Laurea Triennale in Informatica - Università di Salerno -
Sistema di autenticazione sulla rete Blockchain Ethereum

KRYPTOAUTH

System Design Document

KryptoAuth

Riferimento	
Versione	1.0
Data	29/07/2022
Presentato da	Montefusco Alberto

Sommario

1. Introduzione

1.1 Scopo del Sistema

Come meglio illustrato nel paragrafo 1.1 del documento "RAD", la realizzazione di KryptoAuth ha l'obiettivo di offrire maggiore sicurezza durante l'operazione di autenticazione sfruttando la tecnologia Blockchain Ethereum.

1.2 Obiettivi di Design Sistema

Di seguito, vengono descritti gli obiettivi di design che devono essere rispettati nella realizzazione del sistema. Tali obiettivi sono organizzati in cinque categorie:

1. **Criteri di performance**
   - **Tempo di risposta**
     - Le operazioni effettuate dall'utente o dall'amministratore devono essere immediate.

2. **Criteri di dependability**
   - **Robustezza**
     - Gli input non validi inseriti dall'utente devono essere segnalati con messaggi d'errore.
   - **Affidabilità**
     - Il Sistema deve garantire che gli utenti vengano registrati una sola volta.
     - Il Sistema deve consentire solo agli amministratori di modificare il ruolo degli altri utenti.
     - Il Sistema deve consentire la disattivazione dell'account solo agli utenti "User" o all'amministratore che è registrato nella sessione.
   - **Disponibilità**
     - Il Sistema deve essere disponibile H24.
   - **Tolleranza agli errori**
     - In caso l'utente inserisce input non validi, gli viene offerta la possibilità di ripetere l'operazione e viene impedita la transizione sulla Blockchain.

- **Sicurezza**
  - Tutte le password devono essere cifrate sulla Blockchain.

3. **Criteri di costo**
   - **Costi di sviluppo**
     - Il costo complessivo del progetto ammonta ad un massimo di 275 ore.

4. **Criteri di manutenzione**
   - **Estensibilità**
     - Il Sistema deve essere progettato in modo tale che sia possibile aggiungere moduli.
   - **Adattabilità**
     - Il Sistema deve essere progettato su una struttura generica in modo da poter essere utilizzato in altri siti.
   - **Portabilità**
     - Il Sistema deve essere fruibile su tutti i dispositivi mobile e desktop in maniera indipendente dal sistema operativo o dall'hardware utilizzato.
   - **Tracciabilità dei requisiti**
     - La tracciabilità dei requisiti deve essere garantita da una matrice di tracciabilità che permette di ricondurre ogni artefatto al proprio requisito.

5. **Criteri dell'utente finale**
   - **Usabilità**
     - Il Sistema deve avere un'interfaccia semplice e immediata in modo da consentire un'interazione rapida ed efficiente, così da ridurre i tempi di controllo.

1.2.1 Design Trade-off

Tempo di rilascio Vs funzionalità

Per ottenere un buon prodotto finale è necessario completare tutte le funzionalità richieste della Web DApp.

Usabilità Vs memoria

Poiché non ci sono dati da memorizzare in uno storage di persistenza, l'obiettivo del Sistema è rispettare l'alta usabilità.

1.3 Panoramica

Dopo questa prima sezione di introduzione del presente documento, il punto 2 descriverà brevemente l'architettura del Sistema proposto. In particolare, questa sezione descriverà la decomposizione in sottosistemi, la corrispondenza tra hardware e software, il controllo degli accessi, la sicurezza e, infine, al punto 4 avremo una descrizione dei termini adottati in questo documento.

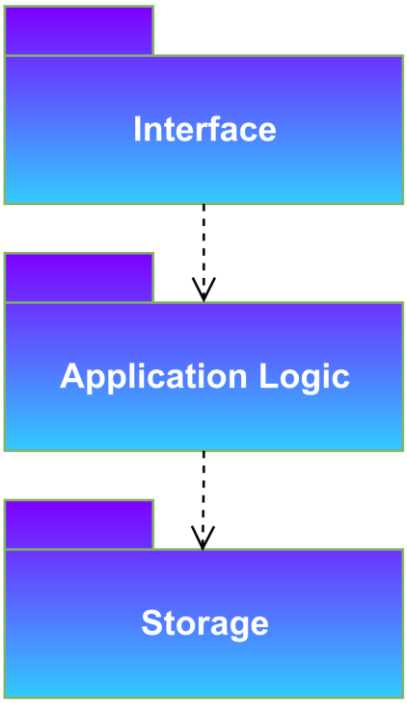
2. Architettura del Sistema proposto

Il Sistema proposto è una Web DApp che vuole rendere più sicura l'operazione di autenticazione da parte di un utente. L'architettura software scelta per la realizzazione del Sistema è la Three Tier, la quale, fornendo la possibilità di eseguire ciascun tier sulla propria infrastruttura, offre numerosi vantaggi, tra cui uno sviluppo più veloce e una maggiore scalabilità, affidabilità e sicurezza.

