eredita dalla classe migrations.Migration di Django.
- `dependencies = [...]`: specifica le dipendenze della migrazione, cioè altre migrazioni che devono essere applicate prima di questa. In questo caso, dipende da una migrazione chiamata 0003_alter_algorand_link nel modulo Smart_Contract.
- `operations = [...]`: questa è la parte chiave. Elenco delle operazioni da eseguire per applicare questa migrazione. In questo caso, c'è una singola operazione: migrations.AlterField.
- `model_name='algorand'`: la modifica avverrà sulla tabella del modello Algorand.
- `name='link'`: il campo da modificare è link.
- `field=models.CharField(editable=False, max_length=500)`: i nuovi attributi del campo. In questo caso, il campo link rimarrà un campo CharField con una lunghezza massima di 500 caratteri, e non sarà modificabile (editable=False).

In altre parole, questa migrazione va a cambiare il campo link del modello Algorand per farlo diventare un campo di testo (CharField) con una lunghezza massima di 500 caratteri, che non è modificabile.

4.1.2 Database

In questa sezione andremo a vedere come è stato progettato il Database, passando per la progettazione dello schema E-R, all'implementazione tramite i modelli in Django, vedremo anche le parti del codice che evitano di scrivere direttamente i comandi SQL.

4.1.3 Schema E-R

Uno schema Entità-Relazione (E-R) è un tipo di diagramma che rappresenta la struttura logica di un database. Questo diagramma visualizza entità (in questo caso, gli utenti, i prodotti, e lo smart contract), gli attributi di queste entità (in questo caso ad esempio gli attributi di prodotto sono tipo, quantità e unita di misura) e le relazioni tra le entità.

Lo schema E-R viene rappresentato con le seguenti caratteristiche:

- Entità: sono oggetti o concetti che esistono indipendentemente dalle altre e possono essere identificati univocamente.Le entità vengono rappresentate da rettangoli.
- Attributi: sono le proprietà o le caratteristiche delle entità. Gli attributi sono spesso rappresentati da piccoli cerchi collegati alle loro entità corrispondenti.
- Relazioni: indicano come due entità sono collegate tra loro nel database. Ad esempio, Le relazioni sono spesso rappresentate da rombi che collegano le entità correlate.
- Cardinalità: specifica quante istanze di un'entità sono collegate a un'istanza di un'altra entità.

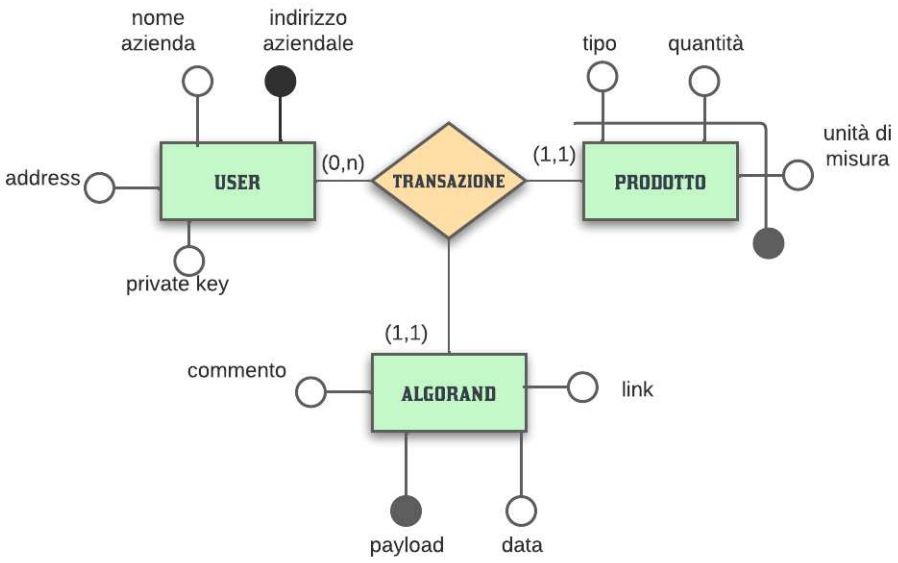

Figura 4.2: Schema E-R

4.1.4 I modelli

Ora è possibile vedere nel dettaglio i modelli del database realizzati, tramite apposite strutture di Django.[2]

Modello del prodotto

Entreremo nel dettaglio di come è stato implementato il database andando a vedere come sono state realizzate le entità, che in Django corrispondono ai modelli

```
1    class Prodotto(models.Model):
2    '''  creazione di prodotto '''
3    Tipo = models.CharField(max_length=500)
4    Quantita=models.CharField(max_length=500)
5    UnitaMisura=models.CharField(max_length=500)
```

Il codice scritto sopra crea una rappresentazione del database per un "Prodotto". Verrà analizzato nel dettaglio:

- `class Prodotto(models.Model)`: questa riga definisce una nuova classe chiamata 'Prodotto', che eredita da 'models.Model'. Indicando che "Prodotto" è un modello Django e rappresenterà una tabella nel database.
- `Tipo = models.CharField(max_length=500)`: questa riga crea un campo caratteri chiamato 'Tipo' per il modello "Prodotto". Questo campo può contenere stringhe fino a 500 caratteri di lunghezza.
- `Quantita=models.CharField(max_length=500)`: analogamente, crea un campo caratteri chiamato 'Quantita' che può contenere stringhe fino a 500 caratteri.
- `UnitaMisura=models.CharField(max_length=500)`: crea un campo caratteri chiamato 'UnitaMisura' che può contenere stringhe fino a 500 caratteri.

Modello dell'utente

```
1    class CustomUser(AbstractUser):
2     """ contiene la creazione degli user"""
3     address = models.CharField(max_length=255, editable=False,unique=True)
4     privatekey = models.CharField(max_length=255, editable=False)
5     IndirizzoAziendale = models.CharField(max_length=255, unique=True)
6     objects = CustomUserManager()
```

[2]Bene ricordare che nel database troveremo attributi che poi non verranno usati in questa rappresentazione, ma che si potrebbero rivelare fondamentali in futuro.

