Università degli Studi di Bergamo
Scuola di Ingegneria
Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione
Corso di laurea in Ingegneria Informatica

Gestione abbonamenti autobus tramite Arduino e NFC

Relatore: Prof. Paraboschi Stefano

Prova finale di:
Daniele Ravasio
Matricola 1045934

Anno Accademico 2018-2019

Abstract - Italiano

Il progetto MyNBS consiste nella realizzazione di un abbonamento per i mezzi tramite l'uso di chip NFC, l'obiettivo è quello di avere con se un abbonamento facilmente trasportabile e poco ingombrante, inoltre l'utilizzo di quest'ultimo rende anche più semplice alle autorità il controllo e il rinnovo. In questo documento saranno evidenziate le principali tecnologie utilizzate per la creazione di questo progetto ponendo l'attenzione sui protocolli e sui meccanismi di sicurezza implementati. Sarà poi presentato un esempio del funzionamento e sviluppi futuri.

Abstract - Inglese

MyNBS consists in the creation of a bus pass using NFC technology, the aim is to have an easy to carry and a less bulky pass, and even the authorites are facilitated when they have to check the bus pass or when they want to renew a pass. In this document we'll discuss about the technology used for the creation of this project, focussing on the protocols and the security mechanism implemented. After that there will be a practical example of the project and a section of future developments

Ai miei genitori Claudio e Roberta, a mia sorella Monica, ai miei nonni: Antonio, Marisa, Giovanni e Lisetta.

A Francesco, Simone, Marcello, Emanuele, Chrstian, Andrea e Carla per avermi accompagnato in questi 3 anni.

Ad Alessia, per avermi sempre sostenuto, incoraggiato, per essermi stata sempre accanto, e per non avermi mai fatto mollare durante tutto questo cammino!

Indice

1 Introduzione

1.1 Idea

L'idea di base è nata, durante un viaggio nei paesi nordici quando vidi che salendo sui mezzi di trasporto pubblici, le persone utilizzavano delle tessere magnetiche, analizzando e chiedendo scoprii poi essere tessere con all'interno dei chip NFC, andando avanti negli anni scoprii che in altri paesi, oltre all'utilizzo delle tessere in parallelo venivano usati anche gli smartphone con in chip direttamente incluso nel telefono.
Da li ho preso spunto chiedendomi "Perché non portare anche nel nostro paese una tecnologia del genere?" L'idea di fondo è quindi quella di avere un abbonamento sempre a portata di mano, facilmente rinnovabile, meno ingombrante, e anche più sicuro, inoltre è anche un'idea **eco-friendly**, infatti si può pensare che al posto di comprare un biglietto od un carnet di biglietti, per poi buttarli via dopo l'utilizzo, si ha a disposizione una tessera magnetica nella quale viene caricato il biglietto/-carnet, e dopo l'utilizzo basterà semplicemente rinnovarlo o cambiarne la tipologie, evitando così uno spreco di carta.

1.2 MyNBS

MyNBS (My NFC Bus Subscription) è un'applicazione sviluppata in Java con un'interfaccia grafica che permette la sottoscrizione di un abbonamento per i pullman o il controllo di un abbonamento già esistente. Abbiamo quindi due funzionalità che vanno a dividersi in molteplici step:
Sottoscrizione abbonamento:

1. L'operatore inserisce i dati dell'utente e le zone volute per l'abbonamento in un'interfaccia grafica

2. L'operatore posizione sull'antenna NFC il tag, nel quale verranno scritti in maniera codificata i dati dell'utente.

Controllo abbonamento:

1. Il controllore seleziona la/e zona/e dove è in questo momento

2. Posizione sopra il lettore il tag NFC, sia che l'abbonamento è valido per quella zona che non è valido verrà segnalato, nel secondo caso verranno evidenziate le zone di validità dell'abbonamento.

1.3 Obiettivo

L'obiettivo principale è quello di avere con se un abbonamento facilmente trasportabile, poco ingombrante e sicuro, infatti tramite la tecnologia NFC e l'implementazione della crittografia è possibile avere un'autenticazione sicura ed evitare anche che qualcuno di esterno riesca ad interpretare i dati del chip anche se dovesse riuscire a copiarlo. Inoltre per le forze dell'ordine è molto più semplice controllare quel chip e i dati associati piuttosto che dover guardare una carta che andando avanti nel tempo subirebbe l'usura e risulterebbe quindi di difficile comprensione.

