roundabout

Roundabout - Etherless

Analisi dei requisiti

Versione	0.1.0
Approvazione	
Redazione	Egon Galvani
	Antonio Zlatkovski
Verifica	Feim Jakupi
Stato	Non approvato
Uso	Esterno
Destinato a	Roundabout
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin

Descrizione
Analisi dei requisiti del gruppo Roundabout per la realizzazione del progetto Etherless

team.roundabout.13@gmail.com

Registo delle modifche

Versione	Data	Nominativo	Ruolo	Descrizione
0.1.0	2020-03-25	Feim Jakupi	*Verificatore*	Verifica sezione §1, §2
0.0.3	2020-03-18	Egon Galvani	*Analista*	Stesura §2, §3.1, §3.2.1
0.0.2	2020-03-18	Antonio Zlatkovski	*Analista*	Scrittura dell'introduzione del documento
0.0.1	2020-03-14	Egon Galvani	*Analista*	Creata struttura del documento in LaTeX

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il presente documento ha l'obiettivo di individuare le funzionalità e i casi d'uso previsti dal progetto Etherless, proposto dall'azienda Red Babel. Le informazioni qui riportate, individuate da un'approfondita analisi del capitolato stesso e dai successivi incontri con il proponente, rappresentano la base di partenza per la successiva fase di progettazione.

1.2 Scopo del prodotto

Si svuole sviluppare una piattaforma cloud che consenta agli sviluppatori di fare il deploy di funzioni Javascript e gestisca il pagamento per la loro esecuzione tramite la piattaforma Ethereum. Il prodotto finale prevede quindi l'integrazione di due tecnologie, Serverless ed Ethereum.
Il lato Serverless si occupa dell'esecuzione delle funzioni fornite dagli sviluppatori. Tali funzioni vengono salvate ed eseguite in un servizio cloud esterno, quale Amazon Web Services.
La richiesta di utilizzo di una funzione e il successivo pagamento vengono invece gestiti tramite la piattaforma Ethereum sfruttando gli smart contracts. Il pagamento viene effettuato in ETH. Una percetuale significativa di ogni pagamento viene riservata agli amministratori del servizio.
Lo sviluppatore e l'utente finale interagiscono con il prodotto tramite una CLI che prevede alcuni comandi intuitivi.

1.3 Documenti complementari

1.3.1 Glossario

I termini tecnici utilizzati in questo documento sono contrassegnati da una 'G' a pedice ed ulteriormente approfonditi nel Glossario denominato "Glossario Esterno 1.0.0", disponibile in allegato al presente documento.

1.4 Riferimenti

1.4.1 Normativi

- **Norme di progetto**: *Norme di progetto v1.0.0*;
- **Verbale esterno con il proponente**: *VE_2020_03_18*;
- **Capitolato d'appalto Etherless**: https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C2.pdf.

1.4.2 Informativi

- **Studio di fattibilità**: *Studio di fattibilità v1.0.0*;
- **Sito ufficiale Ethereum**: https://ethereum.org/;
- **Sito del framework Serverless**: https://serverless.com/;
- **Amazon Web Services**: https://aws.amazon.com/.

2 Descrizione generale

2.1 Obiettivi del prodotto

Il progetto *Etherless* ha come obiettivo finale creare una piattaforma per permettere agli sviluppatori di fare il deploy di funzioni JavaScript, preoccupandosi solamente della relativa codifica e non dell'architettura sottostante. Allo stesso tempo, tali funzioni vengono messe a disposizione agli altri utenti, che possono eseguirle e pagare secondo un modello Caas (Computation As a Service), cioè solamente per il tempo e le risorse richieste dalla loro esecuzione.

2.2 Funzionalità del prodotto

L'applicativo deve fornire la possibilità agli sviluppatori di caricare nel cloud le proprie funzioni e renderle disponibili secondo un'ideologia Faas (Function As A Service). Gli utenti finali possono usufruire di tali servizi pagando; in questo modo gli sviluppatori non si devono preoccupare della gestione dell'infrastruttura alla base dei servizi e possono guadagnare ad ogni esecuzione di una funzione da loro caricata.
Nello specifico:

- gli utenti finali e gli sviluppatori possono:
  a. autenticarsi all'interno della rete Ethereum;
  b. eseguire una funzione presente della piattaforma e visualizzarne i risultati;
  c. elencare tutte le funzioni disponibili nella piattaforma;
  d. visualizzare i dettagli di una determinata funzione;
  e. visualizzare la propria cronologia di esecuzione di funzioni;
  f. ricercare funzioni in base ad un termine di ricerca.
- gli sviluppatori possono:
  a. caricare all'interno della piattaforma delle proprie funzioni JavaScript;
  b. eliminare una funzione da loro precedentemente caricata;
  c. modificare le informazioni e il codice relativo ad una loro funzione.