Questo pattern architetturale si compone di tre livelli:

1. **Interface**: è il tier di presentazione, dunque delle interfacce utente. Si occupa di visualizzare le informazioni all'utente e di raccogliere informazioni da quest'ultimo;
2. **Application Logic**: è il tier della logica di business dell'applicazione. Si occupa di elaborare le informazioni raccolte nel tier Interface e di aggiungere, rimuovere o modificare i dati nel tier Storage;
3. **Storage**: è il tier dell'accesso ai dati. In particolare, nell'ambito di questo progetto non è stato necessario l'utilizzo di uno storage di persistenza come un database, in quanto tutte le operazioni di cambio stato di un'entità sono state svolte effettuando delle transazioni sulla Blockchain Ethereum.

Nel Sistema realizzato con la suddetta architettura tutte le comunicazioni passano attraverso l'Application Logic tier; l'Interface tier e lo Storage tier non comunicano mai direttamente tra loro.



2.1 Decomposizione in Sottosistemi

Il sistema si compone in totale di 8 componenti, suddivisi tra i layer Interface, Application Logic e Storage.

L'**Application Logic** layer contiene le componenti:

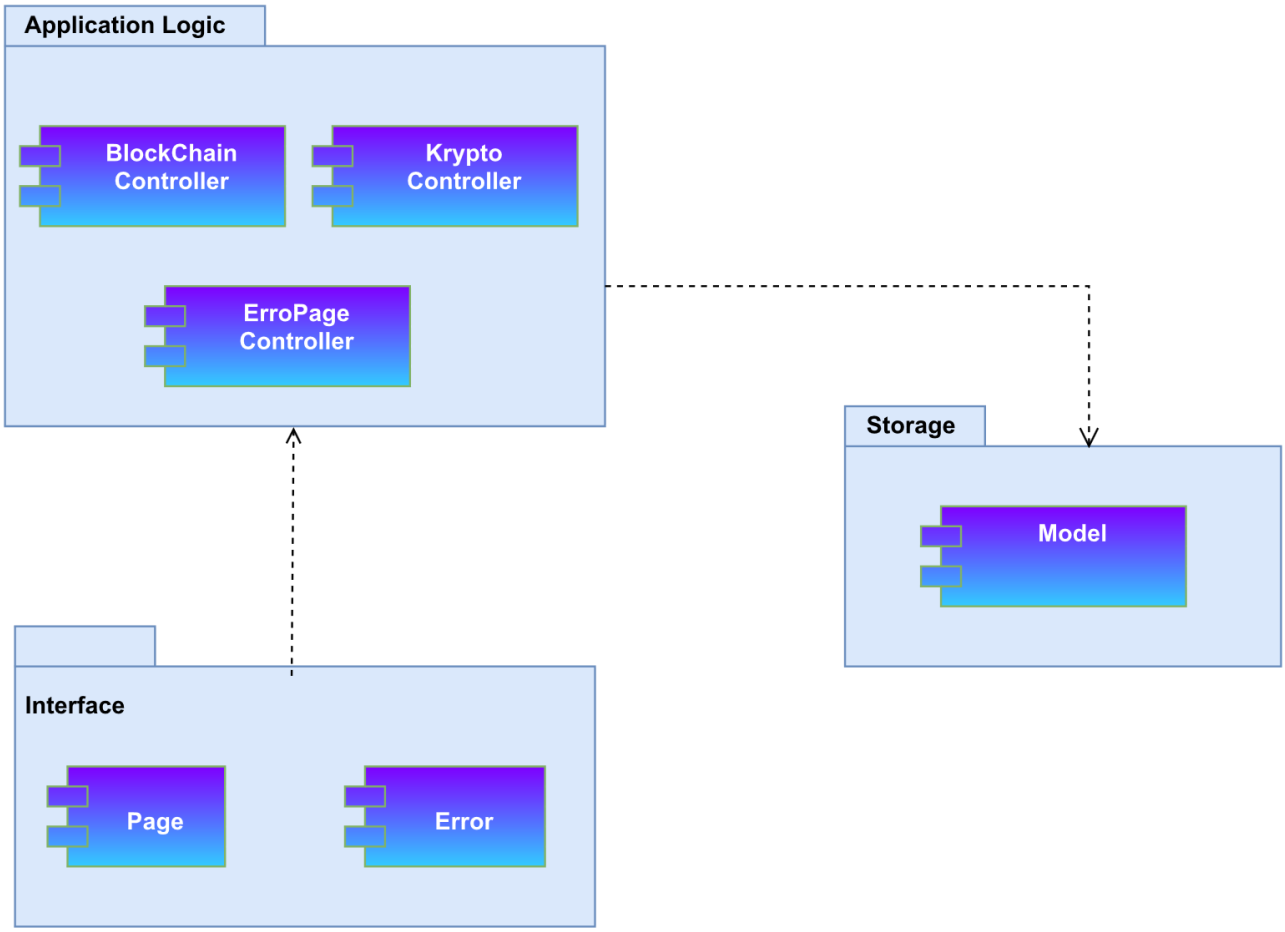
- **BlockchainController**: si occupa di tutte le operazioni di transizione che vengono effettuate sulla Blockchain, in particolare: il login, la registrazione, l'attivazione di un account, la disattivazione di un account, il cambio di privilegi per un account.
- **KryptoController**: si occupa di mostrare tutte le interfacce grafiche per un utente o un amministrare e di effettuare il logout per questi attori.
- **ErrorPageController**: si occupa di gestire gli errori http mostrando delle pagine di errore personalizzate.
- **Service**: contiene tutti i servizi dell'applicazione, in particolare permette di effettuare le transazioni alla Blockchain utilizzando lo smart contract "Authentication" convertito in linguaggio Java mediante Web3j.
- **Contracts**: permette l'interfacciamento con la Blockchain.

