ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

TITOLO DELLA RELAZIONE FINALE

Studio ed Implementazione di un servizio di “Sim Authentication” tramite messaggistica online

Relazione finale in
Algoritmi e strutture di dati

Relatore

Luciano Margara

Presentata da

Brizzi Alessandro

Sessione: III° sessione
Anno Accademico: 2013/2014

A tutti quelli che mi hanno sostenuto
e accompagnato durante questo cammino.

INDICE

Introduzione

In questa tesi si andrà a presentare la realizzazione di un sistema che permette di autenticare la sim di un cellulare, quindi di verificare che un determinato numero di telefono sia realmente nelle mani dell'utente, attraverso servizi di messaggistica online come WhatsApp o Telegram.

Il bisogno di questo progetto nasce dal fatto che molte delle attuali applicazioni presenti sul mercato utilizzano sms per autenticare la reale appartenenza di un numero telefonico ad un determinato utente, ma questo comporta un notevole dispendio di soldi, dato che per inviare un gran numero di sms, bisogna sottoscrivere un abbonamento a servizi Gateway Sms che permettono di inviare sms ad un determinato numero dopo aver ricevuto i relativi dati dal mittente. Molto spesso i costi di questi servizi corrispondono a poco meno del costo di sms inviato da cellulare, quindi se il nostro servizio ha un numero consistente di utenti, il costo per mantenere un servizio che magari per i nostri utenti risulta fruibile gratuitamente, per noi risulterà una spesa esorbitante, specialmente all'inizio. Per questo si è pensato di usare servizi come WhatsApp e Telegram che già autenticano la sim dell'utente attraverso sms e quindi se si verifica che quel determinato numero collegato al determinato account di messaggistica è dell'utente allora sarà ovvio che il numero che stiamo verificando apparterrà senza dubbio al nostro cliente.

Il progetto si è sviluppato dopo un attento studio delle problematiche legate ad esso e delle ricerche relative alla presenze sul mercato di servizi simili, che al momento della stesura del progetto risultavano relativamente scarse, dopodiché si è passati allo studio delle tecnlogie applicabili per il progetto, ricordando che questo è stato sviluppato sotto ambiente Linux Ubuntu, la scelta è ricaduta su tecnologie gratuite e opensource. La parte più complessa è stata quella di esaminare i metodi disponibili per inviare messaggi WhatsApp e Telegram che sono le due piattaforme più utilizzate sempre al momento della creazione del progetto, le difficoltà erano date dal fatto che le due applicazioni sono unicamente per dispositivi mobili quindi si sono dovuti adattare al progetto dei client che permettono l'utilizzo di comandi shell per l'invio di messaggi su queste due piattaforme.

Le tecnlogie principali in uso in questo progetto sono basate su linguaggio Java, dato che la piattaforma su dispositivo mobile è stata sviluppata per Android, per la quantità di mercato acquisita, sempre in crescente aumento, e per la versatilità di sviluppo; anche se si potrebbe benissimo utilizzare un dispositivo con sistema Ios avendo gli strumenti adatti a disposizione. Il client da cui partirà la richiestà per autenticare il numero quindi sarà su dispositivo Android mentre il server potrà essere liberamente collocato e funzionerà tramite Tomcat Rest Webservice, quindi all'arrivo di una richiesta da parte del nostro Client il servizio Rest risponderà di conseguenza in base al percorso selezionato dal Client che varia in basa al tipo di richiesta, richiesta del codice di autenticazione e invio dei relativi dati associati all'utente o invio del codice di verifica di registrazione dell'utente. Tutti i dati saranno mantenuti all'interno di un database in SqlLite che al momento del suo utilizzo risulta molto semplice, ma questo non compromette future applicazioni dato che il concetto che andiamo ad esporre è sperimentale e verificato il suo funzionamento potrebbe tranquillamente essere utilizzato da una qualsiasi applicazione più o meno complessa.

Nel **Capitolo 1** si andranno a descrivere nel dettaglio le tecnlogie utilizzate per lo sviluppo per progetto.

Nel **Capitolo 2** si andrà a descrivere nel dettaglio il funzionamento del Client Android con tutte le sue relative classi e codice.

Nel **Capitolo 3** si descriverà invece il funzionamento dettagliato del Webservice REST in Tomcat e di tutte le relative chiamate possibili, classi e codice.

Nel **Capitolo 4** si esporrà invece il funzionamento della libreria Yousup (WhatsApp) e delle relative chiamate che permettono la comunicazione con il Webservice.

Nel **Capitolo 5** invece si andrà ad esporre il funzionamento del Client di Telegram e delle relative chiamate che lo collogano al Webservice.

Nel **Capitolo 6** infine si discuteranno i risultati ottenuti e le future applicazioni.

Capitolo 1

Tecnologie Utilizzate

In questo capitolo saranno descritte le tecnologie utilizzate per la realizzazione della nostra applicazione.

L'architettura di base è stata implementata in modo RESTful, architettura che sarà descritta nelle caratteristiche e prerogative. Tale architettura è ormai considerata uno standard nella realizzazione di Web Service.

La trasmissione dello stato di rappresentazione delle risorse avviene attraverso l'uso del formato JSON. La scelta di tale formato, alternativo all'XML, è determinata dalla sua semplicità di utilizzo ed efficacia nella descrizione di ogni tipo di risorsa.

Per lo sviluppo dell'applicativo, destinato ai terminali mobili, si è imposta la scelta di Android per la quantità di mercato acquisita, sempre in crescente aumento, e per la versatilità di sviluppo.

Queste scelte hanno determinato la realizzazione di un applicativo facilmente riutilizzabile in altre categorie di dispositivi come i tablet o i mini-PC e soprattutto integrabile con i sempre più numerosi componenti messi a disposizione dei developer Android.

1.1 PARADIGMA REST

REpresentational State Transfer (REST) è un tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web.

REST si riferisce ad un insieme di principi di architetture di rete, i quali delineano come le risorse sono definite e indirizzate. Il termine è spesso usato nel senso di descrivere ogni semplice interfaccia che trasmette dati su HTTP senza un livello opzionale come SOAP o la gestione della sessione tramite i cookie.

È possibile progettare ogni sistema software complesso in accordo con l'architettura REST senza usare HTTP e senza interagire con il World Wide Web.

I sistemi che seguono i principi REST sono spesso definiti "RESTful".