2 Strumenti utilizzati

Per la realizzazione della tesi sono stati usati i seguenti strumenti:

- Arduino combinato con NFC Shield v2.0: formano il dispositivo per la lettura/-scrittura degli abbonamenti

- NFC Tag: dispositivo sul quale sono memorizzati gli abbonamenti

- Java: linguaggio di programmazione con il quale è stata progettata l'interfaccia grafica per l'applicazione

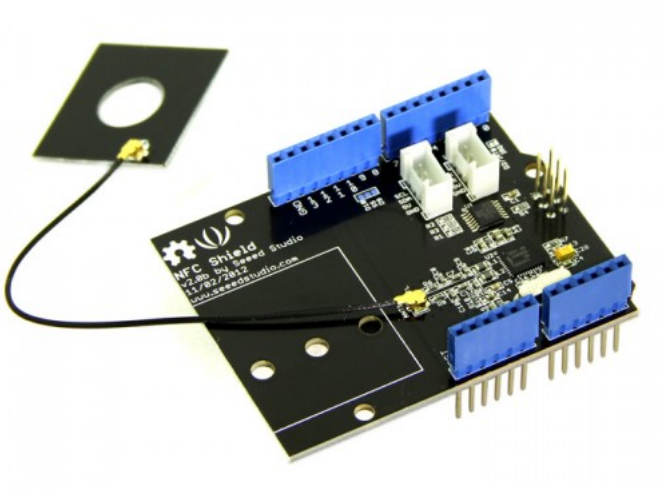
- NoSQL Database: database dove vengono salvati i dati degli utenti.

2.1 Arduino Uno

Arduino è una piattaforma hardware composta da una serie di schede elettroniche dotate di un microcontrollore. È stata ideata e sviluppata nel 2003 da alcuni membri dell'Interaction Design Institute di Ivrea come strumento per la prototipazione rapida e l'utilizzo in vari ambiti, per esempio la robotica e la domotica.

2.2 NFC Shield v2.0

Le shield sono schede che possono vengono inserite sopra l'Arduino, permettono l'estensione delle capacità della scheda stessa.La shield usata un questo progetto è quella NFC composta da un'antenna che collegandosi ad Arduino abilita la capacità di leggere/scrivere sui chip NFC



2.3 NFC Tag

La tecnologia NFC è una combinazione d'identificazione senza contatto (**RFID**) e altre tecnologie di connettività.NFC permette una comunicazione bidirezionale: quando due apparecchi NFC (initiator e target) vengono accostati entro un raggio di 4 cm, viene creata una rete peer-to-peer tra i due ed entrambi possono inviare e ricevere informazioni.

La tecnologia NFC opera alla frequenza di 13,56 MHz e può raggiungere una velocità di trasmissione massima di 424 kbit/s.

Il formato dei chip NFC usato nel progetto è **NDEF**

2.3.1 Tipologie di tag

Prima di iniziare questa sezione è bene sapere che esistono solo cinque tipi di tag standardizzati e appunto per questo il loro funzionamento è garantito con gli accessori che possediamo in questo momento. I nuovi tipi di tag richiederanno una standaridzzazione e potrebbero anche richiedere un aggiornamento dell'architettura NFC che possediamo attualmente.

- **Tipo 1**: il primo tipo è quello più semplice a causa di questo viene anche considerato come il più lento, ed essendo così semplice anche a livello di prezzo risulta essere molto economico. Il problema riguarda il fatto che certe funzionalità per delle applicazioni specifiche potrebbero mancare. Solitamente questi tag vengono usati per:

  - Applicazioni in sola lettura
  - Accoppiamento di dispositivi bluetooth
  - Lettura di un tag specifico quando c'è ne sono tanti

- **Tipo 2**: Prima di iniziare questa sezione è bene sapere che esistono solo cinque tipi di tag standardizzati e appunto per questo il loro funzionamento è garantito con gli accessori che possediamo in questo momento. I nuovi tipi di tag richiederanno una standaridzzazione e potrebbero anche richiedere un aggiornamento dell'architettura NFC che possediamo attualmente.

- **Tipo 3**: il terzo tipo si basa su standard differenti rispetto agli altri. Vengono anche conosciuti come i tag della Sony FeliCa, sono un'innovazione giapponese e molto usata in Asia. Offre molte funzionalità però il prezzo è molto alto, vengono appunto usati molto in Giappone e per questi tipi di applicazioni:

  - Biglietti
  - Moneta elettronica

- Dispositivi per l'assistenza sanitaria

• **Tipo 4**: il quarto tipo di tag offre la maggior memoria e flessibilità tra tutti. Il prezzo è variabile però può aggirarsi dal medio all'alto, in funzione alla quantità di memoria che vuoi. La funzione più importante di questo tag è la *sicurezza*, infatti è equipaggiato con funzionalità che permettono la "true authentication". Inoltre questo tag è l'unico che supporta lo standard ISO 7816 relativo alla sicurezza, inoltre permette l'auto-modifica del contenuto NDEF. Solitamente questo tag viene usato per applicazioni di biglietteria.

• **Tipo 5**: il quinto tipo offre supporto per la specifica ISO 15693, viene supportata la Modalità di Comunicazione Attiva, che permette di ottenere performance in ambito di trasferimento dei dati simili alle tecnologie RF. La distanza di lettura è uguale a quella degli altri tag NFC, solitamente questi tag vengono usati per:

- Impacchettamento di prodotti

- Biglietti

- Dispositivi per l'assistenza sanitaria

2.3.2 Standard ISO

Ci sono diversi standard ISO usati per la tecnologia NFC tra cui: ISO 15693, 18092 e 21481 poi ECMA 340, 352 e 356 ed ETSI TS 102 190. NFC è inoltre compatibile con la diffusa architettura delle smart card contactless, basate su ISO 14443 MIFARE e Sony FeliCa (tipo 4).