- `class CustomUser(AbstractUser)`: si tratta della definizione di una classe CustomUser che eredita da AbstractUser, una classe base fornita da Django per la gestione degli utenti. Questo ti permette di estendere le funzionalità di base degli utenti senza dover riscrivere da zero tutto il codice.

- `address = models.CharField(max_length=255, editable=False, unique=True)`: questo è un campo di testo con una lunghezza massima di 255 caratteri che contiene l'indirizzo dell'utente. Il campo è impostato come non modificabile (editable=False) e univoco (unique=True), il che significa che ogni utente deve avere un indirizzo unico.

- `privatekey = models.CharField(max_length=255, editable=False)`: questo campo è utilizzato per memorizzare una chiave privata per ogni utente. Anche questo campo è impostato come non modificabile.

- `IndirizzoAziendale = models.CharField(max_length=255, unique=True)`: questo campo contiene l'indirizzo aziendale associato all'utente e deve essere unico.

- `objects = CustomUserManager()`: questa è una istanza del manager del modello CustomUser. I manager sono delle interfacce che Django utilizza per accedere al database.

Modello del salvataggio dati SmartContract

```
1  class Algorand(models.Model):
2   ''' contiene generazione di smart contract su Algorand '''
3   payload = models.CharField(max_length=500, editable=False)
4   data = models.DateTimeField(auto_now_add=True, editable=False)
5   link = models.CharField(max_length=500, editable=False)
6   commento= models.CharField(max_length=500, blank=True)
7   user = models.ForeignKey(CustomUser, on_delete=models.CASCADE,
    ↪ related_name='Algorand')
8   prodotto = models.ForeignKey(Prodotto, on_delete=models.CASCADE,
    ↪ related_name='Algorand')
```

- `class Algorand(models.Model)`: questa linea definisce una nuova classe di modello Django chiamata 'Algorand', che eredita dalla classe base `models.Model`.

- `payload = models.CharField(max_length=500, editable=False)`: definisce un campo di testo chiamato 'payload' con una lunghezza massima di 500 caratteri. Questo campo è impostato come non modificabile ('editable=False')[3].

[3]Questo attributo è servito per fare delle prove, nella creazione di Smart Contract diversi, è rimasto nel caso servisse in seguito

- `data = models.DateTimeField(auto_now_add=True, editable=False)`: questo campo memorizza la data e l'ora di creazione del record. L'opzione `auto_now_add=True` significa che la data e l'ora saranno impostate automaticamente quando un nuovo record viene creato. Anche questo campo è impostato come non modificabile.

- `link = models.CharField(max_length=500, editable=False)`: un altro campo di testo per memorizzare un link, anch'esso con una lunghezza massima di 500 caratteri e impostato come non modificabile.

- `commento= models.CharField(max_length=500, blank=True)`: Questo campo è utilizzato per memorizzare un commento, e l'opzione 'blank=True' indica che è un campo opzionale.

- `user = models.ForeignKey(CustomUser, on_delete=models.CASCADE, related_name='Algorand')`:
  questo è un campo di chiave esterna che collega ogni record della classe 'Algorand' a un record della classe 'CustomUser'.
  L'opzione `on_delete=models.CASCADE` significa che se un utente viene eliminato, tutti i suoi record correlati in 'Algorand' verranno anch'essi eliminati. 'related_name' è usato per definire il nome da utilizzare per la relazione inversa da 'CustomUser' a 'Algorand'.

- `prodotto = models.ForeignKey(Prodotto, on_delete=models.CASCADE, related_name='Algorand')`: simile al campo 'user', questo campo di chiave esterna collega ogni record della classe 'Algorand' a un record della classe 'Prodotto'. Anche qui, l'eliminazione di un prodotto comporterà l'eliminazione di tutti i record correlati in 'Algorand'.

Questi sono i modelli creati per la realizzazione di questo progetto, in ogni migrazione risulta possibile vedere i cambiamenti e gli aggiustamenti svolti durante tutta la fase di sviluppo.

4.2 Registrazione Utente

4.2.1 Visualizzazione utenti

La schermata seguente, ci mostra la visualizzazione di tutti gli utenti inseriti nel database, risulta possibile personalizzare la vista, con del codice che verrà analizzato successivamente