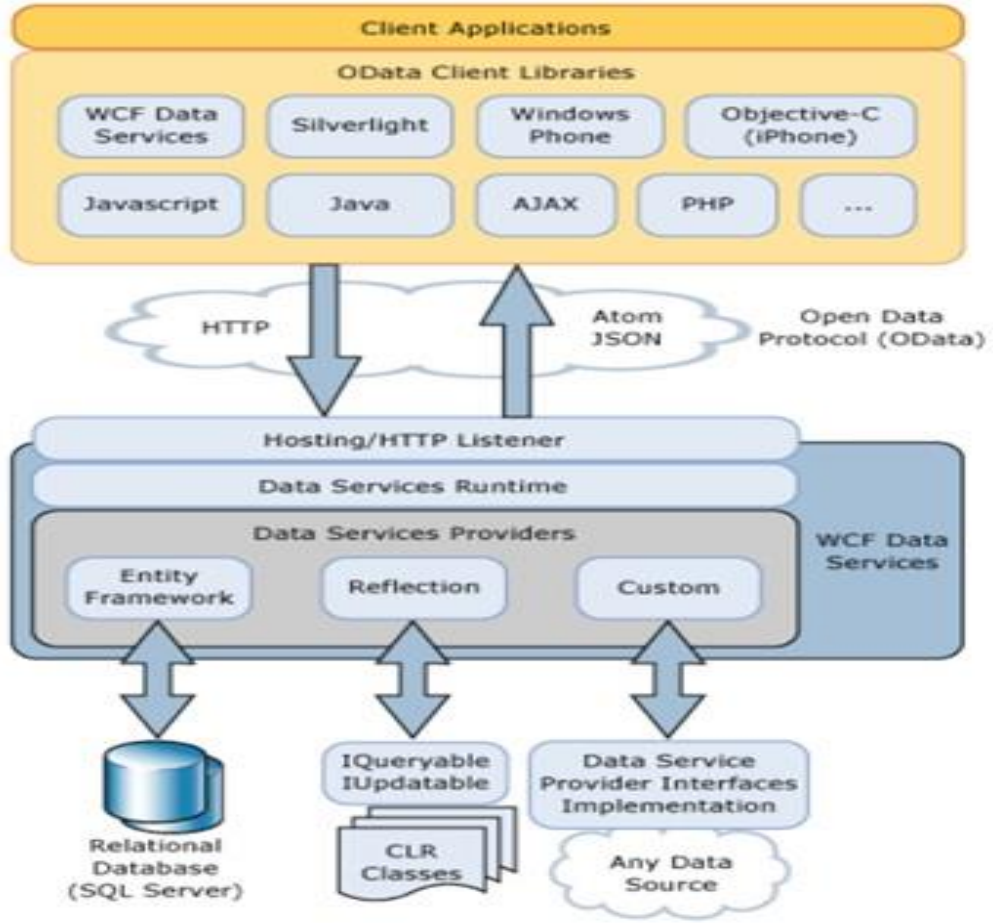


Figura 1.1 – Tipica Struttura RESTful

1.1.1 PRINCIPI REST

I servizi REST o RESTful Service sono basati sul principio fondamentale dell'utilizzo delle risorse (fonti di informazioni).

Per utilizzare le risorse, le *componenti* di una rete (componenti client e server) comunicano attraverso una interfaccia standard (ad es. HTTP) e si scambiano *rappresentazioni* di queste risorse (il documento che trasmette le informazioni).

Un numero qualsiasi di *connettori* (client, server, cache, ecc.) può mediare la richiesta, ma ogni connettore interviene senza conoscere la “storia passata” delle altre richieste.

Di conseguenza una applicazione può interagire con una risorsa conoscendo due cose: l'identificatore della risorsa e l'azione richiesta - non ha bisogno di sapere se ci sono proxy, gateway, firewalls, tunnel, ecc tra essa e il server su cui è presente l'informazione cercata.

L'applicazione comunque deve conoscere il formato dell'informazione (*rappresentazione*) restituita, tipicamente un documento HTML, XML o JSON,come nel nostro caso, ma potrebbe essere anche un'immagine o qualsiasi altro contenuto.

1.1.2 VINCOLI

Client–server. Un insieme di interfacce uniformi separa i client dai server. Questa separazione di ruoli e preoccupazioni significa che, per esempio, il client non si deve preoccupare del salvataggio delle informazioni, che rimane all'interno di ogni singolo server, in questo modo la portabilità del codice del client ne trae vantaggio.

I server non si devono preoccupare dell'interfaccia grafica o dello stato dell'utente, in questo modo i server sono più semplici e maggiormente scalabili.

Server e client possono essere sostituiti e sviluppati indipendentemente fintanto che l'interfaccia non viene modificata.

Stateless. La comunicazione client–server è ulteriormente vincolata in modo che nessun contesto client venga memorizzato sul server tra le richieste.

Ogni richiesta da ogni client contiene tutte le informazioni necessarie per richiedere il servizio, e lo stato della sessione è contenuto sul client.

Cacheable. Come nel World Wide Web, i client possono fare caching delle risposte.

Le risposte devono in ogni modo definirsi implicitamente o esplicitamente cacheable o no, in modo da prevenire che i client possano riusare stati vecchi o dati errati.

Una gestione ben fatta della cache può ridurre o parzialmente eliminare le comunicazioni client server, migliorando scalabilità e performance.

Layered system. Un client non può dire se è connesso direttamente ad un server di livello più basso od intermedio, i server intermedi possono migliorare la scalabilità del sistema con load-balancing o con cache distribuite.

Layer intermedi possono offrire inoltre politiche di sicurezza.

Uniform interface. Un'interfaccia di comunicazione omogenea tra client e server permette di semplificare e disaccoppiare l'architettura, la quale si può evolvere separatamente. **[1]**

1.2 Il formato JSON

JSON, acronimo di **JavaScript Object Notation**, è un semplice formato per lo scambio di dati, in applicazioni client-server.

Esso è stato formulato da Douglad Crockford ed è stato scritto usando le forme sintattiche dello JavaScript. **[2]**.

Si tratta di una rappresentazione formale di progetti, di qualsiasi complessità, che utilizza, dello JavaScript, la sintassi usata, principalmente, per gli array e gli oggetti. **[3]**

Si basa su un sottoinsieme del Linguaggio di Programmazione JavaScript, Standard ECMA-262 Terza Edizione - Dicembre 1999.

JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C, come C, C++, C#, Java, JavaScript, Perl, Python, e molti altri. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati.