2.3 Analisi della struttura

Il prodotto si divide nelle seguenti parti:

- **Etherless-cli:** interfaccia a linea di comando tramite cui i vari utenti della piattaforma interagiscono con Etherless;
- **Etherless-smart:** insieme di smart-contract che si occupano della gestione della comunicazione tra *Etherless-cli* ed *Etherless-server* e del pagamento richiesto per l'esecuzione delle funzioni;
- **Etherless-server:** si occupa di ascoltare gli eventi trasmessi da *Etherless-smart* e di avviare l'esecuzione delle funzioni così richieste. I risultati ottenuti vengono inviati tramite un ulteriore evento nella blockchain e mostrati all'utente attraverso *Etherless-cli*;

2.3.1 Comandi disponibili

Etherless-cli mette a disposizione dell'utente i seguenti comandi:

- **etherless-init:** avvio dell'applicativo;
- **etherless-login:** esecuzione della procedura di autenticazione all'interno della rete Ethereum;
- **etherless-signup:** creazione di un nuovo wallet all'interno della rete Ethereum;
- **etherless-logout:** esecuzione del logout dalla propria utenza;
- **etherless-whoami:** visualizzazione dell'indirizzo associato all'utenza corrente;
- **etherless-list:** elenco di tutte le funzioni disponibili all'interno della piattaforma *Etherless*;
- **etherless-deploy:** esecuzione del deploy di una determinata funzione;
- **etherless-run:** esecuzione di una funzione e visualizzazione del relativo risultato;
- **etherless-mylist:** visualizzazione del nome delle funzioni di cui l'utente corrente ha eseguito il deploy;
- **etherless-info:** visualizzazione di informazioni più dettagliate di una determinata funzione;
- **etherless-search:** elenco delle funzioni con nome simile ad un termine inserito;
- **etherless-delete:** eliminazione di una determinata funzione;
- **etherless-logs:** visualizzazione dei log di esecuzione di una specifica funzione;
- **etherless-history:** visualizzazione della cronologia di esecuzione dell'utente corrente;
- **etherless-exit:** chiusura dell'applicativo.

2.3.2 Vincoli implementativi

Viene richiesto che:

- gli smart-contract possano essere aggiornati;
- *Etherless* sia sviluppato utilizzando Typescript 3.6, in particolare facendo riferimento all'approccio di promise/async-await;
- typescript-eslint deve essere utilizzato come rinforzo di ESLint durante il processo di sviluppo;
- *Etherless-server* deve essere implementato usando il framework Serverless.

2.3.3 Ambienti di esecuzione

Viene inoltre richiesto che il progetto possa funzionare nei seguenti ambienti di esecuzione:

- **Locale:** viene simulata una rete Ethereum all'interno della macchina locale, per tale scopo può essere utilizzata la rete Ethereum testrpc messa a disposizione da Truffle;
- **Test:** ambiente di test, in cui vengono eseguiti i test di verifica; la rete può coincidere con quella usata in ambiente locale;
- **Staging:** ambiente pubblicamente accessibile, per tale scopo può essere usata la testnet Ropsten di Ethereum;
- **Produzione:** non richiesto, ma il progetto deve essere pronto per la produzione. In questo caso si fa riferimento alla Ethereum mainnet.

2.3.4 Vincoli generali

L'utente, per usufruire del servizio, deve possedere una connessione internet e aver installato Node.js e il modulo relativo a *Etherless-cli*.
Per la gestione del servizio, e quindi l'esecuzione di *Etherless-server*, oltre ai requisiti già indicati è necessario avere un'utenza AWS e aver completato correttamente la relativa configurazione.

3 Casi d'uso

3.1 Attori dei casi d'uso

3.1.1 Attori primari

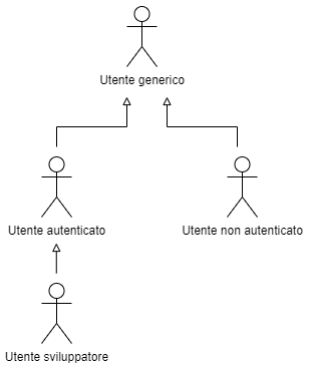

Figura 3.1.1: Gerarchia degli attori

Utente generico
Si riferisce ad un utente generico che ha avviato l'applicativo.

Utente non autenticato
Si riferisce ad un utente generico che non ha ancora effettuato l'autenticazione all'interno della rete Ethereum.