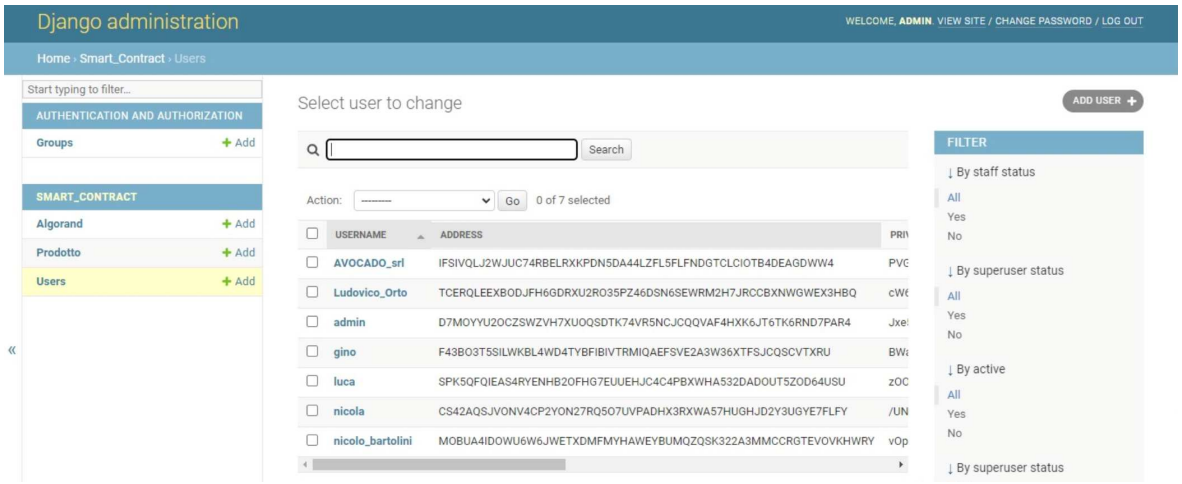

Figura 4.3: Schermata per vedere tutti gli User

La schermata è stata generata dal codice che verrà analizzato nel paragrafo seguente.

```
    class CustomUserAdmin(UserAdmin):
    model = CustomUser
    list_display = ['username', "address", "privatekey", "email", "
    ↪ IndirizzoAziendale"]
    readonly_fields= ['address', 'privatekey']
    search_fields = ["address", "username"]
    list_filter = UserAdmin.list_filter + ('is_superuser', 'is_staff')
    add_fieldsets = (
        (None, {
            'classes': ('wide',),
            'fields': ('username', 'email', 'password1', 'password2', '
    ↪ IndirizzoAziendale')}
        ),
    )


admin.site.register(CustomUser, CustomUserAdmin)
```

Il codice definisce una classe CustomUserAdmin che eredita da UserAdmin, fornita da Django. Questa classe viene utilizzata per personalizzare l'interfaccia di amministrazione al fine di gestire oggetti CustomUser, che è un modello utente personalizzato definito in precedenza nel codice. Vediamo in dettaglio cosa fanno le diverse parti:

- `model = CustomUser`: specifichiamo che il modello da gestire in questa interfaccia di amministrazione è CustomUser.
- `list_display = ['username', "address", "privatekey", "email", "IndirizzoAziendale"]`: definisce quali campi del modello CustomUser devono essere

mostrati nella lista di oggetti utente nell'interfaccia di amministrazione. In questo caso, i campi sono 'username', 'address', 'privatekey', 'email', e 'IndirizzoAziendale'.

- `readonly_fields= ['address', 'privatekey']`: indica quali campi sono in sola lettura nell'interfaccia di amministrazione. In questo caso, gli indirizzi 'address' e 'privatekey' non possono essere modificati direttamente dall'interfaccia di amministrazione.
- `search_fields = ["address", "username"]`: abilita una casella di ricerca nell'interfaccia di amministrazione che permetterà agli utenti di cercare gli oggetti CustomUser in base ai campi 'address' e 'username'.
- `list_filter = UserAdmin.list_filter + ('is_superuser', 'is_staff')`: Eredita tutti i filtri di default del UserAdmin standard e aggiunge la possibilità di filtrare gli utenti anche in base ai campi `'is_superuser'` e 'is_staff'.
- `add_fieldsets = (...)`: definisce i campi che saranno mostrati quando si crea un nuovo oggetto CustomUser attraverso l'interfaccia di amministrazione. In questo caso, i campi sono 'username', 'email', 'password1', 'password2', e 'IndirizzoAziendale'.
- `admin.site.register(CustomUser, CustomUserAdmin)`: registra il modello CustomUser e la classe CustomUserAdmin personalizzata nell'interfaccia di amministrazione di Django. Questo fa sì che CustomUser venga gestito usando la configurazione definita in CustomUserAdmin.

Questa è la schermata per il salvataggio di nuovi utenti cosi da poterli inserire nel database.

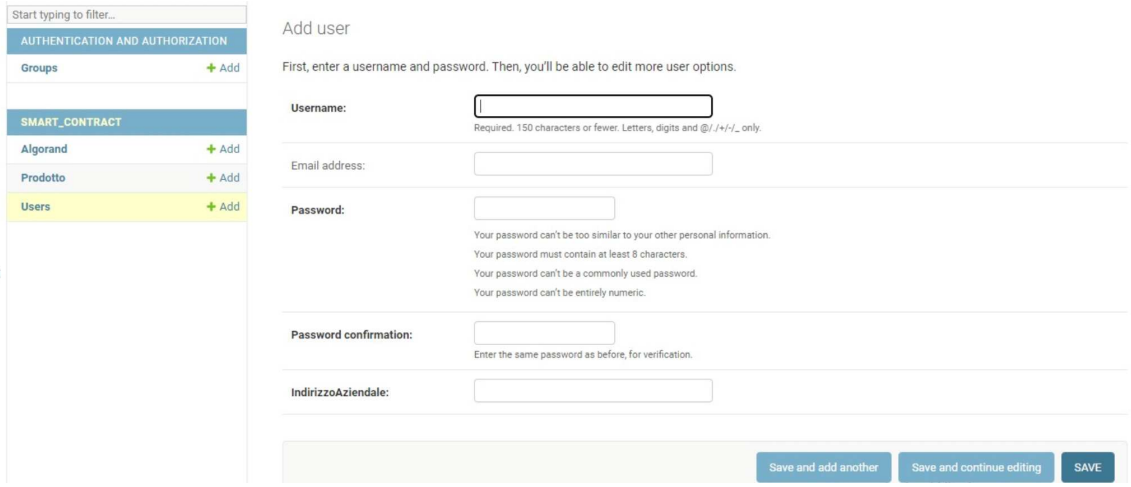

Figura 4.4: Schermata di Registrazione utente

```
def save(self,*args, **kwargs):
    private_key, address = account.generate_account()
```

```
3       self.privatekey= private_key
4       self.address= address
5
6        super().save(*args, **kwargs)
7
8       def __str__(self):
9           return f"{self.username} {self.IndirizzoAziendale}"
```

Di seguito andremo nel dettaglio del codice, analizzando gli aspetti principali:

- `def save(self, *args, **kwargs)`: questo è un metodo sovrascritto del metodo save ereditato dalla classe Model di Django. Verrà chiamato ogni volta che chiamerai save() su un'istanza di questo modello.