JSON è basato su due strutture:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.
- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

Alla base della rappresentazione JSON abbiamo quindi due strutture tipo.

La prima creata secondo un insieme di coppie nome-valore (oggetto), la seconda è un elenco ordinato di valori (array):

• Un **oggetto** è descritto tra parentesi graffe ({ }), sotto forma di coppie nome + valore, separate da virgole. Il nome è dato come una stringa di caratteri racchiusa fra apici e, il valore, può essere un qualsiasi valore letterale JavaScript: numero, stringa di caratteri, valore booleano, e così via.

• Un **array** è descritto fra parentesi quadre ([ ]) ed i suoi elementi sono separati da una virgola. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza. Queste sono strutture di dati universali.

Virtualmente tutti i linguaggi di programmazione moderni li supportano in entrambe le forme. Questa caratteristica è necessaria nella realizzazione di un formato di dati che sia interscambiabile con diversi linguaggi di programmazione.
Analizziamo ora nel dettaglio e schematicamente la rappresentazione dei dati in JSON:

Un ***oggetto*** è una serie non ordinata di nomi/valori. Un oggetto inizia con { (parentesi graffa sinistra) e finisce con } (parentesi graffa destra). Ogni nome è seguito da : (due punti) e la coppia di nome/valore sono separata da , (virgola).

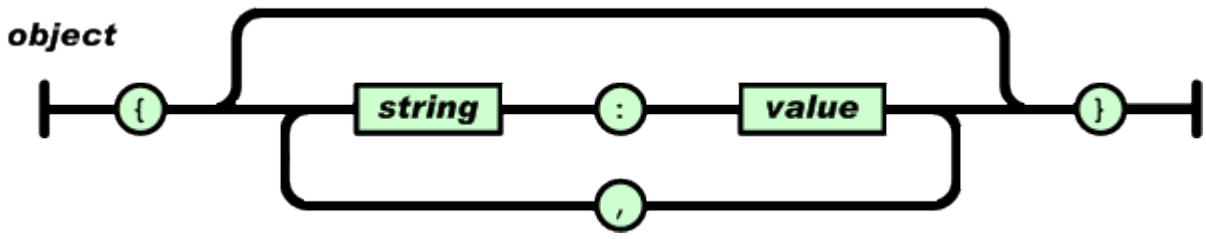


Figura 1.2 - JSON Object

Un ***array*** è una raccolta ordinata di valori. Un array comincia con [(parentesi quadra sinistra) e finisce con] (parentesi quadra destra). I valori sono separati da , (virgola).

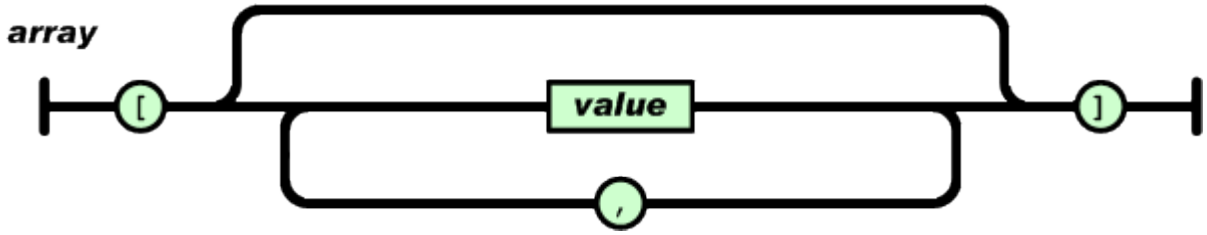


Figura 1.3 - JSON Array

Un ***valore*** può essere una stringa tra virgolette, o un numero, o vero o falso o nullo, o un oggetto o un array. Queste strutture possono essere annidate.

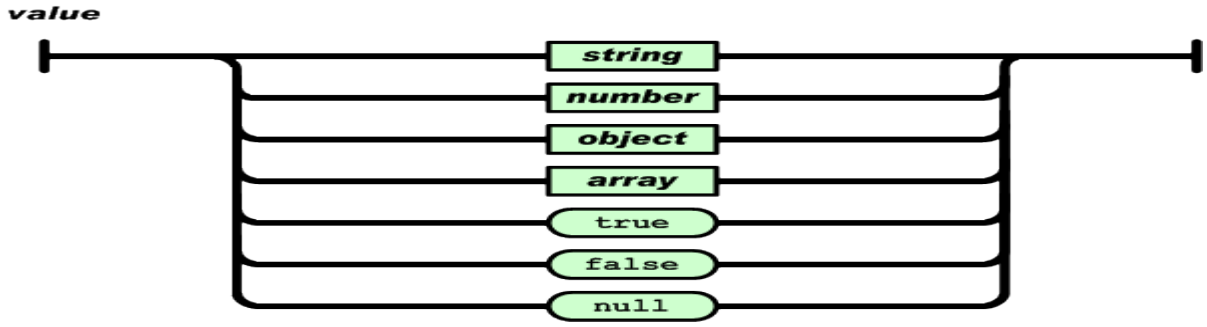


Figura 1.4 - JSON Value

Una ***stringa*** è una raccolta di zero o più caratteri Unicode, tra virgolette; per le sequenze di escape utilizza la barra rovesciata. Un singolo carattere è rappresentato come una stringa di caratteri di lunghezza uno. Una stringa è molto simile a una stringa C o Java.

