ALMA MATER STUDIORUM - UNIVERSITA DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria - DISI
Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA

in

LABORATORIO DI AMMINISTRAZIONE DI SISTEMI

Integrazione di sistemi di autenticazione e autorizzazione basati su blockchain in applicazioni Android

CANDIDATO

Bartolotti Luca
N° matricola: 0000692202

RELATORE

Chiar.mo Prof. Marco Prandini

CORRELATORE

Dott. Andrea Melis

Anno Accademico 2017/18

Sessione I

- “I am not looking for trouble!”

- “What a horrible way to live”

-Captain Jack Sparrow

Abstract

Questo progetto di tesi è incentrato sullo studio e l'implementazione di un sistema di autenticazione e gestione di autorizzazioni basato su *blockchain*. Questo sistema viene realizzato con riferimento ad una particolare architettura (MyData), e costituisce oggetto di studio per essere rimodellato come modulo generico riutilizzabile in diverse situazioni. Il progetto sviluppato consente l'autenticazione con un Account registrato su una *blockchain* e di inviare transizioni alla stessa. In questo modo vengono confermate e comunicate con la rete le autorizzazioni per l'utilizzo dei dati personali all'interno di un'architettura MyData.

Il lavoro presentato è un'estensione del progetto sviluppato dalla collega Valentina Protti nel quale è stata inserita l'autenticazione e l'invio di transazioni a blockchain tramite il progetto Uport.

All'interno del progetto di tesi viene presentato il processo analitico e realizzativo della parte software per l'autenticazione e l'invio delle transizioni tramite *blockchain*, e la successiva integrazione con il progetto citato sopra.

Infine, è presente un'analisi sulle possibili evoluzioni e i possibili sviluppi futuri.

Indice

Indice delle figure

1. Capitolo primo

Introduzione

Nella società odierna i dati personali hanno sempre più valore sociale, economico e pratico. Attualmente gli individui hanno poco controllo sui propri dati personali e come questi vengono usati dalle società e dai governi. I dati personali attualmente sono materiale grezzo e non ottimizzato dai nuovi servizi a causa della mancanza di interoperabilità e portabilità tra set di dati tra i vari servizi e settori. Si necessita quindi di un nuovo approccio a livello infrastrutturale su come gestire i dati personali e le autorizzazioni ad esso connesse. Notati i recenti problemi di privacy riscontrati dalla crescita dell'analisi di Big Data, molti dei potenziali e innovativi usi degli stessi potrebbero diventare impossibili se gli individui li percepissero come invasivi, oscuri o inaccettabili.

Per far fronte a questo scenario è nato MyData.

1.1 MyData e Mobility as a Service

MyData è un modello nato in Finlandia, dal progetto di un team di sviluppatori dell'Università di Aalto e commissionato dal Ministero dei Trasporti e delle Comunicazioni Finlandese. Il termine MyData si riferisce a un nuovo approccio, un mutamento di paradigma nella gestione e nel processamento dei dati personali che punta a trasformare l'attuale organizzazione a sistema centrale in un sistema centrato sulla persona, attraverso la visione dei dati personali come risorse alle quali l'individuo può accedere e controllare. Il modello mira a fornire gli individui di strumenti pratici per accedere, ottenere e usare set di dati contenenti le loro informazioni personali, come fatture, referti medici, informazioni finanziarie, e incentivare le organizzazioni che posseggono i dati personali a restituire all'utente il

controllo su di essi. Dando all'individuo il potere di scegliere come i propri dati possano essere usati, l'approccio di MyData permette di collezionare e usare i dati personali in modo da massimizzare i benefici ottenuti minimizzando la perdita di privacy.

Per quanto concerne l'uso etico e l'analisi delle informazioni personali, MyData e Big Data non sono mutualmente esclusive ma complementari. Il concetto di Big Data enfatizza le potenzialità di combinare e analizzare una grande quantità di dati e MyData si focalizza invece sull'abilità dell'individuo di controllare e beneficiare del valore dei suoi dati personali.

La prima applicazione del modello MyData la troviamo all'interno del concetto di *MaaS*, *Mobility as a Service*, un modello che raccoglie in una sola interfaccia tutte le necessità degli utenti per la mobilità quotidiana cittadina con l'obiettivo secondario di sostituire l'uso dell'auto proprietaria in favore della mobilità condivisa. Ne sono soggetti il trasporto pubblico urbano, biciclette, sistemi di *car sharing* e taxi. Data la quantità di dati personali in gioco e la tendenza da parte dei fornitori dei servizi di richiederne sempre in maggior numero, il Ministero finlandese ha cercato un sistema che garantisse la privacy dell'individuo: MyData.

1.2 Struttura della tesi

In questa sezione vengono sinteticamente illustrati i passaggi seguiti nello svolgimento del lavoro.

Il secondo capitolo viene spiega il contesto concettuale di inserimento del lavoro, presentando le architetture e i progetti su cui si basa.

Nel terzo capitolo si presenta lo studio eseguito sugli strumenti a disposizione e quindi la scelta di quelli ritenuti ottimali ai fini dell'elaborato.

Il quarto capitolo espone il lavoro software eseguito, cioè l'inserimento all'interno del progetto esistente delle funzionalità ricercate.

Il quinto e ultimo capitolo dell'elaborato espone un'indagine sui limiti del progetto, sui futuri sviluppi e le conclusioni sul lavoro svolto.

2 Capitolo secondo

Contesto di studio

2.1 MyData

MyData fornisce degli strumenti pratici per implementare la protezione dei dati e la privacy nel corso delle analisi di *Big Data* e rende trasparente il modo in cui i dati dell'individuo sono collezionati e processati.

In breve, l'architettura MyData è basata su quattro componenti principali: l'utente finale, o *Account-Owner*, l'operatore MyData, detto *Operator*, i servizi a cui l'utente si registra, *Service*, e il *Personal Data Storage*. *L'Operator* gestisce gli utenti, i servizi e le interazioni fra di essi. I servizi registrati presso l'utente vengono salvati nel *Service Registry*, all'interno *dell'Operator*. Ogni nuovo *Service* deve essere confermato dall'utente e verificato da procedure automatiche prima di poter essere inserito nel *Service Registry*.

Da normativa GDPR[1], tutte le operazioni svolte sui dati personali dell'individuo necessitano dell'approvazione da parte dello stesso tramite un consenso, detto *Consent*, e definisce a quali dati il servizio può accedere e in che modo. L'utente può in qualsiasi momento ritirare il consenso già confermato e un *Consent* ha tre stati possibili per permetterlo: *Active*, *Disabled*, *Withdrawn*. L'accesso ai dati è consentito solo in caso lo stato sia *Active*, mentre è bloccato per gli altri due stati. In caso il

[1] *General Data Protection Regulation*

consenso sia *Disabled* è possibile cambiare lo stato a seguito della conferma dell'utente, mentre nel caso *Withdrawn* il servizio deve essere registrato nuovamente per essere utilizzato.

Il *Personal Data Storage*, o *Personal Data Vault*, è un database contenente tutti i dati di un determinato *Account-Owner* ed è mantenuto all'interno del dispositivo mobile. L'accesso ai dati salvati all'interno del *Personal Data Vault* sono accessibili solo in presenza di un *Consent* adeguato.

Per tutelare meglio la privacy dell'utente finale, abbiamo trovato Uport: un sistema di identità aperto per il web decentralizzato.

2.2 Uport: Identità e Blockchain

Uport è un sistema di identità costruito sulla *blockchain* di *Ethereum* che assicura la sovranità dell'utente sulla propria identità. Questo sistema si basa su tre componenti principali: *Ethereum smart contracts*, *developer libraries* e una applicazione mobile.