• ISO 15693: Lo standard ISO 15693 utilizza la frequenza 13.56 MHz, viene offerta una distanza di lettura che può variare tra 1 metro ed 1.5 metri, poiché le carte devono operare a distanza, è richiesta la presenza di campi magnetici inferiori rispetto a quelli usati per altre carte. Inoltre a differenza di altri standard c'è una funzione di anticollisione implementata, questo serve per consentire una lettura simultanea di più card senza incorrere in errori o fail di ricezione.

- ISO 18092, definisce lo standard di una smartcard contactless, viene prevalentemente usato nei sistemi RFID (pre-NFC). Questa tecnologia viene usata con i tag di tipo 4 quindi in Giappone. Questa specifica è formata dalle ISO 15693 (vista prima) e dalla ISO 9798 che regolamenta il protocollo di comunicazione a mutuo riconoscimento, questo implica che il tag può essere letto solo da un lettore già programmato.

2.4 NoSQL Database

I database NoSQL (non relazionali), sono realizzati per modelli di dati specifici, hanno degli schemi flessibili, un'ottima facilità di sviluppo, una scalabilità orizzontale e delle prestazioni ottime. Si può pensare all'attuale diffusione dei sistemi cloud, quindi una diffusione dove ci sono moltissimi nodi e usare un RDMBS (database relazionale) in questi casi diventerebbe molto difficile. I principali metodi d'implementazione dei database NoSQL sono:

- Chiave-valore: I dati sono immagazzinati in un elemento che contiene una chiave oltre che i dati veri e propri, dal punto di vista implementativo questo è il metodo più semplice, però è anche quello meno efficiente nel caso nel quale le operazioni riguardano solo una parte dell'elemento e non l'elemento nel totale

- Column family: i dati vengono organizzati in righe e colonne, non si ha il bisogno di definire il numero di colonne e una riga può avere molte colonne

- Document store: è un'evoluzione del metodo chiave-valore, i dati vengono salvati in un documento che può contenere un numero "infinito" di campi di "infinita" lunghezza, in questo modo riusciamo ad evitare sprechi di campi inutilizzati che andrebbero ad occupare spazio in memoria.

2.5 WindowBuilder

Per la realizzazione dell'interfaccia grafica in Java è stato usato il plug-in WindowBuilder di Eclipse. Questo plug-in è composto a partire dalle liberire SWT

Designer e Swing Designer e rende comoda e veloce la creazione di interfacce grafiche (GUI) per le applicazioni Java. Usando il WYSIWYG visual designer e gli strumenti di layout è possibile creare finestre complesse, e per ogni elemento messo verrà generato il codice contente la posizione dell'elemento e la sua dichiarazione.
Inoltre, il codice generato da questa libreria non richiede l'uso di altre librerie personalizzate per compilarlo ed eseguirlo, inoltre è possibile dall'interfaccia grafica creare eventi che poi andranno compilati mediante codice scritto in dei blocchi di *ActionListener*, è possibile generare diversi tipi di eventi, dal click del mouse, alla pressione di un tasto sulla tastiera, o persino al movimento nella finestra del mouse.

2.6 Cassandra

Apache Cassandra è un DBMS distribuito e open source. Si tratta di un progetto Top-Level, sviluppato da Apache Software Foundation per gestire grandi quantità di dati dislocati in diversi server, fornendo un servizio orientato alla disponibilità.
È una soluzione NoSQL che inizialmente fu sviluppata da Facebook, un modello di dati simile a BigTable in esecuzione su un'infrastruttura tipi Amazon-Dynamo. Cassandra fornisce una struttura di memorizzazione chiave-valore, con Eventual Consistency.

Alle chiavi corrispondono dei valori, raggruppati in famiglie di colonne: una famiglia di colonne è definita quando il database viene creato. Tuttavia le colonne possono essere aggiunte a una famiglia in qualsiasi momento.
Le colonne sono aggiunte solo specificando le chiavi, così differenti chiavi possono avere differenti numeri di colonne in una data famiglia. I valori di una famiglia di colonne sono memorizzati insieme, questo perché Cassandra adotta un approccio

ibrido tra DBMS orientato alle colonne e la memorizzazione orientata alle righe. Come caratteristiche principali abbiamo:

- Fault-tolerance: i dati vengono replicati in maniera automatica su più nodi, la replica è supportata tramite vari data center e la sostituzione dei nodi può avvenire senza downtime[1].

- Tunable consistency: il livello di coerenza può essere modificato (da writes never fail a block for all replicas to be readable).

2.6.1 CQLSH

cqlsh[2] è una shell a linea di comando per interagire con Cassandra attraverso il CQL (Cassandra Query Language), è inclusa con ogni versione di Cassandra, questa shell utilizza dei protocolli nativi di Python e permette la connessione al nodo specificato nella linea di comando [3].
Generalmente parlando, il funzionamento di una data versione di cqlsh è garantito solo con la versione di cassandra che lo ha rilasciato, ci possono essere casi particolari nei quali vengono rilasciati update per far lavorare un cqlsh datato con una nuova versione di Cassandra, però questo non è supportato ufficialmente.