- `private_key, address = account.generate_account()`: vengono generati automaticamente questi campi.[4] `self.privatekey = private_key`: La chiave privata generata viene assegnata al campo privatekey dell'oggetto modello che sta per essere salvato. `self.address = address`: allo stesso modo, l'indirizzo generato viene assegnato al campo address dell'oggetto modello.

- `super().save(*args, **kwargs)`: Questa è una chiamata al metodo save della classe madre (cioè la classe Model da cui questo modello eredita). È responsabile dell'effettiva persistenza dell'oggetto nel database.

- `def __str__(self)`: questo è un metodo di Python utilizzato per definire come un'istanza di un oggetto dovrebbe essere convertita in una stringa.

- `return f"self.username self.IndirizzoAziendale"`: qui viene definita la rappresentazione in stringa dell'oggetto. Quando si tenta di convertire l'oggetto in una stringa (per esempio, per la stampa), verrà restituito un stringa formattata che include il username e l'Indirizzo Aziendale dell'oggetto (solo questi elementi verranno poi utilizzati per lo smart contract).

In altre parole, ogni volta che si salva un oggetto di questo modello, vengono generati automaticamente una nuova chiave privata e un nuovo indirizzo, che vengono poi salvati nel database assieme all'oggetto. La rappresentazione in stringa dell'oggetto includerà l' username e l'Indirizzo aziendale

[4]Nella soluzione finale nella creazione dello Smart Contract non vengono usati questi campi, ma vengono mantenuti per un implementazione futura da parte dell'azienda

4.3 Registrazione prodotto

La schermata seguente, ci mostra la visualizzazione di tutti i prodotti inseriti nel database, risulta possibile personalizzare la vista, con del codice che verrà analizzato successivamente

La schermata 4.5 è stata generata dal seguente codice:

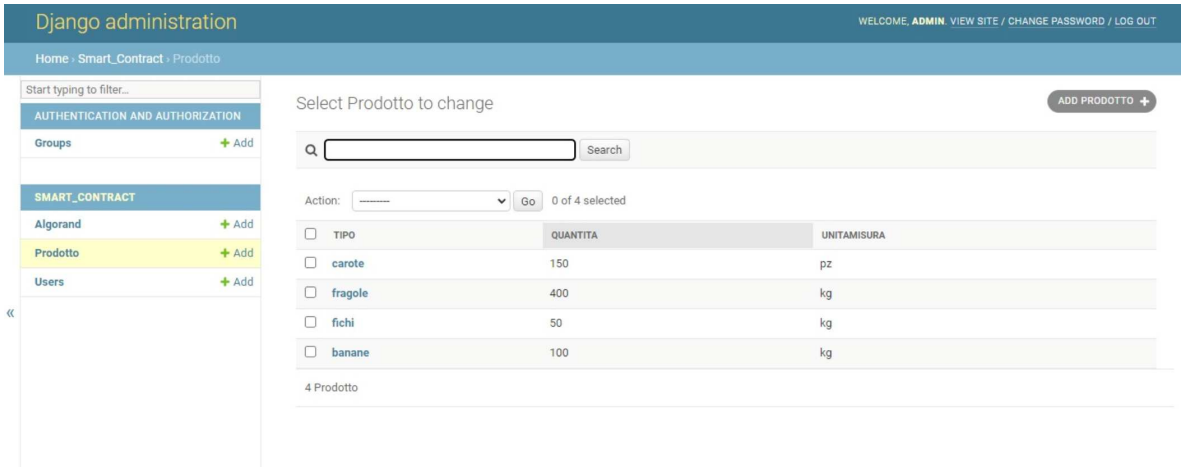

Figura 4.5: Visualizzazioni prodotti inseriti

```
1
2  class ProdottoAdmin(admin.ModelAdmin):
3      list_display = ["Tipo", "Quantita", "UnitaMisura"]
4      search_fields = ["Tipo"]
5      add_fieldsets = (
6          (None, {
7              'classes': ('wide',),
8              'fields': ('Tipo', 'Quantita', 'UnitaMisura')}
9           ),
10     )
11
12
13 admin.site.register(Prodotto,ProdottoAdmin)
```

Il codice fornisce un'interfaccia di amministrazione personalizzata per il modello Prodotto nell'applicazione Django. Utilizza la classe ProdottoAdmin, che eredita da admin.ModelAdmin, per specificare alcune personalizzazioni. Vediamo cosa fa ciascuna riga:

- `class ProdottoAdmin(admin.ModelAdmin)`: definisce una nuova classe ProdottoAdmin che eredita da admin.ModelAdmin. Questa classe servirà per personalizzare l'interfaccia di amministrazione di Django per il modello Prodotto.

- `list_display = ["Tipo", "Quantita", "UnitaMisura"]`: specifica quali campi del modello Prodotto devono essere mostrati nella lista di oggetti nell'interfaccia di amministrazione. In questo caso, verranno mostrati i campi "Tipo", "Quantita" e "UnitaMisura".

- `search_fields` = ["Tipo"]: abilita una casella di ricerca nell'interfaccia di amministrazione che permetterà agli utenti di cercare i prodotti in base al campo "Tipo".

- `add_fieldsets = (...)`: definisce i campi che saranno mostrati quando si crea un nuovo oggetto Prodotto attraverso l'interfaccia di amministrazione. In questo caso, i campi sono "Tipo", "Quantita" e "UnitaMisura".

- `admin.site.register(Prodotto,ProdottoAdmin)`: registra il modello Prodotto e la classe ProdottoAdmin personalizzata nell'interfaccia di amministrazione di Django. Questo collega effettivamente il modello Prodotto alla sua interfaccia di amministrazione personalizzata.