2.2.1 Developer Libraries

Grazie alle *developer libraries*, gli sviluppatori di applicazioni di terze parti possono integrare in esse il protocollo *Uport*. Dovendo implementare le funzionalità di Uport nel servizio modulare di gestione dei permessi di MyDataModule[2], e dato che gli strumenti principali forniti da *Uport* sono librerie in JavaScript, ci si è posti davanti a tre strade. La prima era creare script in *JavaScript* con le funzionalità di Uport al loro interno ed eseguirli a necessità; la libreria richiesta per permetterlo avrebbe impedito la retrocompatibilità della applicazione coi dispositivi con Android inferiore a 8.0 Oreo.

La seconda strada era trasformare l'interfaccia grafica dell'applicazione Android MyDataModule, da *Activities* a *WebViews*, e quindi ricreare l'interfaccia di gestione dei

[2] Per ulteriori informazioni si veda la Tesi di Laurea di Valentina Protti.

servizi come insieme di pagine HTML dove sarebbero state utilizzate le funzioni *Uport*.

L'ultima strada era usare il *Software Developer* Kit, o SDK, scritto in Kotlin[3], sebbene le funzionalità limitate e la mancanza di documentazione ne avrebbero reso complicato lo studio e l'utilizzo.

Nonostante il *Software Developer Kit* per applicazioni mobili Android sia ancora in fase di sviluppo, era l'unico strumento che non avrebbe comportato lo sconvolgimento dell'applicazione MyDataModule e si è quindi optato per questo metodo.

2.2.2 Ethereum e Smart Contract

Ethereum è un'architettura a *blockchain* con un database degli stati associato, capace di immagazzinare programmi e il loro stato. Questi programmi sono comunemente chiamati *Smart Contracts*, possono essere depositati da un qualsiasi utente *Ethereum* e hanno un'interfaccia *function-based*.

Una volta depositato, lo *smart contract* può essere riferito tramite il suo indirizzo, che è un identificatore univoco. Un utente può eseguire uno *smart contract* inviando una transazione con l'indirizzo del *contract* alla destinazione. Quando un contratto viene eseguito, ne viene aggiornato lo stato.

Gli *Ethereum smart contracts* formano il nucleo dell'identità e contengono la logica che permette all'utente di recuperarla in caso di perdita del dispositivo.

2.2.3 Applicazione mobile e Identità

La applicazione mobile Uport contiene le chiavi dell'utente tramite le quali conferma e gestisce la sua Identità, o *Identity*.

[3] Kotlin è un linguaggio di programmazione *general purpose*, multi-paradigma, *open source* sviluppato dall'azienda di software JetBrains. Si basa sulla *Java Virtual Machine* per cui è assolutamente compatibile con Java

Le Identità Uport possono essere di varia natura: individui, dispositivi, entità o istituzioni. Le *Identity* sono *self-sovereing*: sono quindi completamente possedute dal creatore e non fanno affidamento su una terza parte per la creazione o la validazione. Attraverso un'Identità Uport è possibile firmare digitalmente e verificare richieste, azioni o transazioni.

Un'Identità può essere collegata crittograficamente a dei dati salvati *off-chain*. Ogni Identity è in grado di salvare l'*hash*[4] o un attributo *data blob*, che sia questo su IPFS, Azure, AWS, Dropbox, etc., dove tutti i dati associati a quell'identità verranno conservati in modo sicuro.

Il nucleo di un'Identità Uport è un identificativo Uport, un numero di 20 byte rappresentato per mezzo di una stringa esadecimale che agisce come identificatore unico, persistente e globale. Questa stringa è definita come l'indirizzo di uno *smart contract* di *Ethereum*, conosciuto come *Proxy contract*. Il *Proxy contract* può trasmettere transazioni ed è grazie a questo meccanismo che l'identità interagisce con altri *smart contracts* nella blockchain *Ethereum*.

I componenti delle Identità sono:

- Un identificatore sotto forma di un MNID (*Multi Network Identifier*),
- Una chiave di firma,
- Una chiave pubblica salvata nel *Uport Registry.*

Grazie ad essi l'Identità è in grado di:

- Firmare JWTs (*JSON Web Tokens*),
    - Autenticare sé stessa a una terza parte,
    - rivelare alcune delle informazioni private di sé stessa,
- Ricevere richieste per rivelarsi,
- Ricevere e salvare verificazione di terze parte firmate su sé stessa,
- Firmare transazioni *Ethereum.*

[4] Elemento definito a pag. 13

In generale, le identità sono create e gestite tramite la app mobile e sempre attraverso questa vengono firmate le transazioni, come illustrato in Figura 1.

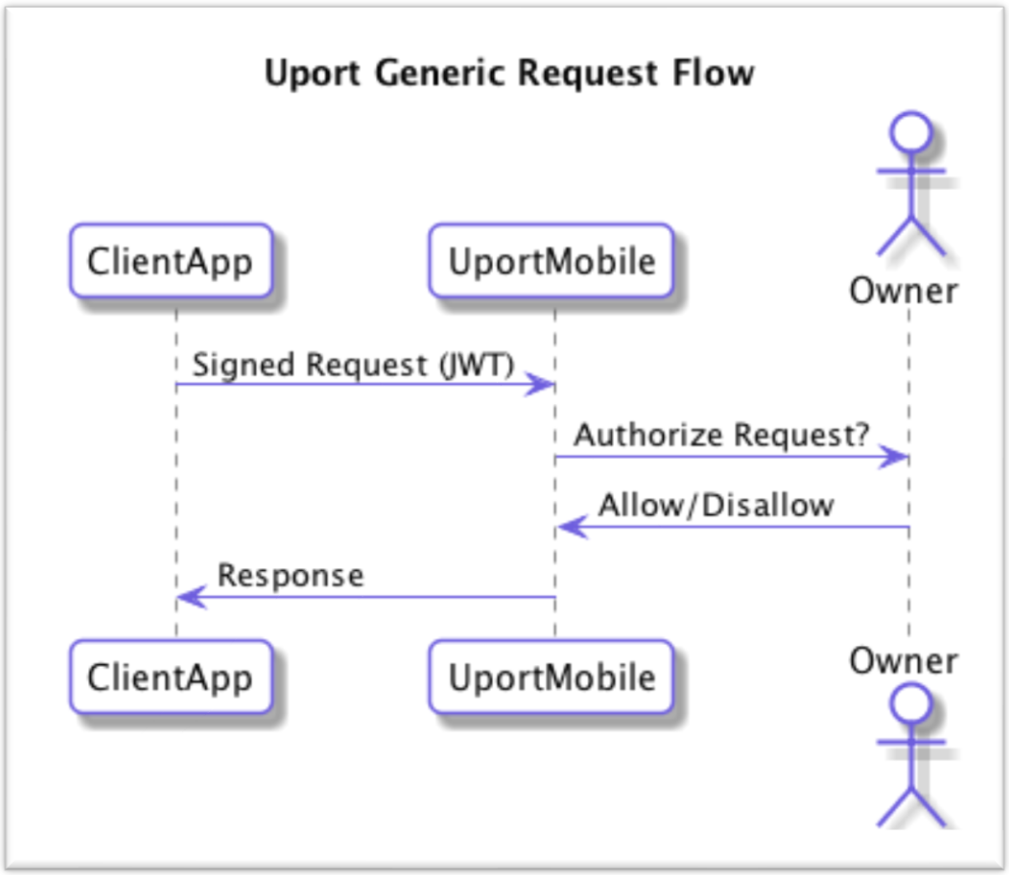

Figura 1 - Flusso genico di una richiesta Uport

In alternativa, le transazioni possono essere firmate direttamente tramite *token* e una coppia di chiavi, una pubblica e una privata, senza la necessità della conferma fisica da parte dell'utente.