[1] la replicazione verrà trattata nel capitolo 3.3.2
[2] http://cassandra.apache.org/doc/latest/tools/cqlsh.html
[3] un esempio di connessione sarà trattato nel capitolo 3.3.4

3 Implementazione

In questa sezione verranno illustrate le tecnologie utilizzate per la realizzazione dell'elaborato e verranno commentati dei tratti di codice fondamentali

3.1 NDEF e Sicurezza

NDEF è un preciso standard per un formato di dati sui chip NFC, viene utilizzato in applicazioni quali le carde di credito, o gli smart poster, la struttura di un messaggio è quella vista nella seguente figura: In ordine abbiamo:

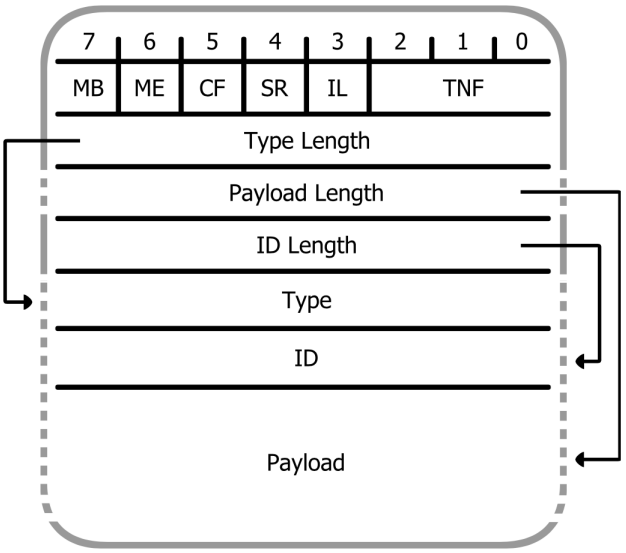


Figura 1: Vulnerabilità NDEF[4]

Header Che è composto da: Message Begin (MB), Message End (ME), Chunk Flag (CF), Short record (SR), ID Length present (IL), Type Name Format (TNF), Length fields, Type, ID

I primi 5 parametri sono dei flag, per finire invece abbiamo il **Payload** che è il messaggio.

Detto questo andiamo a concentrarci meglio su determinati campi quindi:

[4] `www.researchgate.net/publication/224227216_Security_Vulnerabilities_of_the_NDEF_Signature_Record_Type`

- CF: indica il record che fa parte di una catena di record, quando il suo valore è pari a 1 vuol dire che c'è *almeno* 1 altro record nella catena, invece quando non è settato vuol dire che ci troviamo o nel caso di record singolo o nel caso di ultimo elemento della catena. Una cosa da notare è che ME e CF non possono essere entrambi settati a 1

- SR: quando viene messo ad 1 vuol dire che il record corrente è di tipologia short, questo comporta che il campo Payload Length, che si trova dentro Length fields, viene espresso tramite 1 byte, in alternativa viene espresso da 4 byte

- IL: indica se nel record è presente un identificatore, se non dovesse essere abilitato questo comporta che non vi saranno il campo ID Length e il campo ID che è quello di identificazione del record.

- TNF: serve per indicare la struttura del campo Type, è composto da 3 bit e può assumere solo i valori da 0 a 6 perché il numero 7 è riservato.

NDEF ha anche una firma

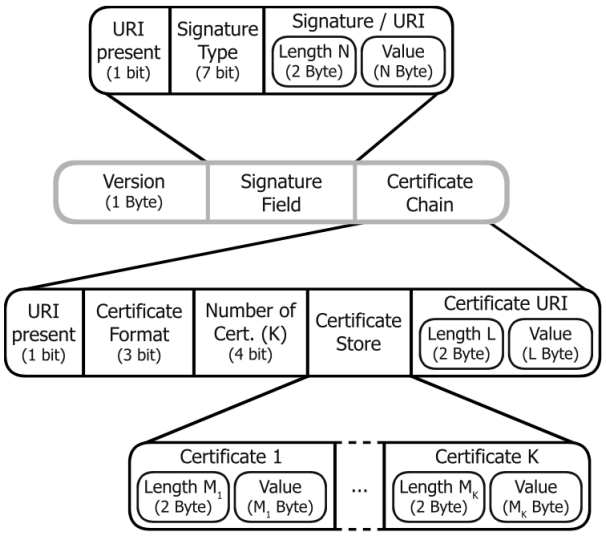


Figura 2: Vulnerabilità NDEF[5]

[5] `www.researchgate.net/publication/224227216_Security_Vulnerabilities_of_the_NDEF_Signature_Record_Type`

L'ultimo documento di specifica è stato rilasciato nel Novembre 2010, sostanzialmente ha una struttura fatta da un **campo firma**, che può essere una firma o una referenza URI ad una firma e da una **catena di certificati**, che è una catena di certificati PKI su un percorso sicuro.
Il record con la firma viene aggiunto ad una sequenza di record, e questo firma ogni record tra il record di firma appena precedente e se stesso, infatti un messaggio NDEF può contenere più di una firma.

3.1.1 Cosa viene firmato?

I campi che non vengono firmati sono MB/ME, questo perché se venissero firmati la firma non potrebbe essere agganciata al messaggio NDEF già firmato. Type, ID e Payload invece vanno firmati per assicurare l'integrità dei dati, mentre quando TNF viene cambiato, l'intero significato del record cambia, può essere quindi usato per nascondere dei record (identificati da type "Unknown").