In altre parole , questo codice personalizza l'interfaccia di amministrazione di Django per il modello Prodotto, definendo quali campi mostrare, quali campi possono essere ricercati e quali campi vengono mostrati quando si aggiunge un nuovo Prodotto.

Di seguito prenderemo in disamina la schermata che ci consentirà di registrare un prodotto, andando a selezionare il tipo, la quantità e l'unità di misura. Questi sono i prodotti che poi verranno aggiunti al database e saranno selezionabili per andare a comporre lo smart contract.

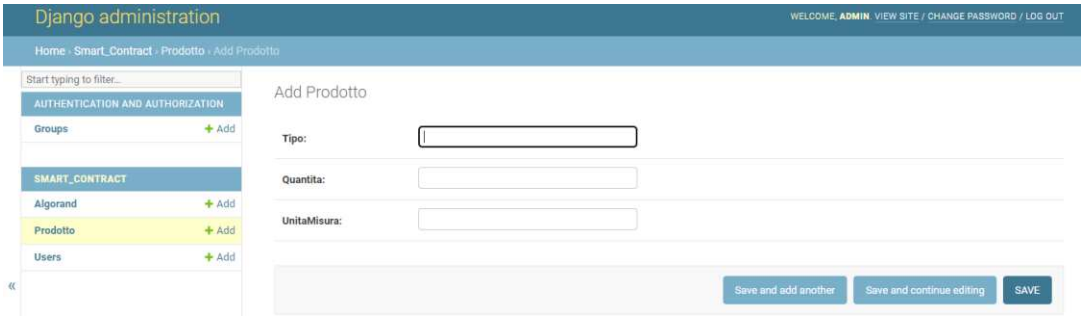

Figura 4.6: Registrazione di un nuovo prodotto

In questa fase non verrà spiegato alcun codice non essendoci sovrascrittura, ma bensì solamente il modello con il metodo di salvataggio base.

4.4 Creazione Smart Contract

Nella schermata 4.7 vediamo la schermata per l' admin nella sezione Algorand, dove si possono vedere le informazioni principali degli smart contract già precedentemente emessi.

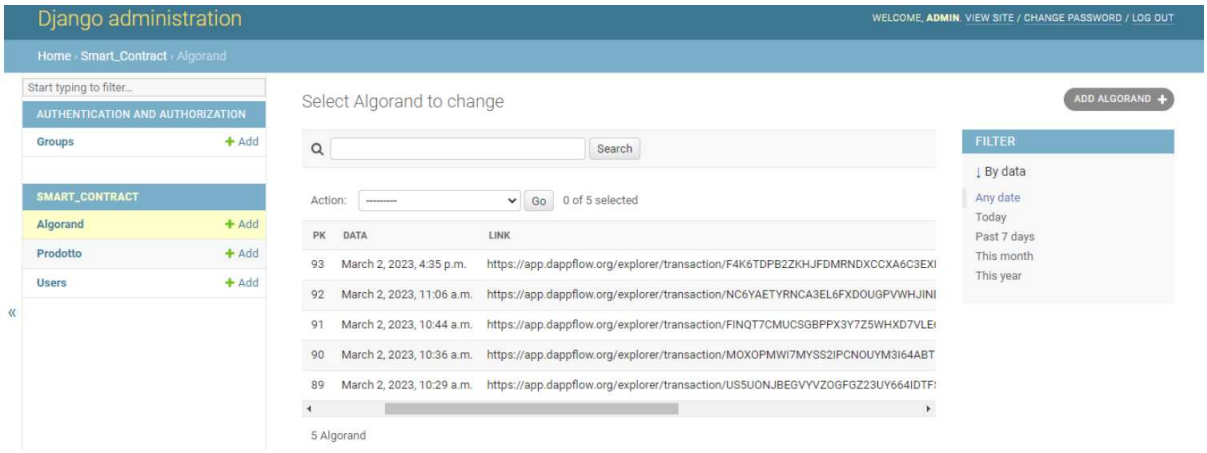

Figura 4.7: Schermata di base Algorand

La configurazione di visualizzazione viene fatta nell'apposita pagina di admin.py situata in Django nella cartella smart contract, tramite il codice che verrà analizzato nella fase successiva.

```
    class AlgorandAdmin(admin.ModelAdmin):
    list_display = ["payload", "pk", "data", "link", "user", "prodotto", "
    ↪ commento"]
    list_filter = ["data"]
    readonly_fields = ["payload", "pk", "data", "link"]
    search_fields = ["payload"]
    add_fieldsets = (
        (None, {
            'classes': ('wide',),
            'fields': ('commento')}
         ),
    )


admin.site.register(Algorand,AlgorandAdmin)
```

Andando a vedere in maniera dettagliata la personalizzazione per la visualizzazione e la gestione del modello 'Algorand' nell'interfaccia di amministrazione di Django. Vediamo cosa fanno le diverse parti:

- `class AlgorandAdmin(admin.ModelAdmin)`: questa classe eredita da 'admin.ModelAdmin' e serve per specificare diverse impostazioni personalizzate per la gestione del modello 'Algorand' nell'interfaccia di amministrazione di django.

- `list_display = ["payload", "pk", "data", "link", "user", "prodotto", "commento"]`: questa lista specifica quali campi del modello 'Algorand' saranno visualizzati nell'elenco di oggetti nell'interfaccia di amministrazione.

- `list_filter = ["data"]`: questo permette di filtrare gli oggetti 'Algorand' in base al campo 'data'. Apparirà un widget di filtro sul lato destro dell'interfaccia di amministrazione.
- `readonly_fields = ["payload", "pk", "data", "link"]`: questi campi sono resi di sola lettura nell'interfaccia di amministrazione, il che significa che non possono essere modificati direttamente.
- `search_fields = ["payload"]`: questo permette di effettuare ricerche nel modello 'Algorand' utilizzando il campo 'payload'.
- `add_fieldsets`: Questo è utilizzato per definire i fieldsets quando viene aggiunto un nuovo oggetto 'Algorand'. Qui, solo il campo 'commento' è reso disponibile per l'immissione.
- `admin.site.register(Algorand, AlgorandAdmin)`: questa linea registra il modello 'Algorand' e la classe 'AlgorandAdmin' con il sito di amministrazione di Django. Ciò permette di utilizzare le impostazioni personalizzate definite in 'AlgorandAdmin' quando si lavora con il modello 'Algorand' nell'interfaccia di amministrazione.