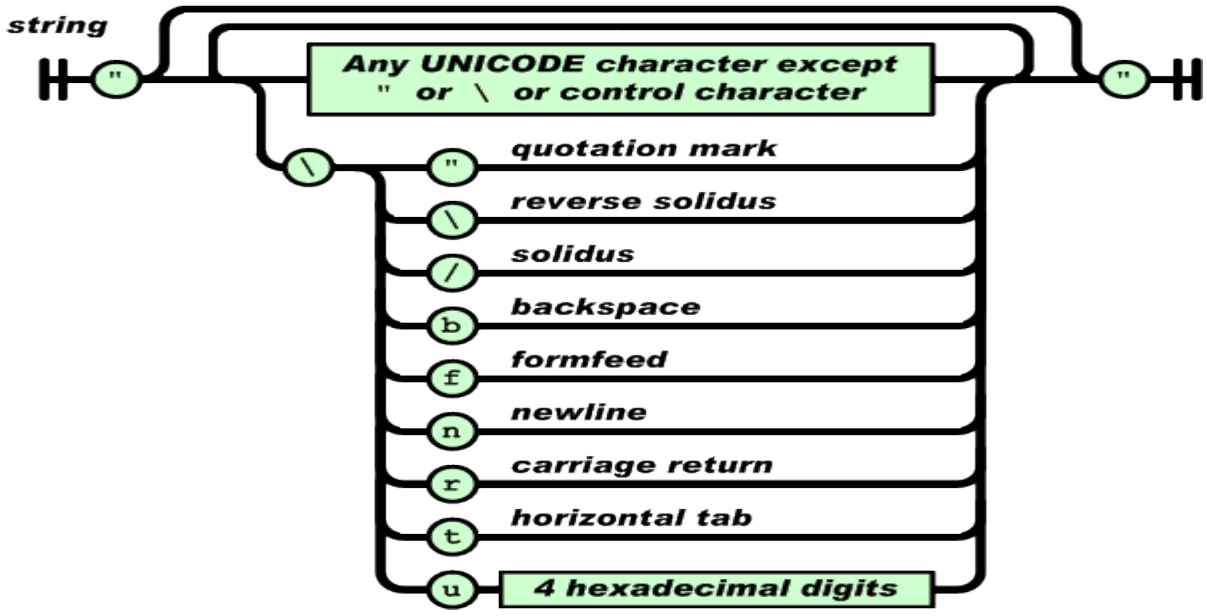


Figura 1.5 - JSON String

Un ***numero*** è molto simile a un numero C o Java, a parte il fatto che i formati ottali e esadecimale non sono utilizzati.

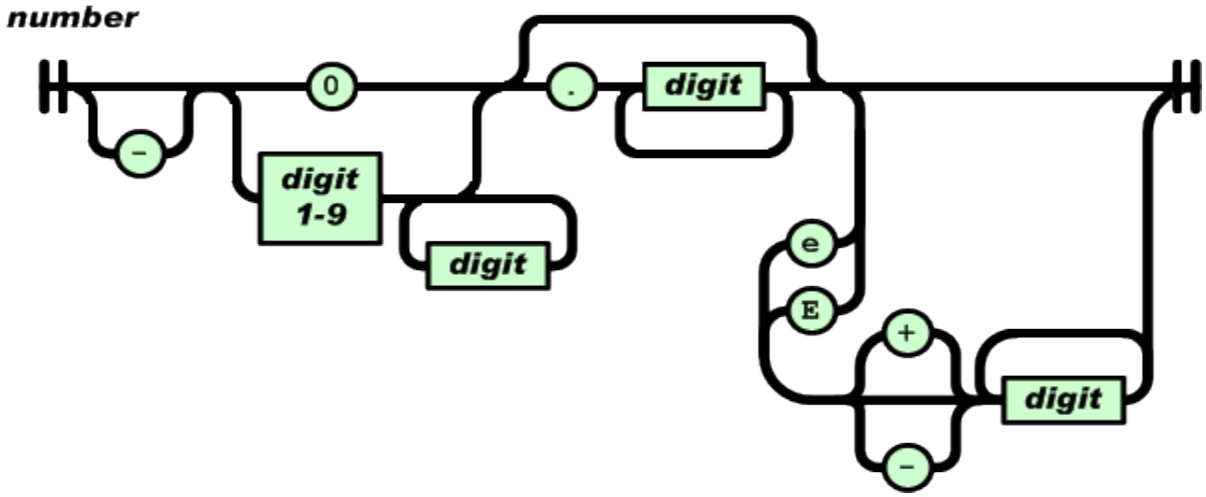


Figura 1.6 - JSON Number

I caratteri di spaziatura possono essere inseriti in mezzo a qualsiasi coppia di token.
Lo scambio di queste informazioni in formato JSON, nel nostro caso, avviene tra un server che mette e a disposizione i web service e un Client realizzato in ambiente Android. [4]

1.3 Il sistema Operativo Android

Android, per le ragioni elencate precedentemente, è il candidato ideale come piattaforma di riferimento per lo sviluppo di applicativi per i nuovi Smartphone. Google ha fornito Android in una soluzione Open Source agevolando i produttori di dispositivi mobili a creare una ricca varietà di dispositivi con prezzi differenti per il pubblico. [5]

Basta pensare che, nel 2010, Android è cresciuto del 886% [6]

Architetturalmente Android è un sistema operativo per dispositivi mobili basato su kernel Linux. Altri componenti open di Android sono:

- OPEN GL –SGI Free Software License
- SGL (Scene Graph Library) – LGPL
- SQLite (Public domain)
- WebKit – LGPL