3.1.2 Sicurezza

La tecnologia NFC è un'evoluzione dell'RFID, da un lato risulta meno predisposta ad attacchi esterni ma dall'altro lato è soggetta alle problematiche di sicurezza del suo predecessore. Le possibili minacce di sicurezza a cui sono sottoposti sono quelle riguardanti l'acquisizione, o l'alterazione dei dati contenuti nel tag, queste minacce possono avvenire mediante interrogazioni fraudolente o mediante l'intercettazione delle informazioni mediante ricevitori radio durante la lettura da parte di un lettore autorizzato.

3.1.3 Intercettazioni

Nell'ambito dell'NFC ma soprattutto in maniera generale, nel campo delle comunicazioni wireless l'intercettazione dei dati è uno degli attacchi più comuni. Per effettuare quest'attacco serve attrezzatura progettata ad hoc, quindi antenne e lettori fatti su misura. Per quanto riguarda il caso specifico NFC è un attacco molto difficile da realizzare a causa sei seguenti fattori:

i potenza emessa dallo strumento sotto intercettazione

ii fattori ambientali

iii presenza della crittografia

Quindi da questo capiamo che anche in base al tipo di tag [6] un attacco può essere più facile o più difficile, per esempio un attacco su un tag di tipo 1 sarà molto più semplice che su un tag di tipo 4.
Ci sono anche contromisure come per esempio quella di diminuire il campo magnetico, magari aumentando il fattore di direzionalità delle antenne, oppure usare algoritmi di cifratura per il messaggio, algoritmi per esempio AES.

3.1.4 Modifica dei dati

La modifica dei dati è un problema molto pericoloso, questo perché risulta trasparente all'utente, ha come scopo quello di modificare i dati trasmessi e renderli "validi". Fortunatamente risulta un attacco molto difficile da eseguire perché bisognerebbe riuscire ad intercettare completamente ogni bit del messaggio e rimandarlo sulle frequenze precise, quindi ogni volta modulando il campo delle frequenze in modo specifico e diverso.

3.1.5 Man in the middle

È uno degli attacchi più pericolosi, infatti può arrecare molti danni ai sistemi coinvolti. Mentre sysA e sysB stanno comunicando, sysH, il dispisitivo dell'hacker, si interpone tra di loro. Durante la comunicazione sysH altera il dialogo che hanno sysA e sysB, mettendosi in mezzo e fingendosi sysA per sysB e sysB per sysA. La soluzione a questo attacco è quella di instaurare un canale sicuro, quindi usando una chiave per criptare i dati. Potrebbe succede che sysH cerchi di negoziare una chiave ai due sistemi però è molto complesso perché richiederebbe la visibilità di sysH.

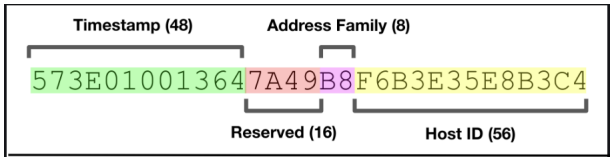
[6] cap 2 par 2.3.1

3.2 UUID

Un UUID[7] conosciuto anche come Universally Unique Identifier, è un numero di 128 bit utilizzato per identificare informazioni nei sistemi informatici, lo si può trovare nell'rfc 4122.

3.2.1 Come è formato?

Il codice è composto da 16 byte, solitamente viene identificato da 32 caratteri esadecimali, a livello di sicurezza assume $3 \cdot 10^{38}$ possibili combinazioni



A differenza di altri sistemi l'unicità del codice generato non dipende da un'autorità centrale di registrazione o dal coordinamento delle parti che lo generano, però la sua probabilità di essere duplicato è talmente vicina a zero che viene considerata trascurabile.

3.2.2 Analisi a livello matematico

Considerando i 128 bit dell'UUID versione 4, 6 bit vengono riservati, quattro per la versione e due per altri parametri, quindi un UUID generato randomicamente ha 122 bit casuali. La probabilità che due UUID hanno lo stesso valore può essere calcolata usando la teoria delle probabilità (Paradosso del compleanno [8]). Usando quindi quest'approssimazione possiamo calcolare:

$$p(n) \approx 1 - e^{-\frac{n^2}{2 \cdot 2^x}} \tag{1}$$

Notiamo quindi che quando il termine $\frac{n^2}{2 \cdot 2^x}$ è vicino allo zero, la probabilità può essere direttamente approssimata in questo modo:

$$p(n) \approx \frac{n^2}{2 \cdot 2^x} \tag{2}$$

[7] https://tools.ietf.org/html/rfc4122
[8] https://betterexplained.com/articles/understanding-the-birthday-paradox/

Analizzandolo quindi in modo numerico potremo pensare che anche generando 1 miliardo di UUID ogni secondo per i prossimi 100 anni la probabilità di creare almeno un duplicato sarebbe circa del 50%.