2.3 Blockchain

La *blockchain* è una lista di *records* in continua crescita, chiamati *blocks* ed ognuno di essi è collegato al precedente e reso sicuro tramite crittografia. I principi della *blockchain* sono: decentralizzazione, trasparenza, sicurezza e immutabilità.

Ogni *block* della *chain* contiene un riferimento, *hash*, al *block* precedente, un riferimento temporale, *timestamp*, e dei dati formati da un insieme di transizioni validate. Ogni *block* possiede quindi il proprio identificativo tramite il quale può essere riferito.

Grazie alla *blockchain* si possono registrare le transazioni tra due parti in modo sicuro, verificabile e permanente. Perché tutto questo sia garantito, viene sfruttata una rete *peer-to-peer* che si collega ad un protocollo per la convalida dei nuovi *blocks*.

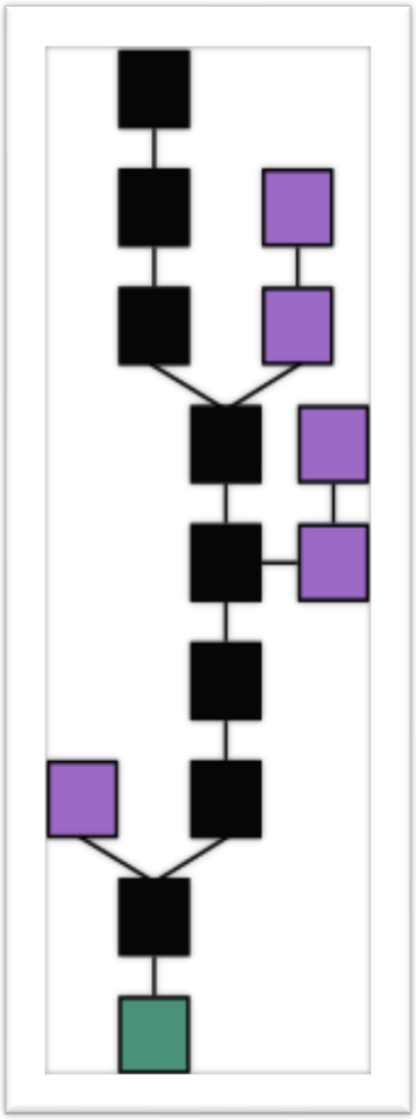

Figura 2- Schematizzazione di una blockchain

Una volta che un *block* è stato registrato, non può più essere modificato retroattivamente se non modificando tutti i *blocks* successivi, cosa che richiederebbe il consenso della maggioranza della rete. Grazie alle *blockchains*, non c'è più bisogno di un'autorità centrale e fidata che verifichi il comportamento degli utenti, dal momento che l'autenticazione avviene tramite la collaborazione di massa, spinta da interessi collettivi.

Si ottiene quindi un flusso di lavoro robusto dove la competenza dei partecipanti in materia di sicurezza dei dati non è necessaria. L'uso di una *blockchain* rimuove i

caratteristici problemi dell'infinita riproducibilità di un bene digitale e conferma che ogni dato venga trasferito solo una volta. Inoltre, a *blockchain* risulta più veloce, sicura ed economica rispetto ai sistemi tradizionali.

Il database *blockchain* è formato da *blocks* e transazioni. Le transazioni sono salvate tramite il loro *hash* sotto forma di *Merkle tree* [5] in un *block* e ognuno di questi contiene *l'hash* crittografato del *block* precedente, collegandoli così in una catena fino al *block* originale di genesi, in verde in Figura 2. Può capitare che due *blocks* siano generati concorrentemente, dando origine così una *fork* temporanea, dato che entrambi hanno *l'hash* di riferimento dello stesso *block*, come può essere notato in Figura 2.

Per garantire una sicura cronologia *hash-based*, ogni *blockchain* ha uno specifico algoritmo per valutare le differenti versioni storiche e assegnargli un punteggio, così che la versione con il punteggio maggiore possa essere selezionata al posto delle altre. I *blocks* che non vengono selezionati per l'inclusione nella *chain* vengono chiamati orfani e sono i *block* evidenziati in viola nella Figura 2.

I *peer* possono avere versioni differenti della storia della *chain* e tengono solo la versione con la migliore valutazione a loro conosciuta. Quando un *peer* riceve una versione con un punteggio migliore della propria, solitamente la vecchia versione con un singolo nuovo *block* aggiunto, estende o sovrascrive il suo database e ritrasmette i miglioramenti ai suoi *peers*. Non c'è garanzia che una particolare versione rimanga la migliore per sempre.

Siccome le *blockchain* sono costruite per aggiungere il punteggio di valutazione dei nuovi *blocks* ai vecchi *blocks*, dati gli incentivi a lavorare solo per estendere con nuovi *blocks* invece di sovrascrivere i vecchi, la probabilità che una *entry* sia sostituita si abbassa esponenzialmente col numero di *blocks* costruiti dopo di essa.

[5] Un Merkle tree, o hash tree, è un albero di hash, ovvero un albero in cui i nodi sono gli hash dei blocchi di dati e non i dati stessi.

Il tempo medio che trascorre fra la generazione di un blocco e l'altro è detto *block time*. Questo intervallo di tempo varia in base alla *blockchain*, ad esempio per *Ethereum* il *block time* è di circa quindici secondi mentre per *Bitcoin* è dieci minuti.

Salvando i dati nella rete *peer-to-peer*, la *blockchain* elimina diversi rischi che comporterebbe un sistema centralizzato. Le *blockchains peer-to-peer* sfruttano il passaggio di messaggi *ad hoc* e un *networking* distribuito e in questo modo non hanno le vulnerabilità tipiche dei sistemi centrali che i *cracker* potrebbero usare. Parte della sicurezza delle *chains* è garantita dal sistema di crittografia a chiavi pubbliche, che rappresentano indirizzi sulla *blockchain*. I *token* inviati nella rete vengono registrati come appartenenti ad una chiave pubblica. La chiave privata, invece, è come una password che permette al suo proprietario di accedere alle proprie risorse digitali, oppure, di interagire con le varie funzionalità fornite dalla *blockchain*.

Ogni nodo del sistema decentralizzato ha una copia completa della *blockchain* e la qualità dei dati è mantenuta da una massiva replicazione del database e dalla *computational trust*. Non esiste, infatti, una copia centrale ufficiale e nessun utente è più affidabile rispetto gli altri: per questo i dati sulla *blockchain* sono considerati incorruttibili.

Le transazioni sono mandate in broadcast sul network della *blockchain* e i messaggi sono consegnati sulla base del *best-effort*. I nodi "minanti" (*mining*) validano le transizioni e le aggiungono al *block* che stanno costruendo, quindi inviano in broadcast il *block* completato agli altri nodi.

Il vantaggio presentato da una rete a *blockchain* aperta, senza permessi, o pubblica, è che non è richiesto proteggersi da attori malevoli e non è necessario nessun controllo sull'accesso. Questo significa che le applicazioni possono essere aggiunte alla rete senza l'approvazione o la fiducia di altri, usando la *blockchain* come un livello di trasporto. *Bitcoin* e le altre *blockchains* di criptovalute assicurano la loro rete richiedendo ai nuovi iscritti una "prova di lavoro" (*proof of work*), mentre quelle private usano invece un controllo di accesso per gestire chi accede alla rete. Opposte

alle *blokchain* pubbliche, i validatori nelle *chains* private sono controllati dal possessore della rete.