Lo **Storage** layer contiene le componenti:

- **Model**: modella lo stato di un Utente e di raccogliere i messaggi di errore o di successo.

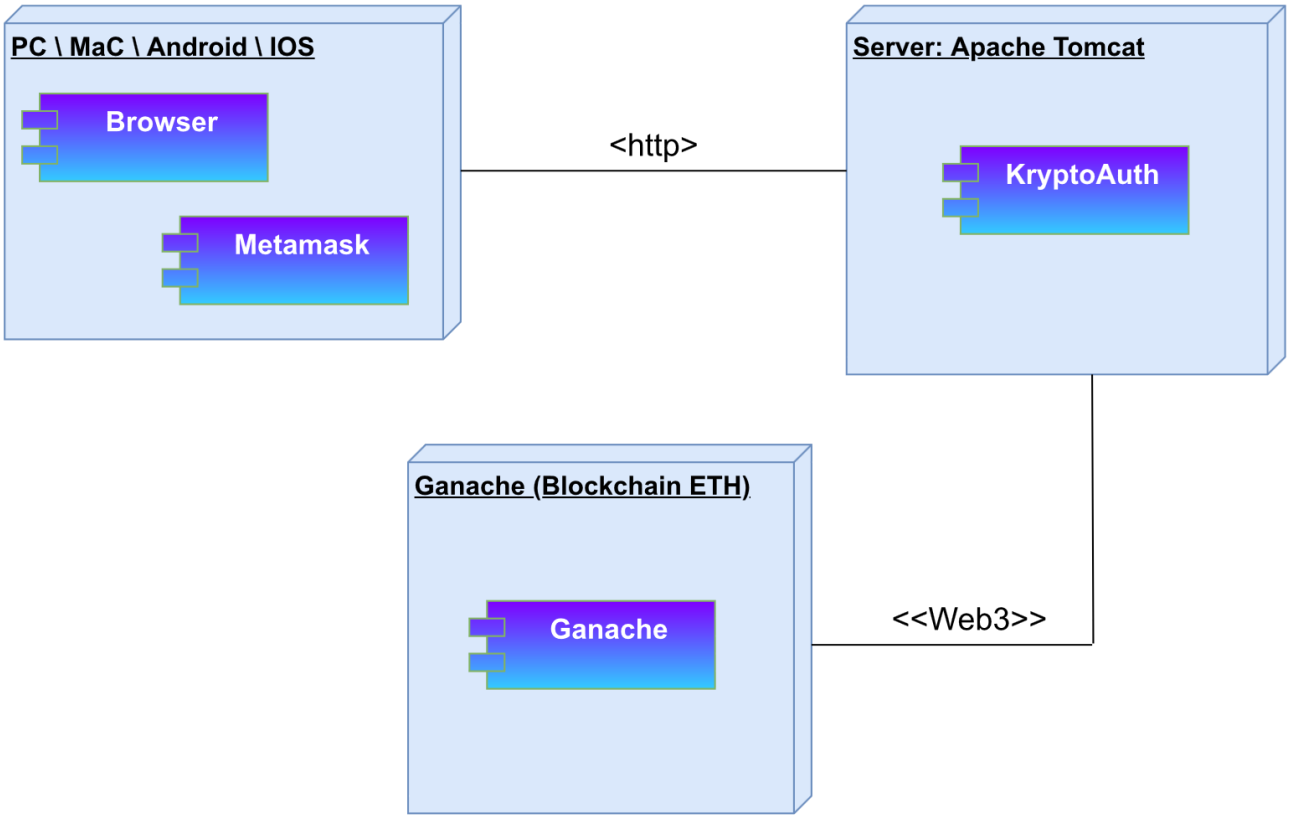
L'**Interface** layer contiene le componenti:

- **Error**: interfaccia grafica per le pagine di errore.
- **Page**: interfaccia grafica per l'utente e l'amministratore.