3.3 Analisi del codice

In questa sezione verranno mostrati i frammenti più importanti del codice

3.3.1 Avvio del Database NoSQL

```
1 #! /bin/bash
2 source ~/.bash_profile
3 cassandra -f
```

Automaticamente il controllore mettendo i suoi dati (validi) apre la connessione al database e va direttamente alla pagina di controllo degli abbonamenti.

3.3.2 Creazione del Database

```
1 CREATE KEYSPACE Abbonamento
2   ... WITH REPLICATION = {
3   ... 'class' : 'SimpleStrategy',
4   ... 'replication_factor' : 3};
```

La creazione del database viene fatta una sola volta, prima del rilascio dell'applicazione.
Potrebbero succedere problemi hardware, oppure il link potrebbe perdere la connessione magari durante un processamento dei dati, per questo una soluzione è quella di avere un backup disponibile appena succede un problema, quindi i dati vengono replicati per assicurare che non ci siano fallimenti. Cassandra posiziona le repliche dei dati su nodi diversi in base a due fattori:

- Strategia di replicazione

- Fattore di replicazione Il primo dice dove posizionare le replica, il secondo invece determina quante repliche vengono posizionate, un esempio lo troviamo nella riga 4 dove abbiamo la stringa *'replication_factor' : 3*, questo indica che vengono fatte 3 repliche su 3 nodi diversi. Solitamente per garantire che non ci siano fallimenti il

fattore di replicazione deve essere 3.
Nella riga 3 vediamo la stringa *'SimplyStrategy'*, questa viene usata quando abbiamo un solo data center, quindi la prima replica viene posizionata sul nodo selezionato dal partizionatore, dopo questo, le restanti repliche vengono posizionate in senso orario a partire dalla direzione del nodo.

3.3.3 Creazione tabella nel Database

```
1 CREATE TABLE abbonato (
2   ... id uuid,
3   ... nome text,
4   ... cognome text,
5   ... Data date,
6   ... zone text,
7   ... PRIMARY KEY (id) );
```

Anche questa parte di codice, come la creazione del database viene fatta una sola volta, prima del lancio dell'applicazione

3.3.4 Connnessione al database mediante CQLSH

```
1 > cqlsh
2
3 Connected to Test Cluster at 127.0.0.1:9042.
4 [cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol
     v4]
5
6 cqlsh:abbonamento> SELECT * from Abbonamento.abbonato;
7
8
9  Id            | nome         | cognome | data       | zone
10 --------------+--------------+---------+------------+---------
11  0a210330**** |[B@1db1b56a] | ******* |  *******   | *******
12
13 [...]
14
15 (21 rows)
```

In questo frammento viene mostrata la connessione con CQLSH ed un esempio di query. Nella riga 3 e 4 vediamo la risposta che viene data in output non appena effettuiamo la connessione, per effettuare la connessione è obbligatorio aver attivato il database[9], altrimenti il servizio non sarà attivo e quindi ci sarà un errore nella connessione di tipo *Connection Refused*.

3.3.5 Connessione al Database NoSQL

```
1  public void connect(final String node, final int port) {
2      CodecRegistry codecRegistry = new CodecRegistry();
3      codecRegistry.register(new DateCodec(TypeCodec.date(),
4       Date.class));
5      cluster = Cluster.builder().addContactPoint(node).
6      withPort(port).withCodecRegistry(codecRegistry).build();
7
8      final Metadata metadata = cluster.getMetadata();
9     [...]
10     session = cluster.connect();
11   }
```

In questo frammento di codice viene mostrato il metodo con il quale viene fatta la connessione al database NoSQL Cassandra!

3.3.6 Apertura della connessione con la porta seriale in Java

```
1  private static final String PORT_NAMES[] = {
2        "/dev/cu.usbmodem143301", // Mac OS X
3        "COM3", // Windows
4    };
5  private static final int DATA_RATE = 9600;
6
7  serialPort = (SerialPort) portId.open(this.getClass().getName(),
8            TIME_OUT);
```

Nel frammento di codice appena visto c'è la dichiarazione delle porte che si vanno ad utilizzare, il **DATA_RATE**, ovvero la quantità di dati digitali che possono

[9] cap 3.3.1

essere trasferiti su un canale in un determinato intervallo temporale e l'apertura della connessione tramite la funzione **portId.open**.

3.3.7 Rilevamento presenza tag NFC Arduino

```
1   NfcTag tag = nfc.read();
2      if(tag.hasNdefMessage())
3      {[...]
4          NdefRecord record = message.getRecord(i);
5          int payloadLength = record.getPayloadLength();
6          byte payload[payloadLength];
7          record.getPayload(payload);
8          Serial.write(payload,payloadLength);
9        [...]}
```

In questo frammento di codice viene evidenziato come il chip NFC se presente viene scannerizzato, l'unica cosa che verrà presa sarà il payload e non l'intestazione! Questo perché l'intestazione ci serve solo a sapere se il chip è formattato in formato NDEF.

3.3.8 Creazione abbonamento in Java

```
1  zoneatt = scritturaZone.getBytes();
2           try {
3
4     String inputLine=input.readLine();
5
6     output.write(zoneatt);
7
8     [...]
9
10       }
```

Nella prima riga notiamo subito la variabile *zoneatt*, in questa vi saranno le zone selezionate in precedenza per la creazione dell'abbonamento, nella riga numero 6 notiamo il comando importantissimo *output.write(zoneatt)*, questo comando ci permette di scrivere sulla porta seriale le zone, che verranno successivamente codificate e poi scritte sul tag NFC.