Quindi la *blockchain* è composta da:

- **Nodo**: sono i partecipanti alla *blockchain* e sono costituiti dai server dei vari partecipanti;
- **Transazione**: sono i dati che vengono inviati da un *endpoint* ad un altro, sono i valori oggetti di scambio e devono essere verificate, approvate e poi archiviate;
- ***Block***: contiene al suo interno un insieme di transazioni che sono unite per essere verificate, approvate e poi archiviate dai partecipanti alla *blockchain*;
- ***Ledger***: è il registro pubblico nel quale vengono salvate con trasparenza e in modo immutabile tutte le transazioni effettuate ed è costituito dai *blocks* incatenati fra loro tramite una funzione crittografica e all'uso di *hash*;
- ***Hash***: è un'operazione non invertibile che mappa una stringa alfanumerica di lunghezza variabile in una stringa unica e univoca di lunghezza determinata. Questo permette di avere un identificativo univoco per ogni *block*.

2.4 Android, GoBoBus e MyDataModule

Android è un sistema operativo *mobile* sviluppato da Google, basato su una versione modificata del kernel Linux e progettato principalmente per i dispositivi mobili *touchscreen* come smartphone e tablet.

GoBoBus è un'applicazione Android, sviluppata da Stefano Gasperini e Francesco Fiacco, per facilitare l'uso dei mezzi pubblici alle persone ipovedenti.

Per la sua implementazione fu scelta la piattaforma Android per diverse ragioni, tra le quali:

- È molto diffusa, infatti ad aprile 2017 contava oltre due miliardi di utenti mensili attivi, circa il 63% del totale;
- Le applicazioni Android sono sviluppate in Java e il kit di sviluppo Android, il Software Developer Kit, è fornito di librerie, documentazione, *debugger* ed un emulatore;
- Espone un servizio *built-in* di accessibilità, *TalkBack*, molto utile agli utenti con particolari necessità. *TalkBack* fornisce un aiuto agli individui ipovedenti per l'interazione col dispositivo *mobile* tramite l'emissione di un *feedback* sonoro e fisico, sotto forma di vibrazione, quando si interagisce con un elemento dello schermo;
- Offre la possibilità di dettare a voce ciò che si desidera scrivere grazie a una funzionalità integrata nel sistema operativo.

GoBoBus utilizza diversi componenti Android e tecnologie tra cui *Activity*, *ServiceIntent*, *TextToSpeech* e *SharedPreferences*. I file di *Layout* e *Manifest* sono scritti in XML, un linguaggio a marcatori i cui obiettivi originari erano la formattazione dei documenti e il salvataggio di informazioni strutturate che potessero essere sia comprensibili dalla macchina che *human-readable*. Attualmente XML viene impiegato soprattutto per rappresentare strutture dati e per descrivere sia i *layout* che le specifiche dell'applicazione, come nel nostro caso.

MyDataModule è un modulo applicativo per la gestione dei servizi MyData all'interno dell'app GoBoBus, sviluppata da Valentina Protti nella tesi "Progetto e realizzazione di componenti per la protezione dei dati personali su piattaforma Android". È questo il progetto in cui si inserirà l'autenticazione e la gestione delle autorizzazioni tramite blockchain usando un'Identità Uport.

3 Capitolo terzo

Analisi

3.1 Scelta degli strumenti

Per quanto riguarda l'ambiente di sviluppo Android sono utilizzati principalmente due software: *Eclipse* e *Android Studio*, per il quale si è optato. *Android Studio* è l'ambiente ufficiale sviluppato da Google e JetBrains, basato su IntelliJ IDEA. Questo ambiente fornisce tutti gli strumenti necessari per sviluppare una applicazione Android scritta in Java o Kotlin, tra cui: un sistema flessibile di produzione (*build*) basata su Gradle, un emulatore veloce e ricco di funzionalità e l'integrazione del sistema GitHub.

Per integrare le funzioni di Uport in MyDataModule sono disponibili diversi strumenti sulla pagina GitHub Uport, la maggioranza di questi in JavaScript. Il principale e più completo è *uport-connect*, una libreria per *front end developer* per lo sviluppo di applicazioni lato *client*. Grazie a questa libreria, è possibile far interagire la applicazione in oggetto con la applicazione mobile di Uport installata nel dispositivo.

Essendo la libreria in JavaScript, i possibili metodi per integrarla con MyDataModule scritta in Java sono:

1) Creare tutti gli Script necessari in file separati ed eseguirli quando necessario. Questo procedimento presenta però diversi problemi, tra cui il principale è la retrocompatibilità. Infatti, la libreria *javax* che permette di chiamare funzioni JavaScript all'interno di codice Android, necessita che il dispositivo abbia almeno Android 8.0 con la versione 26 del *Software Developer Kit* per funzionare, perdendo così un numero alto di dispositivi.

2) Sostituire le *Activity* della applicazione con delle *WebView* e ricreare l'interfaccia grafica del modulo MyDataModule come insieme di pagine HTML, come fosse un sito web. Questa opzione comporterebbe la riscrittura di buona parte della app MyDataModule e rende insicura la compatibilità con GoBoBus.

La soluzione ritenuta migliore è stata utilizzare l'SDK di Uport per app mobili Android, anche se ancora in fase di sviluppo. Pur essendo uno strumento minimale e acerbo, per i fini di questo progetto era sufficiente ciò che poteva offrire. Questo kit di sviluppo ha permesso infatti di creare una nuova identità Uport di dispositivo oltre ad inviare, firmare e confermare le transizioni. Lo strumento è scritto in Kotlin ed è risultato integrabile con il modulo MyDataModule senza stravolgerne la struttura, risultando anche discretamente retrocompatibile. Infatti, il valore minimo di API Android viene posto a 23, cioè Android Marshmallow 6.0 .

3.2 Creazione dell'identità Uport

Nella maggioranza dei casi l'identità Uport viene creata e gestita tramite la applicazione mobile Uport. Il processo di creazione è semplice e veloce, infatti una volta installata la applicazione è sufficiente cliccare su "Create Identity", accettare i termini e le condizioni di utilizzo, verificare il proprio dispositivo tramite un codice inviato per SMS ad un numero di telefono o un *captcha*, inserire il proprio nome e una foto, questi ultimi dati vengono salvati soltanto nel dispositivo, e l'identità è creata, registrata nella blockchain e la applicazione pronta per confermare le transazioni.

Figura 3 - QR code mostrato alla creazione

Figura 4 - Card della transazione tramite la quale è stata creata la app MyPort

Grazie a questa identità, uno sviluppatore può creare le app sulla blockchain dalla sezione *App manager* del sito Uport. Selezionando "*Create An App*" viene richiesta l'autorizzazione all'utente per creare una nuova applicazione, sotto forma di *smart* contract nella *blockchain*, tramite una Transizione. Se l'operazione è eseguita da PC, compare un QR code, Figura 3, da scansionare tramite la applicazione mobile Uport e, successivamente, viene mostrata nella applicazione mobile la *card* della transazione, Figura 4, da confermare per verificare l'operazione. Nel caso l'operazione sia eseguita usando il dispositivo mobile, cliccando su "*Create an App*" ci verrà mostrata direttamente la *card* della transazione da confermare.

Una volta confermata la transizione, si viene reinderizzati dal browser alla pagina del nuovo *smart contract* creato, nella quale viene mostrato il suo indirizzo, è possibile cambiarne il nome e inserire un'immagine appropriata. In questo modo è stata creata la applicazione "*MyPort*" a cui verranno inviate le transizioni dalla applicazione sviluppata.

Inoltre, ogni volta che viene autorizzata una transazione, l'applicazione Uport ci offre la possibilità di prendere visione della transazione in dettaglio, come esposto nella Figura 5.