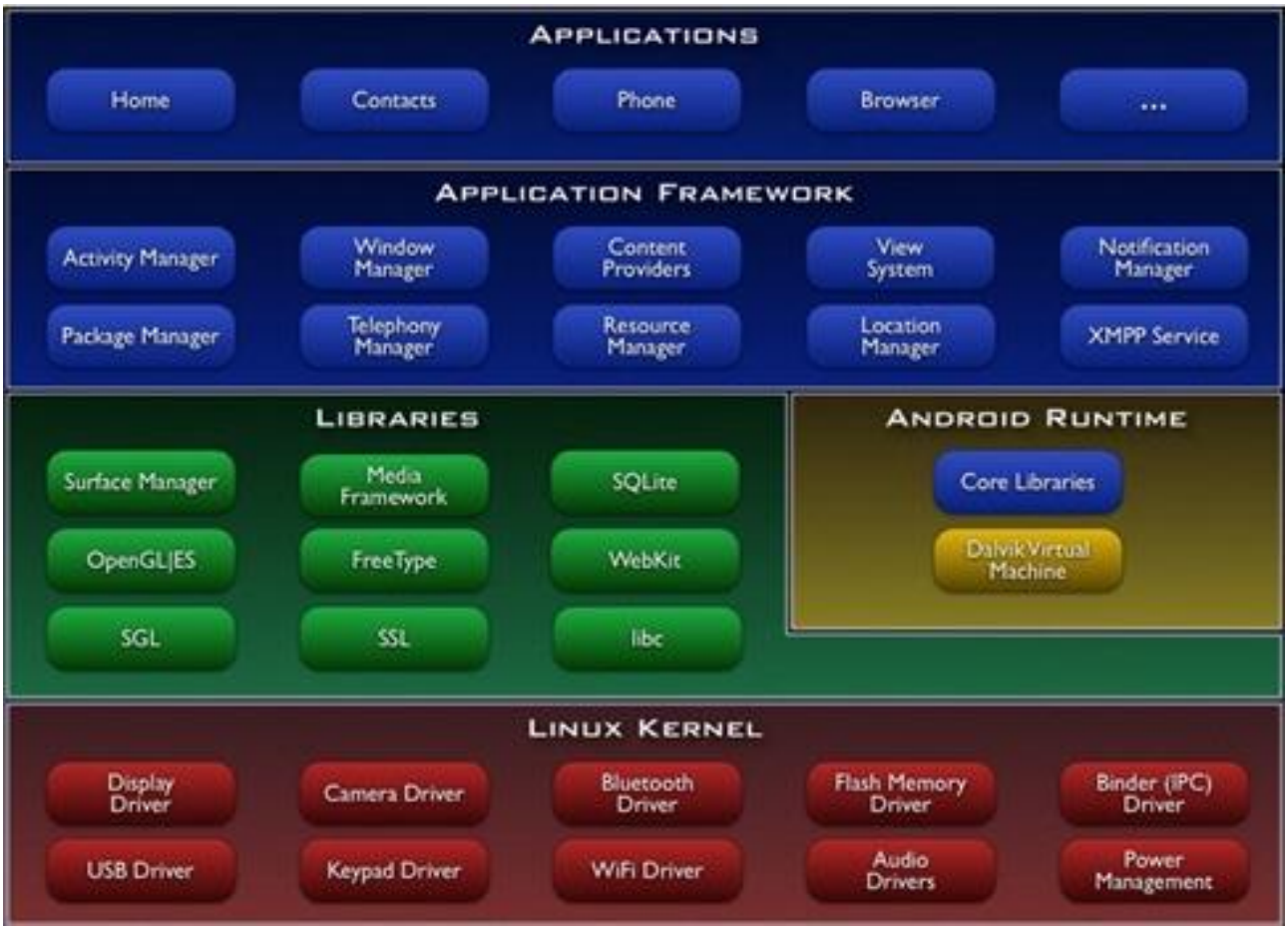


Figura 1.7 – Architettura Android

Un'applicazione Android è trasformata in un codice intermedio (bytecode) che è eseguito da una Virtual Machine Java. [7]

Per gli applicativi Android è stata scritta una nuova Virtual Machine di nome Dalvik Virtual Machine (DVM). Essendo la Virtual Machine uguale per tutti i dispositivi Android, ognuno, indipendentemente dal costruttore può eseguire l'applicativo. La Java Virtual Machine carica il bytecode che si riferisce all'instanza della classe Java che il programma usa. Nell'utilizzo delle componenti dell'interfaccia grafica utente, il sistema, richiamando la UI (User Interface), carica anche il codice relativo a tali eventi grafici e li aggiunge al bytecode in esecuzione.

Per realizzare la nostra applicazione abbiamo utilizzato la piattaforma integrata Eclipse con l'installazione del Development Toolkit (ADT), plug-in Android per Eclipse Classic.

Per il testing dell'applicazione è stato utilizzato un dispostivo Android Lg L9.

Capitolo 2

Client Android

In questo capitolo si andrà ad esporre nel dettaglio la parte dedicata al Client Android, con le relative classi ad esso associate.

Il Client Android è la parte del servizio che si interfaccia direttamente con l'utente, quindi si compone principalmente di una interfaccia grafica, al momento molto semplice, che permette l'invio di dati fondamentali, ma comunque si possono aggiungere tutti i campi necessari per permettere al servizio di essere utilizzato in un qualunque contesto.

Il nostro Client si compone di diverse classi, molte delle quali fondamentali al servizio, come la classe che permette di inviare e ricevere dati dal servizio Rest, la classe gestisce l'interfaccia grafica per la Signup ed infine la classe che permette la verifica del codice di autenticazione (Code Verification) inviato dal nostro Webservice tramite WhatsApp o Telegram.

2.1 Le Activity

Innanzi tutto partiamo con il descrivere il concetto di Activity.

L'Activity è il primo dei quattro componenti basilari che troviamo nelle applicazioni Android. Nel nostro progetto ne troveremo due che permettono all'utente di interagire con la nostra applicazione, la prima permette di registrare i dati e richiedere un codice di verifica, la seconda invece permette di inserire il codice di verifica ottenuto da WhatsApp o Telegram. Per creare un'Activity è necessario fare due cose:

- estendere la classe Activity, appartenente al framework Android;
- registrare l'Activity nell'AndroidManifest.xml mediante l'uso dell'apposito tag XML <activity>. Tra l'altro, questo dettame vale per tutte le quattro componenti fondamentali di un'applicazione.

Le Activities sono organizzate in uno stack, dove l'attività in cima è quella visualizzata in quel momento. La visualizzazione di una nuova screen corrisponde quindi allo start() di una nuova activity che viene conseguentemente posta in cima allo stack.

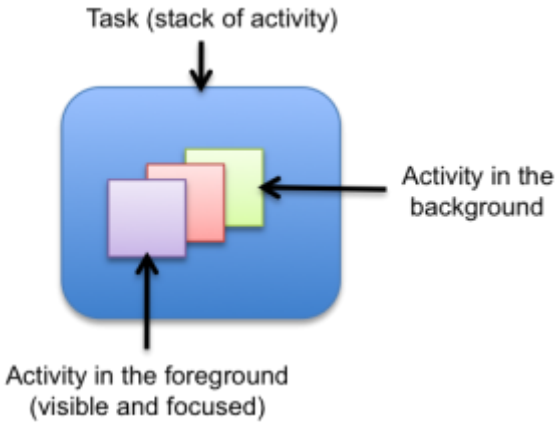


Figura 2.1 - Esempio di Stack delle Activity

L'invocazione di una seconda activity da parte dell'activity attiva avviene attraverso la definizione degli Intent che sono istanze dell'omonima classe del package android.content.

Gli Intent rappresentano una descrizione astratta di una funzione che un'activity richiede di eseguire a un'altra activity.
Con gli Intent sono lanciati non solo le Activity ma anche Content Provider e Services. Un content provider è un contenitore di dati ovvero il modo per un'applicazione Android di condividere informazioni globali con le altre applicazioni.

Essendo Android un sistema Linux-based, ogni APP ha il suo userid, la sua directory "data" (/data/data/nome_package) e un suo spazio protetto di memoria. Per questo motivo gli applicativi Android hanno bisogno dei Content Provider per comunicare tra loro.