3.3.9 Scrittura abbonamento nel Database

```
1  {
2    [...]
3    final CassandraConnector client = new CassandraConnector();
4
5    final String ipAddress = "localhost";
6    final int port = 9042;
7
8      client.connect(ipAddress, port);
9    Abbonamento ab = new Abbonamento(client);
10
11   ab.Abbonato(
12       UUIDs.timeBased(),
13       scritturaNome.toString(),
14       scritturaCognome.toString(),
15       scritturaData.toString(),
16       scritturaZone.toString()
17
18     );
19   [...]
20
21   client.esegui();
22 }
```

Le righe 5 e 6 servono per definire i parametri per la connessione, quindi indirizzo IP e porta, la linea 3 serve ad istanziare l'oggetto client che sarà quello che ci permetterà la connessione al database e la linea 9 ci permetterà l'apertura della connessione col database.

Andando avanti nel codice creiamo un abbonamento nel quale vengono passati i parametri da specifiche funzioni[10], per finire troviamo il comando più importante *client.esegui()*, questo ci permetterà l'inserimento dei dati nel database, infatti eseguirà questo comando:

```
1 INSERT INTO abbonamento.abbonato (id, nome, cognome, datascadenza,
    zone) VALUES (?, ?, ?, ?, ?)
```

[10] I parametri sono presi dalla form di creazione dell'abbonamento e vengono crittografati

3.3.10 Scrittura abbonamento su tag NFC

```
1  if(nfc.tagPresent())
2    {
3      NdefMessage message = NdefMessage();
4      message.addUriRecord(s);
5
6      bool success = nfc.write(message);
7      if(success)
8      {
9        [...]
10
11     }else{
12       [...]
13     }
14   }
15   } else
16     Serial.println("\nTag non inserito\n");
17 }
```

In questo frammento si mostra la scrittura del messaggio sul chip NFC, importante è l'istruzione alla linea 3, dove definiamo l'oggetto *NdefMessage*, che sarà esattamente il tipo del tag, la funzione invece per scrivere direttamente sul chip è alla linea 6, ed è *nfc.write(message)*, dopo di questa vengono fatti controlli per verificare che effettivamente il tag sia stato scritto.

3.3.11 Lettura dell'abbonamento Java

```
1  if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
2    try {
3
4      byte []b = val.getBytes();
5      String inputLine=input.readLine();
6
7      if(inputLine.contains("AB"))
8        {
9            int []zoneDivision = new int[inputLine.length()/2];
10           [...]
11
```

```
12        for(int i = 3; i<inputLine.length(); i++) {
13
14              char valoreI = inputLine.charAt(i);
15          if((valoreI == '0') && (ch == false)) {
16                inizio = i;
17                ch = true;
18                }
19              if((valoreI == 'A')&& (ch == true)) {
20                fine = i;
21                break;
22              }
23
24            }
25
26            for(int i = inizio; i<fine; i+=2) {
27
28              char partial1 = inputLine.charAt(i);
29              char partial2 = inputLine.charAt(i+1);
30
31              // vado ad identificare la zona
32              String zone = Character.toString(partial1) +
                              Character.toString(partial2);
33
34
35              zoneDivision[j] = Integer.parseInt(zone);
36
37              j++;
38          }
39          [...]
40
41        }
42
43
44      } catch (Exception e) {
45        [...]
46      }
47    }
```

Una delle istruzioni più importanti la troviamo nella prima linea, questa si attiva ogni volta che sulla porta seriale ci sono dei dati disponibili! Andando avanti nel codice notiamo che vengono fatti vari cicli, questi servono per identificare le zone e per vedere se effettivamente è un abbonamento di MyNBS o no.

4 Sviluppi futuri

In questa sezione verranno illustrati alcuni sviluppi futuri possibili per il progetto

4.1 Applicazione per smartphone

In futuro è possibile pensare ad una realizzazione di questo progetto su smartphone, indifferentemente che siano iOS o Android, questo per il fatto che dal rilascio di iOS 13, tutti gli iPhone, a partire dal 7 sono in grado di leggere e scrivere un tag NFC, inoltre gli ultimi modelli (da iPhone XS in poi) offrono anche il supporto per la lettura in background. Inoltre tutti i dispositivi riescono a lavorare con tutti i tipi di chip[11]

4.1.1 Scelta del linguaggio

Una delle cose fondamentali per l'applicazione riguarda la scelta del linguaggio, si potrebbe pensare a due sviluppatori che sviluppano due app diverse, quindi uno con **swift** e l'altro con **java**, oppure si può pensare ad un team di sviluppatori che lavora insieme ad uno stesso progetto utilizzando per esempio **flutter** oppure anche **kotlin**.