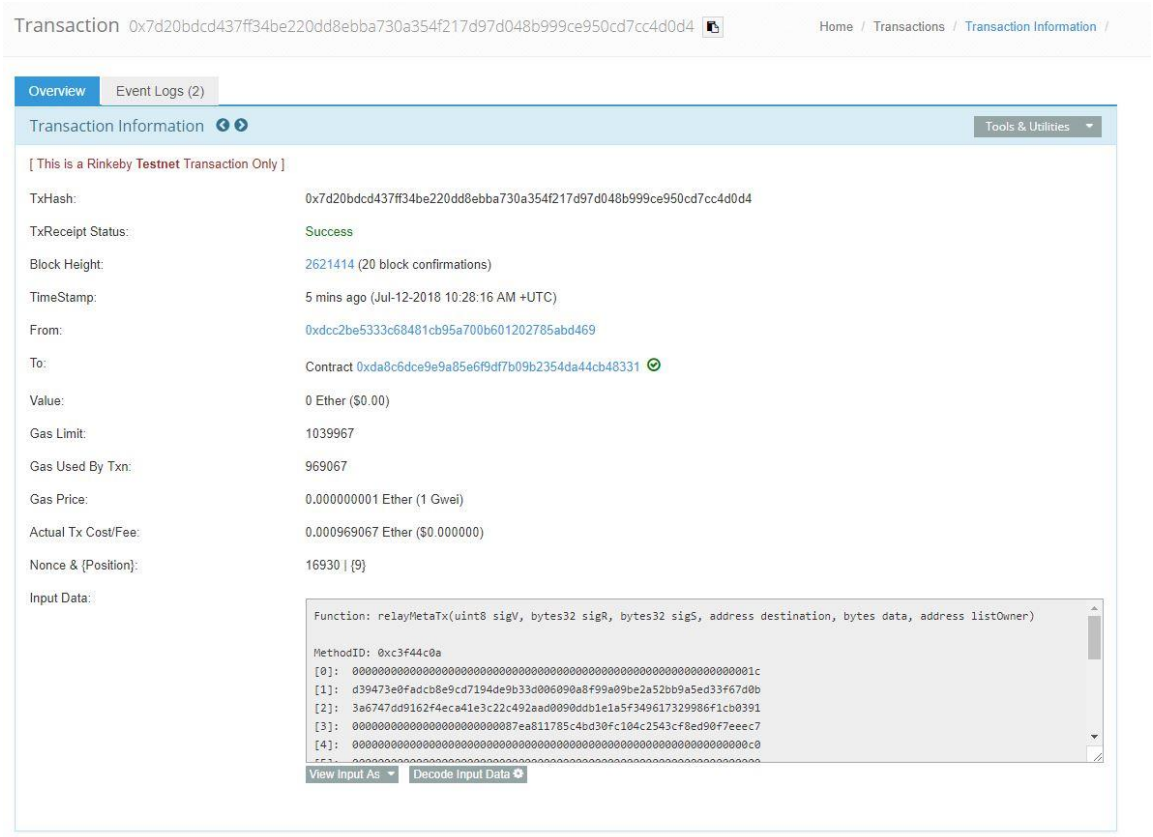

Figura 5 - Dettaglio della transazione eseguita per creare la app MyPort

Tuttavia, avendo deciso di utilizzare il *Software Developer Kit* l'utente finale non necessita di installare la applicazione Uport e di creare la propria identità, in quanto la creazione e la gestione dell'identità e delle transazioni viene eseguita via *software*. Dopo aver correttamente inizializzato l'oggetto Uport implementato nell'*SDK* è possibile creare un Account che non necessiti dell'interazione umana per l'autorizzazione delle transazioni.

Per creare un Account è necessario inizializzare un oggetto Configuration attraverso il suo costruttore e quindi inserire al suo interno il contesto dell'applicazione e il fornitore di token tramite i metodi:

- configuration.setApplicationContext(context: Context)

- configuration.setFuelTokenProvider(provider: IfuelTokenProvider)

Il provider di fuelToken è lo *smart contract* MyPort creato tramite il sito Uport, identificato tramite il suo indirizzo.

Dopodiché si inizializza l'oggetto Uport invocando il metodo:

- Uport.initialize (configuration: me.uport.sdk.Uport.Configuration)

È possibile quindi creare l'identità del dispositivo grazie a:

- Uport.createAccount (network: EthNetwork, completion: AccountCreatorCallback)

Dove EthNetwork è un oggetto che rappresenta una delle quattro reti di *Ethereum* (mainnet, ropsten, kovan, rinkeby) e la completition è la funzione chiamata una volta creato l'account, nel nostro caso una lamda function all'interno della quale troviamo l'istanza dell'account Uport rappresentante l'identità del dispositivo.

3.3 Invio delle transizioni e analisi della ricevuta

Grazie al metodo "*send*" fornito dall'account Uport è possibile inviare sia transazioni contenenti criptovaluta *Ethereum*, sia transazioni di dati. Sono proprio le seconde che verranno usate nell'applicazione comunicando alla *blockchain* le autorizzazioni dell'utente, sia per l'aggiunta di account presso nuovi servizi, sia per le variazioni di *Consent* presso di essi. Al metodo "*send*" vengono passati come argomenti il contesto dell'applicazione, l'indirizzo dello *Smart Contract* a cui verrà inviata la transizione, MyPort nel nostro caso, e i dati convertiti in forma di ByteArray. Si aspetta quindi che la transazione venga confermata invocando il metodo "*awaitConfirmation*", all'interno del quale è presente l'oggetto *receipt*, ovvero la ricevuta della transazione. In questo oggetto si possono visionare:

- il ***blockHash***, ovvero *l'hash* del *block* dove è stata registrata la transazione;

- il ***blockNumber***, il numero del *block* in cui è stata inserita la transizione;
- il ***transactionHash***, che è l'*hash* della transazione;
- il ***transactionNumber***, l'indice posizionale della transazione all'interno del *block*;
- il **contractAddress**, in caso la transizione sia di creazione di un contratto, questo sarebbe il suo indirizzo. Risulta "*null*" in caso contrario;
- lo ***status***, ovvero lo stato della transazione. 0x1 sta per successo, 0x0 per fallimento;
- ***comulativeGasUsed***, l'ammontare di Gas usato quando questa transazione è stata eseguita nel blocco;
- ***logs***, ovvero il *log* della transazione che ha creato questa ricevuta.

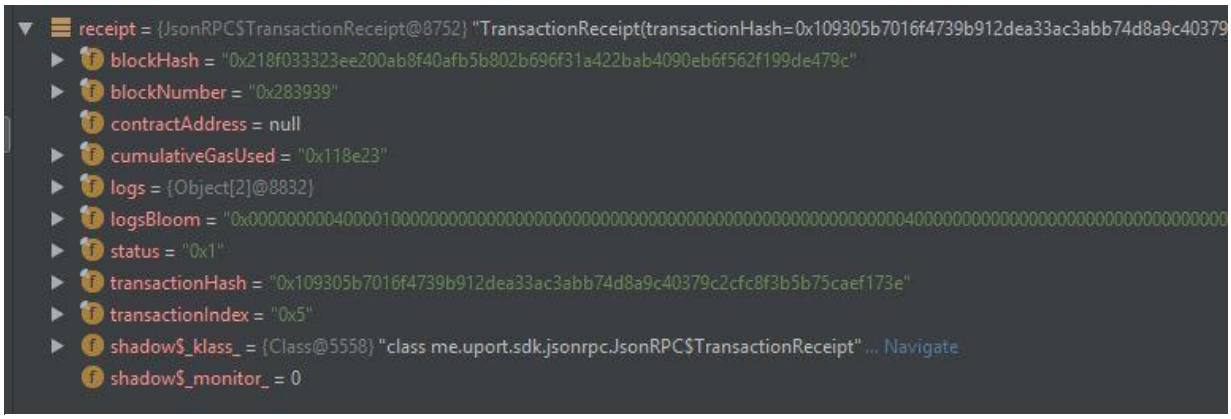

Figura 6 - Dettaglio della ricevuta di una transazione

4 Capitolo quarto

Implementazione

Come precedentemente enunciato l'obiettivo di questo elaborato è gestire le autorizzazioni concesse dall'utente all'applicazione per l'esecuzione di servizi e l'accesso a dati personali tramite invio di transizioni alla *blockchain*. Per realizzarlo è stata aggiunta la classe *UportData* al MyDataModule, è stato implementato il *serviceUport* e modificate le *Activity*.