I processi possono segnalarsi al sistema come Content Provider di un insieme di dati. Quando le informazioni sono richieste, sono richiamate da Android grazie ad un insieme specifico di API che permettono di agire sul contenuto secondo delle specifiche predefinite. Un esempio di dati che possiamo incorporare in un Content Provider è un database SQLite.
Android è comunque provvisto di un insieme di Content Provider nativi, documentati nel package "android.provider" dell'SDK.

Il codice Java che realizza l’Activity risiede nella cartella *src*, la classe generalmente ha un nome che identifica la schermata che andrà visualizzata sul device, ma comunque dovrà sempre estendere la classe Activity.Al suo interno viene implementato l’override del metodo onCreate, una delle tappe fondamentali del ciclo vitale di una activity. Abbiamo due righe di codice fondamentale che caratterizzano l'override del metodo onCreate e sono:

- super.onCreate(savedInstanceState): invoca il metodo omonimo della classe base. Questa operazione è assolutamente obbligatoria;
- setContentView(R.layout.activity_main): specifica quale sarà il “volto” dell’Activity, il suo layout. Al momento la dicitura R.layout.activity_main può apparire alquanto misteriosa ma lo sarà ancora per poco, fino al momento in cui verrà illustrato l’uso delle risorse. Il suo effetto è quello di imporre come struttura grafica dell’Activity il contenuto del file activity_main.xml presente nella cartella res / layout.

Un'altro componente fondamentale della nostra activity è Il file AndroidManifest.xml che definisce la configurazione generale della nostra applicazione Android.Quello che segue è il manifest del nostro Client Android.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.phoneme"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="19" />

    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" >
    </uses-permission>
    <uses-permission android:name="android.permission.READ_CONTACTS" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="Ruby"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".auth.Splash"
            android:label="" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name="com.phoneme.auth.login.LoginActivity"
            android:label="Login to your Account" >
        </activity>

        <!-- Entry for RegisterActivity.class -->
        <activity
            android:name="com.phoneme.auth.signup.SignupActivity"
            android:label="Register New Account" >
        </activity>
        <activity
            android:name="com.phoneme.auth.signup.CodeVerificationActivity"
            android:label="Verify Your Code" >
        </activity>
        <activity
            android:name=".mobile.MainActivity"
            android:label="@string/app_name" >

            <!-- <intent-filter> -->
            <!-- <action android:name="android.intent.action.MAIN" /> -->
            <!-- <category android:name="android.intent.category.LAUNCHER" /> -->
            <!-- </intent-filter> -->
        </activity>

        <service
            android:name="com.octo.android.robospice.JacksonSpringAndroidSpiceService"
            android:exported="false" />
    </application>

</manifest>
```

Questo , come possiamo vedere nello specifico, è composto da diversi nodi; in particolare i più importanti sono il nodo <application> che contiene le componenti usate nell'applicazione come l'icona e il tema; abbiamo poi il nodo <activity> che con l'attributo android:name specifica il nome della classe Java che incarna l'Activity, mentre android:label indica la scritta che andrà a riempire la label in cima alla nostra schermata dell'activity. Se non viene specificato un package è sottintesa l'appartenenza della classe al package riportato nel nodo <manifest>, la *root* del file.

Abbiamo poi il tag <uses-sdk> che al suo interno contiene le informazioni di identificazione della versione SDK (Software Development Kit) utilizzato per creare la nostra applicazione, abbiamo poi il tag <uses-permission> che permette di inserire diversi permessi di cui godrà la nostra applicazione, nello specifico caso abbiamo che i più importanti sono quello di Internet (android.permission.INTERNET) e quello di leggere la rubrica dell'utente (android.premission.READ_CONTANCTS). [8]

La nostra Activity potrà poi assumere diversi stati durante il suo ciclo vitale.

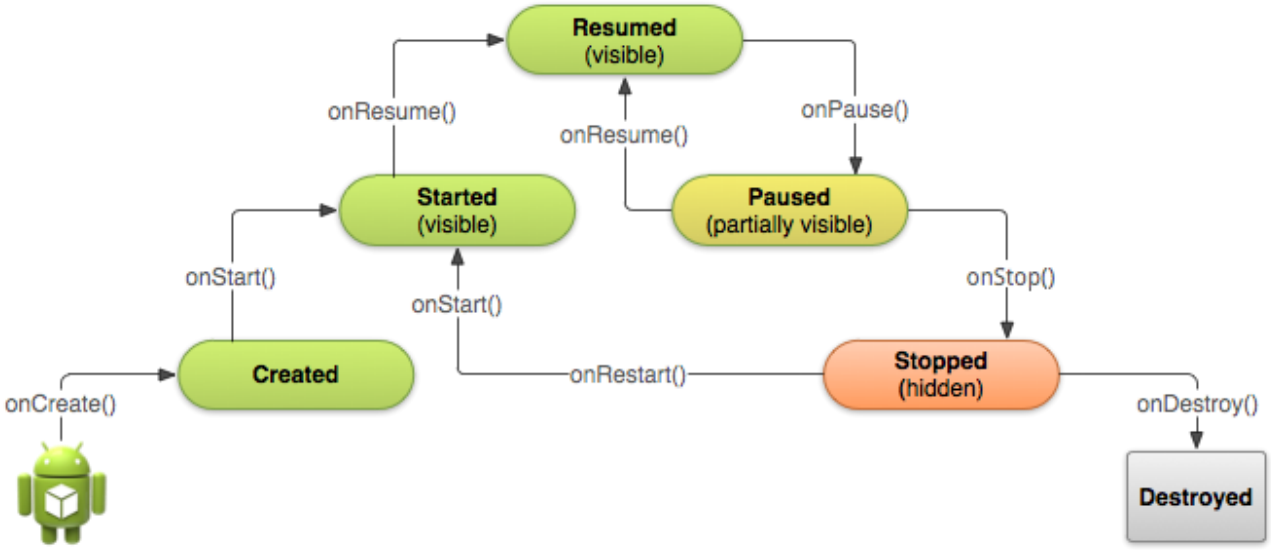


Figura 2.2 - Ciclo di vita di una Activity

Si tratta di una serie di stati attraverso i quali l'esistenza dell'Activity passa. In particolare, nell'illustrazione riportata, gli stati sono rappresentati dalle figure colorate. L'ingresso o l'uscita da uno di questi stati viene notificato con l'invocazione di un metodo di callback da parte del sistema. Il codice inserito in tali metodi dovrà essere allineato con la finalità del metodo stesso.
Quando un'activity va in esecuzione per interagire direttamente con l'utente vengono obbligatoriamente invocati tre metodi:

- **onCreate**: l'activity viene creata. Il programmatore deve assegnare le configurazioni di base e definire quale sarà il layout dell'interfaccia;
- **onStart**: l'activity diventa visibile. È il momento in cui si possono attivare funzionalità e servizi che devono offrire informazioni all'utente;
- **onResume**: l'activity diventa la destinataria di tutti gli input dell'utente.

Android pone a riposo l'activity nel momento in cui l'utente sposta la sua attenzione su un'altra attività del sistema, ad esempio apre un'applicazione diversa, riceve una telefonata o semplicemente – anche nell'ambito della stessa applicazione – viene attivata un'altra Activity. Anche questo percorso, passa per tre metodi di callback:

- **onPause** (l'inverso di onResume) notifica la cessata interazione dell'utente con l'activity;
- **onStop** (contraltare di onStart) segna la fine della visibilità dell'activity;
- **onDestroy** (contrapposto a onCreate) segna la distruzione dell'activity.

Presentato brevemente il funzionamento di un' Activity andiamo a visionare nel dettaglio il codice delle due Activity principali che compongono il nostro Client Android. **[9]**

2.1.1 SIGNUP ACTIVITY

Tralasciando l'importazione delle dovute librerie esaminiamo gli attributi della classe che, naturalmente, estende la classe Android.App.Activity nel dettaglio.