La pagina per la creazione dello smart contract risulta essere la seguente, si evince immediatamente come ci siano molti campi non modificabili

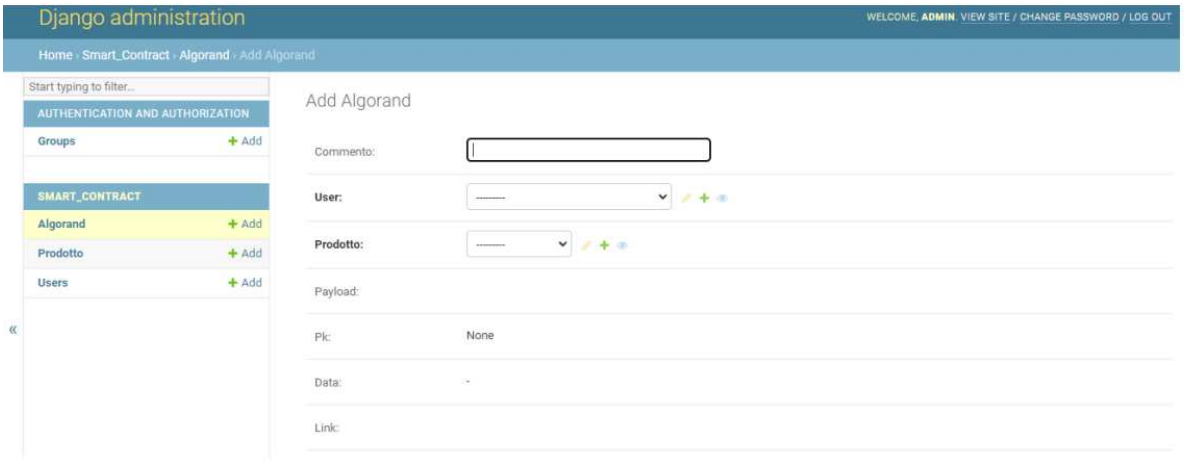

Figura 4.8: inserimento Smart Contract

- Payload: serve per avere una stringa diversa e riconoscere le varie prove, non sarà visualizzato nello smart contract finale[5]
- Pk: indica il numero di Smart Contract prodotto
- Data: indica la data di creazione dello smart contract

[5]Importante ricordare che il progetto è un P.O.C

- Link: viene dato il link di creazione in modo da poterlo andare a cercare effettivamente sullo scanner della blockchain(in questo caso essendo solo in una testnet locale servirà usare Dappflow)

I campi non modificabili servono per dare una maggiore autenticità e automazione al sistema, al fine di garantire una maggiore tracciabilità e rintracciabilità.

Invece i campi modificabili sono:

- Commento: opzionale se serve
- l'utente: selezionabile dagli utenti già creati
- Prodotto: selezionabile dagli utenti già creati

Di seguito viene riportato il codice che serve per la creazione dello smart contract, che poi andrà ad attivare il vero e proprio codice fornendogli le variabili necessarie per l'effettiva personalizzazione e il deploy sulla blockchain principale

```
        def save(self,*args, **kwargs):
        self.payload= str(payload_random(lista, 2))
        ''' inserimento smart contract '''
        first_app = FirstApp(version=8)
        first_app.dump()

        """ funzione che restituisce la data precisa al millisecondo"""
        now = datetime.datetime.now()

        date_str = now.strftime("%Y-%m-%d %H:%M:%S.%f")

        acct = sandbox.get_accounts()[0]

        app_client = client.application_client.ApplicationClient(
            client=sandbox.clients.get_algod_client(),
            app=first_app,
            sender=acct.address,
            signer=acct.signer,
        )

        app_client.create()

         """ smart contract orario"""
        orario: str = f" emesso nel giorno  {(date_str)}"


        return_obj = app_client.call(
            method=FirstApp.logger,
```

```
            address=(f"{self.user}"),
            merce=(f"{self.prodotto}"),
                a=arrivo_fine,
                Tragitto=Tragittof,
                c=orario,
            )

        self.link= f"https://app.dappflow.org/explorer/transaction/{return_obj
    ↪ .tx_id}"

        super().save(*args, **kwargs)


    def __str__(self):
        return f"{self.payload} {self.data.strftime('%d/%m/%Y  %H:%M:%S')}"

    class Meta():
        verbose_name= "Algorand"
        verbose_name_plural= "Algorand"
```

- `def save(self, *args, **kwargs)`: override del metodo save per questo modello specifico. Nel momento che viene chiamato save() su un'istanza di Algorand, questo metodo viene eseguito.
- `self.payload = str(payload_random(lista, 2))`: Il codice genera un payload randomico attraverso la funzione payload_random() e lo converte in una stringa.
- `first_app = FirstApp(version=8)`: viene creato un oggetto di tipo FirstApp.
- `first_app.dump()`: rende visibili le varibili.
- `now = datetime.datetime.now()`: ottiene l'orario corrente fino al millisecondo.
- `date_str = now.strftime("%Y-%m-%d %H:%M:%S.%f")`: Converte l'orario in una stringa.
- `acct = sandbox.get_accounts()[0]`: ottiene il primo account dalla sandbox di Algorand.
- `app_client = client.application_client.ApplicationClient(...)`: crea un client per l'applicazione Algorand.
- `app_client.create()`: crea un nuovo smart contract su Algorand.
- `return_obj = app_client.call(...)`: esegue una chiamata all'applicazione o allo smart contract su Algorand con diversi parametri.