Utente autenticato
Si riferisce ad un utente generico che si è autenticato nel sistema tramite il comando di login. Ciò implica che sia in possesso di una chiave o una frase mnemonica valida all'interno della rete Ethereum.

Utente sviuppatore
Si riferisce ad un utente autenticato che ha svolto il deploy di almeno una sua funzione.

3.1.2 Attori secondari

Ethereum network
Piattaforma decentralizzata utilizzata per la gestione dell'autenticazione, dei pagamenti e della comunicazione tra i vari moduli della piattaforma *Etherless*.

3.2 Elenco dei casi d'uso

In questa sezione vengono elencati tutti i casi d'uso individuati. Ogni caso d'uso rappresenta uno scenario per uno o più attori, ed è descritto tramite degli appositi diagrammi dei casi d'uso.

Operazioni utenti

Di seguito sono riportati tutti i casi d'uso aventi come attore primario l'utente generico, l'utente non autenticato o l'utente autenticato.

3.2.1 Visualizzazione guida introduttiva

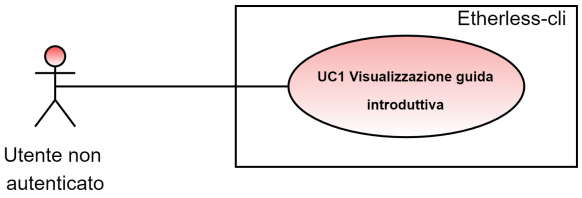

Figura 3.2.1: UC1 - Visualizzazione guida introduttiva

- **Attori primari:** utente generico
- **Descrizione:** l'utente visualizza una guida contenente una breve spiegazione per ogni comando disponibile in *Etheress-cli*. Tale guida ha lo scopo di aiutare l'utente a procedere nell'utilizzo dell'applicativo;
- **Scenario principale:** l'utente avvia l'applicativo e visualizza la guida;
- **Precondizione:** l'utente avvia l'applicativo tramite il comando "etherless-init" e quest'ultimo si apre correttamente;
- **Postcondizione:** il sistema fornisce all'utente tutte le informazioni necessarie per procedere con l'utilizzo dell'applicativo.

3.2.2 Registrazione

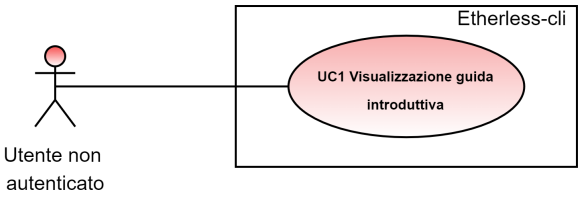

Figura 3.2.2: UC1 - Visualizzazione guida introduttiva

- **Attori primari:**
- **Descrizione:**

- **Scenario principale:**
- **Precondizione:**
- **Postcondizione:**

Operazioni sviluppatori

Di seguito sono riportati tutti i casi d'uso aventi come attore primario l'utente sviluppatore.

4 Requisiti

Ogni requisito è composto dai seguenti elementi:

- **Codice identificativo**: ogni codice identificativo è univoco e segue la seguente codifica:

  R[Importanza][Tipologia][Codice]

  Dove:

  - **Importanza**: indica il grado di importanza del requisito ai fini del progetto. Può assumere i valori:
    - **1**: requisito obbligatorio ai fini del progetto, irrinunciabile per gli stakeholders;
    - **2**: requisito desiderabile: non strettamente necessario ai fini del progetto ma che porta valore aggiunto;
    - **3**: requisito opzionale, contrattabile più avanti nel progetto.
  - **Tipologia**: classe a cui appartiene il requisito in questione. Può assumere i valori:
    - **F**: funzionale;
    - **P**: prestazionale;
    - **Q**: qualitativo;
    - **V**: vincolo.
  - **Codice**: identificatore univoco del requisito.

  Il codice stabilito secondo la convenzione precedente, una volta associato ad un requisito, non può più essere modificato.

- **descrizione**: breve descrizione del requisito, strutturata in maniera da evitare ambiguità;
- **classificazione**: indica il grado di importanza del requisito considerato. Sebbene tale informazione sia già presente nell'identificativo, la sua ripetizione rende la lettura più semplice e scorrevole;
- **fonti**:
  - *capitolato*: requisito indicato nel capitolato;
  - *interno*: requisito individuato dagli analisti;
  - *caso d'uso*: il requisito è stato estrapolato da uno o più casi d'uso. In questo caso vengono riportati gli identificativi dei casi d'uso considerati;
  - *verbale*: si tratta di un requisito individuato a seguito di un incontro tra i membri del gruppo o di una richiesta di chiarimento con il proponente. In questo caso è riportato il codice identificativo presente nella tabella delle decisioni dei verbali considerati.