```
String JsonResult;
String numberPrefix=null;
String msgType=null;
```

Dichiarazione di attributi globali, numberPrefix conterrà il prefisso del numero di telefono, msgType invece il tipo di messaggio restituito dal nostro WebService, mentre JsonResult andrà a contenere il risultato Json ottenuto dalla risposta del nostro WebService.

```
final PrefixContainer container = new PrefixContainer();
```

diachiarazione e implementazione di un attributo di tipo PrefixContainer che deriva appunto dalla classe PrefixContainer che contiene una variabile di tipo Map che permette di associare ad ogni prefisso una stringa che andrà ad indicare il luogo del prefisso stesso. Il codice della classe è il seguente:

```
public class PrefixContainer {
            private Map<String, String> prefixMap = null;
            public PrefixContainer(){
                  prefixMap = new HashMap<String, String>();
                  prefixMap.put("Italy", "39");
                  prefixMap.put("Spain", "340");
            }

            public String[] getPrefixList(){
                  List<String> list = new ArrayList<String>();

                  for ( String key : prefixMap.keySet() ) {
                      list.add(key);
                  }

                  String[] array = list.toArray(new String[list.size()]);
                  return array;
            }

            public String getPrefixValue(String prefixName){
                  return (String) prefixMap.get(prefixName); }     }}
```

La classe come si può vedere è composta da due funzioni principali.

getPrefixList() ritorna come risultato della sua esecuzione un' array di stringhe, contenente quindi tutti i luoghi disponibili per la nostra applicazione, ai quali sarà collegato il relativo prefisso; questi poi andranno visualizzati nel nostro Spinner per permettere al nostro utente il suo prefisso di appartenenza.

getPrefixValue(String prefixName) invece permette al programma di recuperare attraverso l'oggetto Map il "valore numerico" effettivo del nostro prefisso, permettendo poi così di comporre il numero telefonico effettivo a cui verrà inviato il relativo messaggio WhatsApp o Telegram.

```
AlertDialogManager alert = new AlertDialogManager();
```

la classe AlertDialogManager è stata creata per semplificare il processo di creazione di un alert-dialog in modo da mantenere il codice il più pulito possibile.

```
public void showAlertDialog(Context context, String title, String
message) {
            AlertDialog alertDialog = new
AlertDialog.Builder(context).create();

            // Setting Dialog Title
            alertDialog.setTitle(title);

            // Setting Dialog Message
            alertDialog.setMessage(message);

            // Setting OK Button
            alertDialog.setButton("OK", new
DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int which) {
                  }
            });

            // Showing Alert Message
            alertDialog.show();
      }
```

Quella scritta sopra è l'unica funzione al momento disponibile per questa classe, che preso in ingresso il Context della nostra Activity, una String Title (titolo) che ne indicherà il titolo, una String message (messaggio) che ne indicherà il messaggio principale ne va a comporre una semplice dialog di allerta personalizzata con un semplice tasto "OK" come unica scelta.

Figura 2.3 - Esempio di AlertDialog

```
// user didn't entered username or password
// Show alert asking him to enter the details
alert.showAlertDialog(
      SignupActivity.this,
      "Signup failed..",
      "Please enter phone number, full name and
password");
```

Questo invece è il suo utilizzo all'interno della SignupActivity, e cioè se dopo una verifica dei campi di testo, che permettono l'inserimento di numero di telefenono, nome , cognome e password, se anche solo uno di questi dovesse risultare vuoto verrebbe mostrato a schermo il messaggio, in modo da avvertire l'utente di riempire completamente i campi prima di procedere.

```
EditText txtPhoneNumber, txtPassword, txtName, txtSurname;

txtPhoneNumber = (EditText)findViewById(R.id.phoneNumber_editText);
txtPassword = (EditText) findViewById(R.id.password_editText);
txtName = (EditText) findViewById(R.id.name_editText);
txtSurname = (EditText) findViewById(R.id.surname_editText);
```

l'oggetto EditText il quale permette l'immissione di testo da parte dell'utente. Quando l'utente clicca su un oggetto di tipo EditText, infatti, appare automaticamente una tastiera virtuale sullo schermo del dispositivo che permette l'immissione del testo.

Come detto la tastiera appare in maniera automatica non appena clicchiamo sull'oggetto EditText, ma è anche necessario farla sparire non appena l'utente termina l'immisione del testo.

```
@Override
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);

	// Session Manager
	session = new SessionManager(getApplicationContext());

	// Set View to register.xml
	setContentView(R.layout.signup_activity);
```

Creazione della classe con definizione del file di layout.

```
TextView loginScreen = (TextView) findViewById(R.id.link_to_login);

	// Listening to Login Screen link
	loginScreen.setOnClickListener(new View.OnClickListener() {

	public void onClick(View arg0) {
		// Closing registration screen
		// Switching to Login Screen/closing register screen
		Intent i = new Intent(getApplicationContext(),
				LoginActivity.class);
		startActivity(i);
		finish();

	}

});
```

In questo caso andiamo a dichiarare ed implementare una semplice TextView di nome loginScreen, quest'ultima infatti implementando il metodo loginScreen.setOnClickListener(new View.OnClickListener() ci permette al suo click di passare dalla schermata della SignupActivity a quella di login.