4.1.2 Applicazione ibrida

SI potrebbe pensare anche al fatto di creare un'applicazione ibrida, quindi un'applicazione per il dispositivo che però è stata scritta con il linguaggio web, quindi HTML5, CSS e Javascript. Le applicazioni ibride vengono eseguite dentro un *native container*, e per renderizzare il codice HTML e per processare la logica Javascript, sfruttano il motore del browser, ma non direttamente il browser stesso, questo è reso possibile dal fatto che è implementato un livello di astrazione web-to-native, che permette di accedere alle funzioni del dispositivo che non sarebbero accessibili dalle applicazioni Web mobile.

[11] cap 2.3.1

4.2 Apple Pay, Google Pay

Sono due strumenti di pagamento innovativi che permettono di pagare tramite lo smartphone e smartwatch

4.2.1 Che tipi di pagamenti fanno?

Possono essere fatti pagamenti via mobile, pagamenti in app e pagamenti via web.
Parlando di Apple Pay, permette di pagare con iPhone, iPad e Apple Watch ove previsto, invece Google Pay, funziona su tutti i sistemi Android con un OS pari o superiore ad Android 5, funziona inoltre con gli smartwatch Wear Os che hanno tecnologia NFC. Entrambi offrono la possibilità di fare acquisti in app. La differenza sostanziale sta nei pagamenti via web, infatti per Apple Pay i pagamenti possono avvenire solo su safari e molte volte devono essere confermati tramite lo smartphone, invece per Google Pay, si possono usare diversi broswer, ed essendo già l'account di google già collegato, il pagamento risulta più semplice da fare.

4.2.2 NFC

La tecnologia principale per usarli è quella NFC, in sostanza dopo che viene attivato il servizio, bisogna soltanto avvicinare il dispositivo ad un terminale di pagamento (POS), e il pagamento verrà effettuato, ultimamente basta trovare il simbolo di contactless, e si può usare il dispositivo senza problemi!

4.3 Uso di altre tecnologie

In questa sezione verranno mostrate altre tecnologie con relativi commenti

4.3.1 BLE

BLE, conosciuto anche come bluetooth low energy, è una delle tecnologie di prossimità che sta ricevendo molte attenzioni in questi ultimi anni, viene anche chiamata Bluetooth Smart. Funziona insaturando dei becaon che vengono utilizzati per identificare il dispositivo e comunicare con quest'ultimo. Pensiamo al lettore RFID, questo deve essere sempre attivo e mandare il suo segnale in broadcast, mentre per il BLE, l'utente deve obbligatoriamente attivare il bluetooth del dispositivo. Inoltre, l'utente in molti dispositivi è obbligato a scaricare un'applicazione per ricevere e inviare messaggi.
Un esempio di uso del BLE per l'applicazione vista è quando l'utente sale su un autobus, passando attraverso le porte, con l'app attiva, viene subito riconosciuto se l'utente è salito nella zona giusta o è in una zona non prevista per il suo abbonamento. Uno svantaggio però risiede nel fatto che attualmente usando il BLE non si possono effettuare acquisti.
La differenza tra NFC e BLE, riguarda sia l'infrastruttura che i compiti svolti, BLE infatti riesce a lavorare anche a distanze di circa 50 metri, trasferisce più dati però richiede un'infrastruttura dedicata da installare per il suo funzionamento, e per quanto riguarda la UX[12] bisognerebbe avere un'app installata. NFC invece necessita solo di un lettore, implementato anche in uno smartposter oltre che in uno smartphone, l'utente necessita quindi solo di avvicinare il telefono per far succedere qualcosa, inoltre con NFC è possibile effettuare i pagamenti.

4.3.2 QR Code

Un'altra alternativa al chip NFC sono i QR Code, i loro vantaggi risiedono nel fatto che sono facilmente stampabili e hanno un costo molto ridotto, inoltre

[12]User Experience

puoi metterli quasi ovunque, lo svantaggio però è che devi avere un'app[13], aprirla, scannerizzarlo e dopo questo sarà visibile il contenuto. Un'altro svantaggio è dato dal fatto che luci cattive o l'inchiostro che svanisce possono dar problemi per la lettura. NFC invece differisce per il fatto che direttamente dallo smartphone, quando tu lo vai ad attivare l'unica cosa che devi fare è avvicinare lo smartphone e il gioco è fatto.

Sul mercato però ci sono anche smartphone che non supportano l'NFC, che però possono interagire tranquillamente con i QR Code, quindi in questo caso si andrebbe a verificare una perdita d'efficienza per il servizio smartphone e NFC. C'è da dire che con il QR non si può effettuare alcun pagamento, quindi l'idea per l'implementazione risiederebbe nella parte della Creazione dell'abbonamento, quindi scannerizzando un QR Code si viene portati ad una pagina internet o in un'applicazione dove si dovrà creare un'utenza e da lì sarà possibile effettuare il pagamento.

[13]Gli smartphone di ultima generazione hanno questa funzione integrata nella fotocamera, quindi non serve un'app dedicata

5 Bibliografia

[1] Vedat Coskun, Kerem Ok, Busra Ozdenizci (2011). Near Field Communication (NFC): From Theory to Practice

[2] Leach, P.; Mealling, M.; Salz, R. (2005). A Universally Unique IDentifier (UUID) URN Namespace

[3] Ruth Stryker (2014) Apache Cassandra: Hands-on-training