- `self.link = f"https://app.dappflow.org/explorer/transaction/return_obj.tx_id"`: Salva un link della transazione su dappflow, poi sarà visibile sulla schermata di visualizzazione da parte dell'admin.

- `super().save(*args, **kwargs)`: esegue il metodo save originale, salvando tutte le modifiche nel database.

- `return f"self.payload self.data.strftime('%d/%m/%Y %H:%M:%S')"`: definisce come convertire un'istanza di questa classe in una stringa. Usa il campo payload e il campo data (formattato).

- `Class Meta` : qui vengono definite alcune metainformazioni per la classe.

4.5 Smart Contract

l'effettivo dialogo tra l'ambiente Sandbox e Sjango avviene in un altra pagina di codice, precisamente in tirocinio_smart_contract_generator/contratto/Contract.py.

Il codice risulta essere il seguente:

```
from beaker import *
from pyteal import *
    class FirstApp(Application):

    @create
    def create(self):
        return Approve()

    @external
    def logger(self, address: abi.String, merce: abi.String, a: abi.String,
    ↪ Tragitto: abi.String, c: abi.String, *,
                output: abi.String):
        log_value = "Smart contract realizzato da Integrity Key per conto di [
    ↪ NOME] ".capitalize()
        output_expr = output.set(Bytes(log_value))
        return Seq(
            Log(address.get()),
            Log(merce.get()),
            Log(a.get()),
            Log(Tragitto.get()),
            Log(c.get()),
            output_expr
        )
```

Di seguito verrà data una spiegazione approfondita del codice:

- importazione delle librerie: beaker e pyteal[6]
- class FirstApp(Application): la classe `FirstApp` è dichiarata come sottoclasse di `Application`, che è una classe base per smart contract su Algorand.

```
    @create
    def create(self):
        return Approve()

```

La funzione `create` con il decorator[7] `@create`, fa in modo che questa funzione viene chiamata quando lo smart contract viene creato. La funzione ritorna `Approve()`, che approva la creazione del contratto.

```
    @external
    def logger(self, address: abi.String, merce: abi.String, a: abi.
    ↪ String, Tragitto: abi.String, c: abi.String, *,
               output: abi.String):

```

Questa è la funzione `logger`, che è marcata come una funzione "esterna" utilizzando il decorator `@external`. Prende diversi argomenti stringa (come `address`, `merce`, etc.) e un argomento chiamato `output`.

- Il corpo della funzione esegue diverse operazioni:
  - imposta il valore del log: crea una stringa che verrà utilizzata per impostare un valore nel log. `log_value = "Smart contract realizzato da Integrity Key per conto di ↪ [NOME] ".capitalize()`
  - imposta l'output: utilizza il valore del log per impostare un "output" `output_expr ↪ = output.set(Bytes(log_value))`
- Sequenza di azioni: la funzione ritorna una sequenza di azioni da eseguire.

```
    return Seq(
        Log(address.get()),
        Log(merce.get()),
        Log(a.get()),
        Log(Tragitto.get()),
        Log(c.get()),
        output_expr
    )

```

[6] sono già state spiegata in 1.3.4 e 1.3.3

[7] Un decorator è un design pattern in Python che permette di aggiungere nuove funzionalità a una funzione o a una classe esistente, senza modificarne il codice sorgente

- `Log(address.get()), Log(merce.get()), ...`: queste righe aggiungono i valori degli argomenti forniti al log.
- `output_expr`: questa è l'ultima espressione nella sequenza e imposta l'output dello smart contract.

Figura 4.9: Smart contract finale

La schermata sopra riportata la 4.9 riporta quello che è l'output finale dello smart contract, visibile tramite dappflow.

4.6 Conclusioni

L'analisi è iniziata dalla strutturazione della base di dati, delineando come le informazioni sono organizzate, memorizzate e gestite. Questo elemento è cruciale, poiché una progettazione accurata della base di dati può notevolmente influenzare le prestazioni e l'efficacia dell'intero sistema.

Successivamente, sono state descritte tutte le operazioni che è possibile eseguire all'interno dell'applicazione. Sono stati anche illustrati gli algoritmi e le metodologie utilizzate per implementare le funzionalità chiave, permettendo così una comprensione più profonda del funzionamento interno dell'applicazione.

Oltre a ciò, l'uso di frammenti di codice ha avuto lo scopo di rendere tangibili i concetti discussi, facilitando la comprensione e fornendo un riferimento utile per future fasi di sviluppo o manutenzione.

CAPITOLO 5

Conclusioni e sviluppi futuri

Conclusioni

A conclusione di questo elaborato e dopo aver introdotto tutte le tecnologie utilizzate, aver espresso tutti gli obiettivi con i vari requisiti funzionali e non funzionali, si può concludere che le aspettative sono state raggiunte appieno e nel tempo stabilito. Per svolgere tutto il lavoro assegnato è stato impiegato tantissimo tempo, il progetto ha avuto inizio tramite lo studio di Django, framework non conosciuto e senza avere nessuna base di tecnologie web, di conseguenza non è stato facile mettere insieme tutti i pezzi di questo puzzle. Lo studio di Django, è stato personale e prima dell'inizio del periodo effettivo del tirocinio, questo studio ha impiegato circa una settimana. Dopo la prima settimana di esercizio in Django, la seconda settimana è stata usata per studiare le diverse tecnologie a disposizione, analizzando moltissime blockhain e framework, alla fine è stato deciso di usare Algorand e Beaker per le caratteristiche espresse precedentemente. La terza settimana è stata usata per settare tutto l'ambiente di sviluppo e utilizzare tutte le tecnologie insieme, quindi installando tutti i pacchetti e capire come creare un primo esempio di smart contract, in questa settimana ci sono stati più problemi di compatibilità, far funzionare Docker con la sandbox di Algorand, con tutti gli altri pacchetti insieme ha richiesto diverso tempo, avendo molti problemi per riuscire ad avere un installazione completa e funzionante. La quarta settimana è servita per creare tutta l'infrastruttura per il sistema IntegrityKEY su Django, progettando e sviluppando tutto il database e facendo funzionare correttamente tutte le funzioni. Nella quinta e ultima settimana è stato creato lo smart contract con i vari requisiti richiesti dall'azienda e creata la connessione tra Django e tutto il sistema di pacchetti necessari per la distribuzione dello smart contract. In questa settimana l'ostacolo maggiore è stato riscontrato per ricevere il link corretto dello smart contract, deployato sulla localnet di sandbox, il problema riscontrato è stato risolto andando a spostare la chiamata dello smart contract.