4.1 UportData

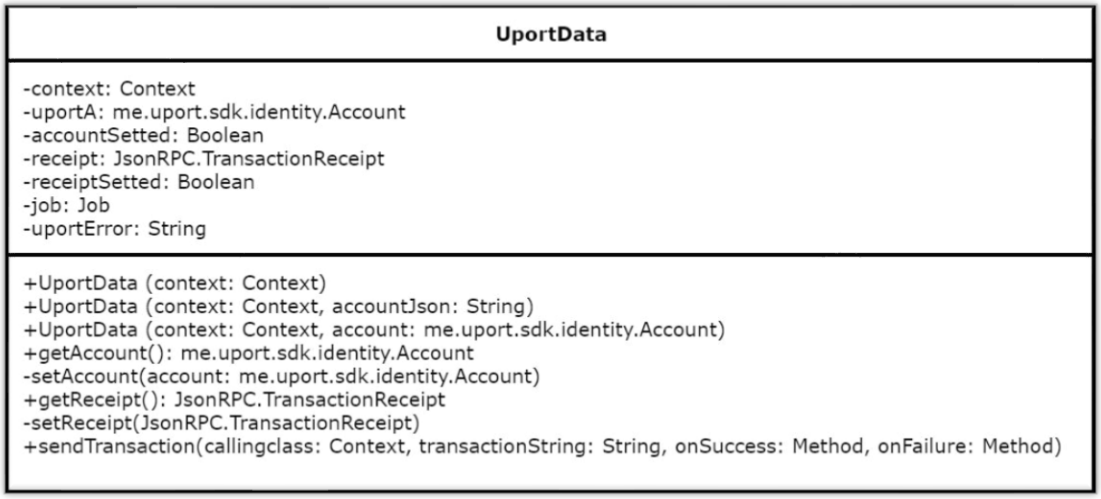

Figura 7 - Diagramma UML della classe UportData

La classe UportData, scritta in Kotlin per interfacciarsi meglio con l'SDK, si occupa di interagire con l'identità Uport, sotto forma di oggetto Account, grazie ai costruttori e inviare transizioni alla blockchain tramite il metodo "*sendTransaction*". Dopo aver notato che una transizione impiega circa un minuto per essere inviata e confermata, si

è optato per il passaggio di due metodi come argomento, "*onSuccess*" in caso di conferma della transazione e "*onFailure*" in caso contrario o di errori.

Questo metodo in futuro dovrà far aprire la applicazione mobile Uport e mostrare la carta della transizione, così che l'utente possa autorizzare l'operazione in corso. Nel costruttore primario UportData (context: Contex) viene verificato se l'utente possiede già un account Uport e, in tal caso, viene salvato nel campo uportA della classe UportData. In caso contrario viene creato un nuovo account inizializzando l'oggetto Uport e chiamando il metodo "*createAccount*" descritto nel *Software Developer Kit*.

Il secondo costruttore richiede in ingresso una String formata dal contenuto del file Json rappresentante un account Uport, lo trasforma in un oggetto Account tramite il metodo "*fromJson*", fornito nell'SDK, e chiama quindi il terzo costruttore, il quale chiede in ingresso un'istanza di Account che viene salvata nell'attributo uportA. Presenta poi *Getter* pubblici e *Setter* privati per i campi *uportA* e *receipt*, così da concedere l'accesso all'account Uport e alla ricevuta dell'ultima transazione eseguita e permettere agli altri metodi della classe di modificarli. I *Setter*, oltre ad assegnare l'oggetto passato all'attributo corrispondente, pongono anche a "*true*" il rispettivo *Boolean*, *accountSetted* o *receiptSetted*, che vengono poi usati per il controllo di errori.

4.2 ServiceUport e implementazione all'interno del Personal Data Vault

Si è quindi creata la classe del servizio relativo ad Uport, *ServiceUport*. Questa classe estende *AbstractService* e ne sovrascrive i metodi. Nel costruttore viene registrato il servizio all' interno dell'architettura MyData attraverso il *ServiceRegistry*. Il metodo *"provideService*", definito nella classe *AbstractService*, restituisce l'oggetto UportData che viene letto dalla *Personal Data Vault* all'interno di "*concreteService*".

Per inserire l'oggetto UportData all'interno del Personal Data Vault è necessario aggiungere il campo pubblico

String UPORTDATA_CONST = UportData.class.getCanonicalName()

nella classe Metadata. Deve poi essere inserito il campo privato UportData nella classe *PersonalDataVault* e implementati il metodo privato "*readAccount(): UportData*" all'interno del quale, se presente, si legge il file json rappresentante l'account Uport del dispositivo e si inizializza l'oggetto UportData tramite di esso. In caso di errore durante la lettura del file viene creato un nuovo account Uport per il dispositivo (tramite costruttore UportData) e quindi restituita l'istanza di UportData così ottenuta. Infine, bisogna aggiungere ai metodi "*getData*" e "*saveData*" il caso della lettura o scrittura dell'oggetto UportData dal o nel Personal Data Vault.

4.3 MainActivity

Nella classe *MainActivity* è stato inserito il campo statico *instance: MainActivity* e il relativo metodo statico "*getInstance(): Context*" che restituisce l'istanza di *MainActivity*. Questo risulta necessario per poter disporre ovunque nel codice di un *Context* da passare come argomento al costruttore della *Configuration* di Uport.

Nel metodo "*onCreate*", dopo il login dell'utente MyData, viene creato e aggiunto ad esso un account presso il servizio *ServiceUport*. Questo è l'unico caso di aggiunta di un servizio senza richiesta dell'autorizzazione dell'utente tramite una transizione con Uport, dato che per poter inviare le transazioni è necessario appunto questo servizio. Notato che la creazione dell'utente Uport richiede circa una decina di secondi è stato creato il metodo "*setEnterButtonClickable (Boolean bool)*" che abilita o disabilita il pulsante "ENTRA" in modo da impedire all'utente la continuazione senza che *serviceUport* sia operativo. Nel momento in cui viene creato l'account presso il servizio serviceUport il pulsante viene abilitato e, quando premuto, viene controllato se

l'utente possiede un account presso *serviceProva* con stato *Active*. In caso affermativo la app passa alla *UserProfileActivity*, altrimenti viene lanciata la *NewAccountActivity*

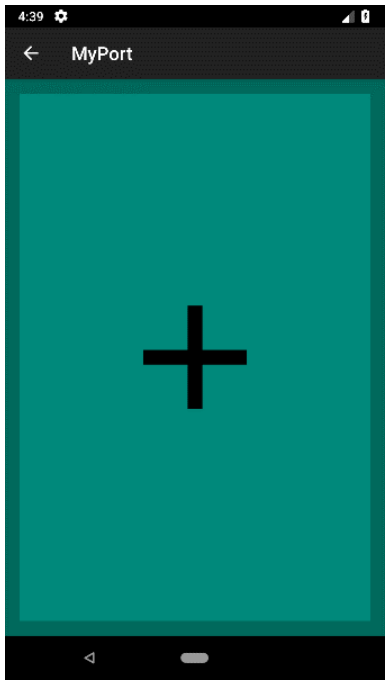

Figura 8 – NewAccountActivity Pulsante attivo

4.4 NewAccountActivity

Nella classe *NewAccountActivity* è stato modificato il metodo "*newAccount*" in modo che alla pressione del pulsante per l'aggiunta dell'account *serviceProva*, questo venga disabilitato, si veda la Figura 9, e venga inviata una transizione Uport tramite il metodo "*sendTransaction*" a cui sono passati come argomenti i metodi "*onConfirmedService*" e "*onFailedService*".