4.1 Requisiti funzionali

Tabella 4.1.1: (continua)

Requisito	Descrizione	Classificazione	Fonti

4.2 Requisiti di qualità

Tabella 4.2.1: (continua)

Requisito	Descrizione	Classificazione	Fonti
R1Q1	La progettazione e la codifica devono rispettare le norme e le metriche definite nei documenti *Norme di Progetto v1.0.0* e *Piano di Qualifica v1.0.0*	Obbligatorio	Interno
R1Q2	Il sistema deve essere pubblicato con licenza MIT	Obbligatorio	Capitolato
R1Q3	Il codice sorgente di *Etherless* deve essere pubblicato e versionato usando Github o GitLab	Obbligatorio	Capitolato
R1Q4	Deve essere redatto un manuale sviluppatore	Obbligatorio	Capitolato
R1Q4.1	Il manuale sviluppatore deve contenere le informazioni per eseguire e fare il deploy dei moduli	Obbligatorio	Capitolato
R1Q5	Deve essere redatto un manuale utente	Obbligatorio	Capitolato
R1Q5.1	Il manuale utente deve contenere tutte le informazioni necessarie all'utente finale per utilizzare correttamente il sistema	Obbligatorio	Capitolato
R1Q6	La documentazione per l'utilizzo del software deve essere scritta in lingua inglese.	Obbligatorio	Verbale 2020-03-18, VE_1.2
R1Q7	Nella scrittura del codice JavaScript deve essere seguita la guida sullo stile di programmazione Airbnb JavaScript style guide	Obbligatorio	Verbale venerdì ?

Tabella 4.2.1: (continua)

Requisito	Descrizione	Classificazione	Fonti
R1Q8	Lo sviluppo del codice JavaScript deve essere supportato dal software di analisi statica del codice ESLint	Obbligatorio	Capitolato

4.3 Requisiti di vincolo

Tabella 4.3.1: (continua)

Requisito	Descrizione	Classificazione	Fonti
R1V1	Gli smart contract devono essere scritti in Solidity	Obbligatorio	Verbale
R1V2	Gli smart contract devono poter essere aggiornati	Obbligatorio	Capitolato
R1V3	L'applicativo deve essere sviluppato utilizzando TypeScript 3.6	Obbligatorio	Capitolato
R1V3.1	Deve essere utilizzato il meccanismo delle promise/async-await come approccio principale	Obbligatorio	Capitolato
R1V4	Il modulo *Etherless-server* deve essere implementato utilizzando il Framework Serverless	Obbligatorio	Capitolato
R1V5	Il progetto deve utilizzare i seguenti ambienti di sviluppo: ambiente di sviluppo locale, ambiente di testing e ambiente di staging	Obbligatorio	Capitolato
R2V5.1	Gli ambienti per la fase di sviluppo locale e testing possono fare utilizzo della rete testrpc fornita dal framework Truffle	Desiderabile	Capitolato
R2V5.2	Per la fase di staging è desiderabile l'utilizzo della rete Ethereum Ropsten	Desiderabile	Capitolato

Tabella 4.3.1: (continua)

Requisito	Descrizione	Classificazione	Fonti
R1V5.3	Durante la fase di staging l'applicativo deve essere pubblicamente accessibile	Obbligatorio	Capitolato
R1V5.4	Al termine del progetto il prodotto deve essere pronto per la produzione	Obbligatorio	Capitolato
R3V5.4.1	L'ambiente di produzione deve fare utilizzo dell'Ethereum main network	Opzionale	Capitolato
R3V6	Il sistema di pagamento deve seguire un meccanismo di escrow	Opzionale	Verbale

4.4 Requisiti funzionali

Non sono stati individuati requisiti prestazionali in quanto la maggior parte delle funzionalità messe a disposizione da *Etherless* necessita dell'interazione con una rete Ethereum. Le operazioni che avvengono all'interno di tale rete hanno un tempo di soddisfacimento che dipende dal carico della rete nel momento in cui viene fatta la richiesta. Nel caso di operazioni che portano ad una modifica nello stato del contratto, deve essere inoltre preso in considerazione anche il quantitativo di Ether pagati per unità di Gas.
Per le motivazioni appena descritte, i tempi di risposta della rete Ethereum e quindi di completamento delle operazioni messe a disposizione da *Etherless*, non risultano essere costanti o prevedibili con precisione.

4.5 Tracciamento

4.5.1 Fonte - Requisiti

4.5.2 Requisito - Fonti

4.6 Considerazioni