Come si evince, imparare a maneggiare tutte queste tecnologie simultaneamente non è stato affatto semplice, lo sforzo maggiore è stato riscontrato nell'andare a cercare, trovare, capire quale tecnologia servisse e farle funzionare simultaneamente. Essendo un P.O.C. la maggior parte delle cose viste ad esclusione del framework Django sono completamente nuove e da scoprire, di conseguenza è stato molto difficile trovare problemi riscontrati da altri sviluppatori, per riuscire a risolvere i bug

Figura 5.1: Contributo realizzato nel progetto in esame

È possibile concludere che lo sviluppo del progetto è arrivato al termine dimostrando la fattibilità di queste strade e tecnologie, che possono realmente servire nel mondo reale, anche se anche sono tecnologie giovani, hanno già molto potenziale.

Sviluppi futuri

Il progetto porta con sé una potenziale minaccia per gli enti certificatori, che verrebbero rimpiazzati da una blockchain sicura, affidabile e trasparente, e anche molto più economica. Spero veramente che in futuro IntegrityKEY arrivi allo sviluppo reale di questo progetto e porti a termine quello che è stato solamente uno studio di fattibilità, che ha dato un esito positivo. Il progetto porta con sé tanta innovazione con tecnologie ancora da sviluppare, scoprire e comprendere, portando veramente qualcosa di nuovo sull'ambito della tracciabilità e rintracciabilità.

5.1 bibliografia

- https://docs.djangoproject.com/en/4.1/
- https://developer.algorand.org/
- https://developer.algorand.org/tutorials/create-and-test-smart-contracts-using-python/
- https://algorand-devrel.github.io/beaker/html/index.html
- https://pureinfotech.com/install-windows-subsystem-linux-2-windows-10/#install_wsl_command_2004_windows10
- https://github.com/algorand/sandbox
- https://github.com/algorand/pyteal
- https://github.com/algorand-devrel/beaker
- https://developer.algorand.org/docs/get-started/algokit/
- https://algoexplorer.io
- https://github.com/algorand
- https://www.youtube.com/watch?v=5O1-mqCGcc0
- https://www.docker.com/get-started
- https://dappflow.org/
- https://app.dappflow.org/explorer/home
- https://bank.testnet.algorand.network/
- https://testnet.algoexplorer.io/tx/Q5RUEJQ4ELTOMPBRUW27DV56PWN2ATIFFACA46HSYVQ2U4IBNDPA[1]
- https://algodea-docs.bloxbean.com/algorand-project-structure
- https://www.youtube.com/watch?v=iwViloTnMLM
- https://algodea-docs.bloxbean.com/
- https://algodea-docs.bloxbean.com/pyteal/untitled

[1]Per trovare la strada corretta ho provato a creare contratti sulla tesnet pubblica, questo è uno smart contract realizzato, con l'unica utilità di essere un contatore

5.2 Ringraziamenti

Doveroso da parte mia ringraziare le persone a me più vicine, coloro che mi hanno sostenuto nelle scelte e lasciato libero nelle mie decisioni.

Un enorme ringraziamento va alla mia famiglia, senza di loro tutto questo non sarebbe possibile, in particolare mio Padre Riccardo e mia Madre Monica, per non avermi fatto mancare nulla, ma allo stesso tempo non avermi viziato, e reso quello che sono, un sostengo morale ed economico costante, e la motivazione forte per renderli fieri dei sacrifici che fanno e faranno ogni giorno per me.

Un ringraziamento fraterno a mia Sorella Giulia, che nonostante sia piccola mi ha sempre dato ottimi consigli, sostenuto, sopportato e preparato tanti caffè nei periodi intensi.

Ringraziamento di cuore a Sofia, colei che mi ha supportato nelle scelte e capito nel momento del bisogno, lasciato i miei spazi grazie alla sua grande maturità, stretto forte e protetto quando gli avvenimenti non erano favorevoli e negli errori non giudicato, ma aiutato, la mia Fidanzata, ma allo stesso tempo la mia migliore Amica.

Un ringraziamento a un Fratello non di sangue, Nicolò Bartolini, collega con cui ho avuto modo di stringere una forte amicizia, e ho avuto il piacere di condividere il lavoro su moltissimi progetti, colui che mi ha insegnato a studiare in maniera intelligente, colui con cui posso parlare di tutto ed essere compreso.

Un ringraziamento a tutti gli amici dell'università, in particolare a coloro che si aiutano tra loro passando esercizi e appunti, favorendo un clima di cooperazione.

Vorrei, infine, ringraziare il mio relatore, il Professor Simone Fiori, per la sua disponibilità e l'opportunità a me concessa, l'azienda IntegrityKEY che mi ha permesso di mettermi a lavorare su un progetto stimolante e innovativo, un ringraziamento ad Andrea Fiorani il mio tutor aziendale che mi ha seguito durante tutto lo sviluppo, persona con cui ho avuto il piacere di stringere anche un' amicizia e con cui sicuramente svolgerò altri progetto in futuro.