Nel caso la transizione venga confermata si procede col metodo "*onConfirmedService*" nel quale viene effettivamente aggiunto all'utente MyData un account relativo al servizio *serviceProva* e si viene poi indirizzati alla *UserProfileActivity*.

In caso la transizione non sia confermata o in caso di errore viene invocato il metodo "*onFailedService*" che semplicemente riabilita il pulsante per l'aggiunta dell'account presso il servizio *serviceProva*, in modo da poter riprovare.

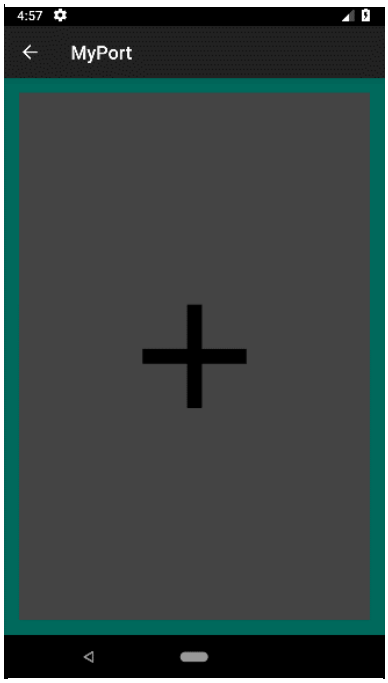

Figura 9 - NewAccountActivity Pulsante Disattivato

4.5 UserProfileActivity

Alla creazione di questa *Activity* viene salvata un'istanza di UportData come campo, ottenuta tramite il metodo *provideService*, e viene assegnata l'istanza dell'oggetto *UserProfileActivity* al campo *instance: UserProfileActivity*, necessario per il metodo "*sendTransaction*". Come visibile nella Figura 10, è stata inoltre aggiunta una casella di testo in cui vengono mostrati parte dei dati dell'account Uport per *debug*, in particolare l'indirizzo dell'account, il *network* su cui è registrato e il *token* utilizzato per la conferma delle transizioni.

Figura 10 - UserProfileActivity Pulsanti abilitati e Consent ACTIVE

4.5.1 changeLocationConsent

Il metodo "*changeLocationConsent*", che viene eseguito all'interazione con lo *switch* "Condividi Posizione", è stato modificato: all'esecuzione viene visualizzato un dialogo, visibile in Figura 11, in cui si chiede la conferma per cambiare il consenso della posizione e, in caso l'utente autorizzi l'operazione, viene invocata la funzione "*sendTransaction*" passando come argomenti i metodi "*onChangeConsentSuccess()*" e "*onChangeConsentFailure()*". In caso di conferma e verifica della transazione viene eseguito il metodo "*onChangeConsentSuccess()*", in caso contrario o di errori viene eseguito "*onChangeConsentFaiulure()*".

Figura 11 - UserProfileActivity Dialogo di richiesta conferma

4.5.1.1 onChangeConsentSuccess

In questo metodo viene effettivamente aggiornato il consenso relativo al dato posizione del *serviceProva*, consentendolo o negandolo a seconda della posizione dello *switch*.

4.5.1.2 onChangeConsentFaiulure

All'esecuzione di questo metodo, lo *switch* relativo al dato posizione viene riportato nella posizione corretta.

4.5.2 disableOption

Di seguito viene analizzata la disabilitazione del *Consent* del *serviceProva*, facendolo diventare *Disabled*. Il processo di abilitazione è identico tranne che, naturalmente, il Consent diventerà *Active*.

Alla pressione del pulsante "Disabilita Consenso" viene invocato il metodo "*disableOption*" che, dopo aver disattivato entrambi i bottoni "Revoca Consenso" e "Disabilita Consenso" come mostrato in Figura 12, mostra un dialogo in cui si chiede all'utente di confermare l'operazione, simile a quello mostrato in figura 11. Se l'utente autorizza la disabilitazione del servizio viene eseguito il metodo "*sendTransaction*" con i riferimenti ai metodi "*onDisableSuccess()*" e "*onDisableFailure()*" passati per argomento.

Figura 12 - UserProfileActivity Pulsanti disabilitati

4.5.2.1 onDisableSuccess

La transazione è stata confermata e verificata e si procede quindi a cambiare lo stato del *Consent* da *Active* a *Disabled*, aggiornare il testo del bottone da "Disabilita" ad "Abilita" e vengono riabilitati entrambi i pulsanti.

4.5.2.2 onDisableFailure

Non viene cambiato il *Consent* e vengono riattivati entrambi i pulsanti in modo che l'utente possa riprendere ad utilizzare la applicazione.

4.5.3 withdrawOption

Il metodo "*withdrawnOption*", collegato al pulsante "Revoca Consenso", quando invocato disabilita sia il bottone "Revoca Consenso" che quello "Disabilita Consenso", quindi, in modo similare a "*changeLocationConsent()*", invia una transizione tramite "*sendTransaction*" dopo aver richiesto l'autorizzazione all'utente. Nel caso la transizione venga confermata, viene invocato il metodo "*onWithdrawnSuccess*", altrimenti in caso di transizione non confermata o di errori viene invocato "*onWithdrawnFailure*".

4.5.3.1 onWithdrawnSuccess

Questo metodo revoca il consenso all'account relativo al servizio *serviceProva*, impostando il relativo *Consent* a *Withdrawn* e indirizza alla *NewAccountActivity*, non avendo più un account *serviceProva* attivo.

4.5.3.2 onWithdrawnFailure

In questo caso il *Consent* resta immutato e vengono riabilitati i pulsanti "Revoca Consenso" e "Disabilita Consenso".

4.5.4 viewDataConsents

Attraverso questo metodo invocato alla pressione del pulsante "Visualizza Data Consent" si verrà indirizzati alla *DataConsentActivity*, una schermata semplice in cui sono visualizzabili tutti i *Consent* relativi ai dati emessi, come visibile in Figura X.

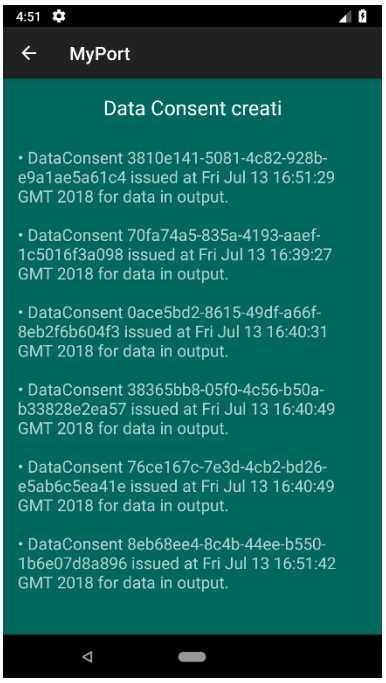

Figura 13 - DataConsentActivity

5 Capitolo quinto

Conclusioni e sviluppi futuri

In questa tesi è stato inserito un sistema di autenticazione e gestione delle autorizzazioni tramite blockchain in un esistente servizio modulare, basato su MyData, per piattaforma Android. In questo modo le preferenze degli utenti saranno salvate in modo permanente e sicuro nella blockchain, così da garantire che vengano rispettate.