2.2 Mapping Hardware/Software

Il sistema KryptoAuth è realizzato come una Web DApp. Per poter interagire con il sistema è necessario accedervi tramite un qualsiasi browser da dispositivi quali PC, MAC, Android e IOS. Il Web Browser comunicherà con Web Server Apache Tomcat, integrato in Spring Boot, attraverso il protocollo HTTP che si occuperà pertanto di elaborare e rispondere a richieste del client. I dati invece vengono reperiti mediante Ganache: una Blockchain di test basata su Ethereum la quale interagisce con Metamask attraverso la creazione e la configurazione di una rete comune tra Ganache e Metamask.



2.3 Gestione dei dati mediante Blockchain

La gestione dei dati avviene mediante l'invio di transazioni alla Blockchain di test Ganache. La Web Dapp interagisce con essa dopo aver effettuato il deploy dello smart contract (in Solidity) sulla Blockchain mediante l'uso di "truffle migrate" da riga di comando. Inoltre, per poter interagire con i vari account generati randomicamente da Ganache è necessario utilizzare l'estensione Metamask disponibile nel marketplace dei vari browser.

Ganache e Metamask interagiscono tra di loro soltanto dopo aver creato e configurato una nuova rete in Metamask reperendo le informazioni da Ganache. Una volta effettuato le opportune configurazioni, la Web DApp potrà effettuare le varie operazioni definite nello smart contract (il quale è stato trasformato in una classe Java mediante l'API Web3j) utilizzando uno dei vari account registrati su Metamask (il collegamento di Metamask alla Web DApp KryptoAuth e la selezione dell'account da cui partirà la transizione è permessa mediante la libreria Web3js).

2.4 Controllo degli accessi e sicurezza

La sicurezza per l'accesso ai servizi forniti dal sistema è garantita tramite l'autenticazione con credenziali personali e il ruolo assunto da ciascun account. Ciascun utente dovrà dapprima registrarsi al sito, specificando il ruolo che vorrà avere, e soltanto dopo l'attivazione da parte di un amministratore potrà usufruire dei vari servizi offerti. Tale livello di sicurezza è garantito grazie all'utilizzo dello standard OpenZeppelin integrato nello smart contract scritto in Solidity.
Le operazioni di assegnazione o revoca dei ruoli sono effettuate solo dall'account che possiede come ruolo "DEFAULT_ADMIN_ROLE" all'interno dello smart contract. Di conseguenza, tutti gli altri utenti sono impossibilitati nel modificare o nell'accedere a quei servizi che potranno essere eseguiti dall'amministratore. Gli account, invece, che hanno come ruolo "USER_ROLE" potranno effettuare solamente il login al sito oltre che a navigare tra le varie pagine ma non potranno mai accedere alla sezione di gestione account riservata agli amministratori.

3. Glossario dei Sottosistemi

In questa sezione descriveremo i termini tecnici che sono stati utilizzati all'interno del Documento stesso.

- **API**: Application Program Interface, acronimo che indentifica delle librerie che forniscono funzioni implementate.
- **Solidity**: linguaggio di alto livello orientato agli oggetti per l'implementazione di smart contracts.
- **Web3j**: libreria Java e Android altamente modulare, reattiva e sicura per lavorare con Smart Contract e integrare con i client (nodi) sulla rete Ethereum.
- **Web3js**: è una raccolta di librerie che consentono di interagire con un nodo Ethereum locale o remoto utilizzando HTTP, IPC o WebSocket.
- **Ganache**: Blockchain di test basata su Ethereum.
- **Ethereum**: piattaforma decentralizzata del Web 3.0 per la creazione e pubblicazione peer-to-peer di contratti intelligenti creati in un linguaggio di programmazione Turing-completo.
- **Metamask**: estensione browser per l'emulazione di un wallet (portafoglio) cifrato e un gateway per le app Blockchain.

- **Truffle**: ambiente di sviluppo per lavorare con gli smart contract.
- **OpenZeppelin**: standard per le applicazioni Blockchain sicure; fornisce prodotti di sicurezza per creare, automatizzare e gestire applicazioni decentralizzate.