Questo progetto basa le sue fondamenta sulla tesi della collega Valentina Protti, ovvero lo sviluppo del modulo Android di base MyDataModule. Una fondamentale risorsa per la comprensione dell'architettura e del modulo MyData è stata fornita dall'elaborato della collega Giada Martina Stivala.

Il progetto ha ampie possibilità di sviluppo e miglioramento, impediteci in questo momento dalla mancanza dei giusti strumenti che, come detto in precedenza, sono ancora in fase di sviluppo. Una volta completati si potrà far accedere l'utente con la sua identità Uport tramite la applicazione Android e, sempre tramite di essa, visionare e confermare le transazioni che gestiscono le autorizzazioni dei dati personali.

Un secondo sviluppo interessante è la possibilità di collegare all'identità Uport un blob di dati salvati su cloud. Così facendo sarebbe possibile conservare in modo sicuro il Personal Data Vault dell'utente e gestire completamente i dati attraverso transizioni.

Il codice sorgente della applicazione sviluppata e discussa nell'elaborato può essere visionato su Github, con le classi implementate e le modifiche apportate al modulo MyData così come sono esposte nel capitolo 4, al repository MyPort-sdk.

Bibliografia

- MyData, sito ufficiale. Ultima visita Lug. 2018. URL: https://mydata.org.
- MyData, whitepaper. Ultima visita Lug. 2018. URL: https://www.lvm.fi/documents/201 81/859937/MyData-nordic-model/2e9b4eb0-68d7-463b-9460-821493449a63?version=1.0.
- SMAll documentation on github. Ultima visita Lug. 2018. URL: https://github.com/small-dev/SMAll.Wiki
- Uport, sito ufficiale. Ultima visita Lug. 2018. URL: https://www.uport.me/#about.
- Uport, whitepaper. Ultima visita Lug. 2018. URL: http://blockchainlab.com/pdf/ uPort_whitepaper_DRAFT20161020.pdf
- Uport specification on github. Ultima visita Lug. 2018. URL: https://github.com/uport-project/specs
- Uport-connect on github. Ultima visita Lug. 2018. URL: https://github.com/uport-project/uport-connect
- Blockchain. Ultima visita Lug. 2018. URL: https://en.wikipedia.org/wiki/Blockchain
- Android, operating system. Ultima visita Lug. 2018. URL: https://en.wikipedia.org/ wiki/Android_(operating_system)
- Android Shared Preferences. Ultima visita Lug. 2018. URL: https://developer. android.com/reference/android/content/SharedPreferences
- Android View.OnClickListener. Ultima visita Lug. 2018. URL: https://developer. android.com/reference/android/view/View.OnClickListener

- Android WebView. Ultima visita Lug. 2018. URL: https://developer. android.com/reference/android/webkit/WebView
- Android TextView. Ultima visita Lug. 2018. URL: https://developer. android.com/reference/android/widget/TextView
- Android Studio, sito ufficiale. Ultima visita Lug. 2018. URL: https://developer. android.com/ studio/intro/
- Android Studio. Ultima visita Lug. 2018. URL: https://en.wikipedia.org/ wiki/Android_Studio
- XML. Ultima visita Lug. 2018. URL: https://en.wikipedia.org/wiki/XML
- GoBoBus app su PlayStore. Ultima visita Lug. 2018. URL: https://play.google.com/ store/apps/details?id=unibo.progettotesi&hl=en_IN
- Stefano Gasperini e Francesco Fiacco. GoBoBus repository on github. Ultima visita Lug. 2018. URL: https://github.com/sgasperi/TesiSF.
- Giada Martina Stivala. Analisi e sviluppo di un gestore di dati personali secondo il modello MyData - repository on Github. Technical report. Ultima visita Lug. 2018. URL: https://github.com/ShurayukiHime/tesi.
- Giada Martina Stivala. Analisi e sviluppo di un gestore di dati personali secondo il modello MyData - Tesi. Ultima visita Lug. 2018. URL: https://github.com/ShurayukiHime/tesi/blob /master/Tesi%20Latex/Tesi%20Latex/tesi.pdf.
- Valentina Protti. Progetto e realizzazione di componenti per la protezione dei dati personali su piattaforma Android. Repository su github. Ultima visita Lug. 2018. URL: https://github.com/valentinaprotti/MyDataModuleTesi
- "How to pass a function as a parameter in Java?" on Stack Overflow. Ultima visita Lug. 2018. URL:

https://stackoverflow.com/questions/4685563/how-to-pass-a-function-as-a-parameter-in-java

- "Call external javascript functions from java code" on Stack Overflow. Ultima visita Lug. 2018. URL: https://stackoverflow.com/questions/22856279/call-external-javascript-functions-from-java-code
- Ethereum wiki, transactionReceipt details on github. Ultima visita Lug. 2018. URL: https://github.com/ethereum/wiki/wiki/JSON-RPC
- MyPort-sdk, di Bartolotti Luca su Github. Ultima visita Lug. 2018. URL: https://github.com/BartolottiLuca/MyPort-sdk

Ringraziamenti

Ringrazio i miei genitori, per avermi sempre supportato economicamente, ma soprattutto moralmente. Grazie Mamma, Grazie Papà. Un grazie particolarmente sentito a mia sorella, Giulia, per tutto l'aiuto che mi ha fornito nelle ultime settimane, e non solo. Ringrazio tutti i parenti che mi stanno vicino da sempre, la cui lista sarebbe troppo lunga da fare. Grazie.

Ringrazio il prof. Prandini e il correlatore dott. Melis per avermi concesso la possibilità di conseguire questo importante traguardo eseguendo un progetto di mio interesse, oltre che per la disponibilità e la pazienza dimostrata.

Un sentito Grazie è rivolto al prof. Vassura che, durante le sue lezioni al liceo, mi fece appassionare all'informatica, questa materia bella e dannata.

Un Grazie enorme ai miei più cari amici, che mi sopportano sempre, in qualsiasi condizione io sia, che sanno sempre come farmi tornare il sorriso in momenti bui e per la loro incomparabile amicizia. Grazie a Matteo, mio compagno di avventure e sventure da quasi 15 anni, a Riccardo, con cui sono cresciuto e ho passato buona parte della mia infanzia, ma ancora oggi rimane un grandissimo amico, a Mirco, sempre pronto ad ascoltare le mie lamentele e replicare a tono, e a Giorgio, per avermi spronato a modo suo, brum brum mf.

Grazie a Elisa, alla quale rubo spesso il fidanzato e finisce per dovermi sopportare anche quando sono insieme.

Un immenso ringraziamento a Silvia, una donna forte che sa ascoltarmi e riesce sempre a rimettermi in riga, Grazie Tigre.

Ringrazio di cuore Dario, che mi ha insegnato tanto allenandomi mente e corpo, non soltanto per lo sport. "Ogni traguardo non è altro che un nuovo punto di partenza".

Grazie a Chiara, al suo supporto morale e ai suoi consigli, a Katia, per i bei momenti passati assieme, a Elena, che riesce sempre a trovare una parola gentile, e a Maria, per la musica e i memes.

Ringrazio i miei valorosi compagni di corso per l'aiuto e la compagnia di questi anni, grazie Beatrice, Marcello, Filippo, Danilo, Marco, Andrea, Matei e Alessandro, che il vento del drago gonfi le vostre vele.

Grazie anche ai coinquilini con i quali ho passato due anni fantastici a Bologna: Giorgia, Carlo, Alessandro, Giulia.

Ringrazio gli Uroboros e i Leather Jacket per le fantastiche avventure condivise fra un esame e l'altro.

Grazie, e scusa, anche a chi ho dimenticato, spero mi perdonerete.

E ora che anche questa avventura è finita voglio ringraziare di nuovo tutti, ma sappiate:

"IN THE END DAVE GROHL WILL SAVE US ALL"