```
Spinner prefixSpinner = (Spinner) findViewById(R.id.prefix_spinner);
	ArrayAdapter<String> adapterPrefix = new ArrayAdapter<String>(
		this,
		android.R.layout.simple_spinner_item,
		container.getPrefixList()
		);
	prefixSpinner.setAdapter(adapterPrefix);
```

```
      prefixSpinner.setOnItemSelectedListener(new
OnItemSelectedListener() {
       public void onItemSelected(AdapterView<?> adapter, View view,int
pos, long id) {
                String selected =
(String)adapter.getItemAtPosition(pos);
                numberPrefix=container.getPrefixValue(selected);
                Toast.makeText(
                          getApplicationContext(),
                          "il prefisso Ã¨ "+numberPrefix,
                          Toast.LENGTH_LONG
                     ).show();
           }
           public void onNothingSelected(AdapterView<?> arg0) {}
           });
```

Qui abbiamo il codice del primo spinner che come spiegato nelle pagine precedenti prende gli elementi della classe prefixContainer() e li inserisce dentro uno Spinner, questo non è altro l'equivalente dei menu a tendina (select) dell'html.
Si implementa poi il metodo onItemSelected che ad ogni cambiamento dell'elemento selezionato, considerando anche il caricamento dell'activity stessa, fa comparire un Toast, che sarebbe un messaggio che compare a schermo senza interrompere l'attività dell'utente, quest'ultimo conferma e indica in valore numerico effettivo il tipo di prefisso selezionato. Quello che segue invece è lo Spinner dedicato alla selezione del tipo di servizio di messaggistica di cui l'utente vorrà usufruire, al momento quelli disponibili sono due e sono WhatsApp e Telegram, i due più utilizzati servizi di messagistica online collegata ad un numero di cellulare attualmente in uso.
Come prima il nostro Spinner ci offre delle selezioni e poi attraverso l'implementazione del metodo onItemSelected, ad ogni cambiamento della selezione farà comparire un Toast che confermerà il tipo di selezione dell'utente.

```
String[] msgTypeList= {"Whatsapp","Telegram"};
           Spinner msgTypeSpinner = (Spinner)
findViewById(R.id.msgtype_spinner);
           ArrayAdapter<String> adapterMsgType = new
ArrayAdapter<String>(
                this,
                android.R.layout.simple_spinner_item,
                msgTypeList);
```

```
            msgTypeSpinner.setAdapter(adapterMsgType);
            msgTypeSpinner.setOnItemSelectedListener(new
OnItemSelectedListener() {
            public void onItemSelected(AdapterView<?> adapter, View
view,int pos, long id) {
                  msgType = (String)adapter.getItemAtPosition(pos);
                  Toast.makeText(
                              getApplicationContext(),
                              "il metodo scelto Ã¨ "+msgType,
                              Toast.LENGTH_LONG
                        ).show();
            }
            public void onNothingSelected(AdapterView<?> arg0) {}
      });
```

La seguente riga di codice invece permette di selezionare come tipo di tastiera pre-selezionata per la EditText txtPhoneNumber, quindi la sezione dove andiamo ad inserire il nostro numero di telefono una tastiera contenente solo i numeri e non l'alfabeto dato che per la scrittura di un numero di telefono risulta inutile.

```
TelephonyManager tMgr = (TelephonyManager) getApplicationContext()
                        .getSystemService(Context.TELEPHONY_SERVICE);

            txtPhoneNumber.setText(tMgr.getLine1Number());
```

Questo grazie all'oggeto TelephonyManager che fornisce accesso alle informazioni dei servizi del nostro telefono e del telfono stesso. Le applicazioni possono utilizzare i metodi di questa classe per determinare i servizi del device e il suo stato, come anche il tipo di tastiera utilizzata.

Le applicazioni oltrettutto possono anche implementare un listener per ricevere informazioni sul cambio di stato del device.

Questa classe non viene istanziata direttamente infatti, si richiama un riferimento all'istanza attraverso lachiamata Context.getSystemService(Context.TELEPHONY_SERVICE).

Bisogna anche ricordarsi che per accedere ad alcuno informazioni bisogna avere accesso ha tutti i diritti di protezione, questo vuol dire che la nostra applicazione non potrà avere accesso alle informazioni protette fino a che queste non veranno specificate all'interno del manifest.

```
// Register button button
final Button btnSignUp = (Button) findViewById(R.id.btnRegister);
      // Register Button button click event
      btnSignUp.setOnClickListener(new View.OnClickListener() {
      @Override

            public void onClick(View arg0) {
                  // Get username, password from EditText
            final String number = txtPhoneNumber.getText().toString();
                  String password = txtPassword.getText().toString();
                  String name = txtName.getText().toString();
                  String surname = txtSurname.getText().toString();


                  // Check if username, password is filled
                  if (number.trim().length() > 0
                              && password.trim().length() > 0
                              && name.trim().length() > 0
                              && surname.trim().length() > 0) {

final WebServiceTask wst = new WebServiceTask(WebServiceTask.POST_TASK,
SignupActivity.this, "Posting data...","signup",number,numberPrefix);
                        wst.addNameValuePair("number", number);
                        wst.addNameValuePair("name", name);
                        wst.addNameValuePair("surname", surname);
                        wst.addNameValuePair("password", password);
                        wst.addNameValuePair("prefix", numberPrefix);
                        wst.addNameValuePair("msgtype", msgType);
                        // the passed String is the URL we will POST to
                        wst.execute(new String[] { SERVICE_URL });
                     } else {

                        // user didn't entered username or password
                        // Show alert asking him to enter the details
                        alert.showAlertDialog(
                              SignupActivity.this,
                              "Signup failed..",
                  "Please enter phone number, full name and password");
                        }
                  }
            });
```

Nelle linee di codice sovrastanti invece abbiamo la dichiarazione del btnSignUp, ogetto di tipo Button.Quest'ultimo è il bottone che al suo click permette di inviare tutti i dati delle EditText al WebService, infatti come possiamo vedere subito dopo la dichiarazione del onClickListener(), abbiamo la dichiarazione di 4 variabili di appoggio di tipo String che conterranno il valore delle EditText, che poi andranno passati al WebService per essere elaborati.Subito dopo abbiamo un controllo sulla lunghezza delle stringhe appena ottenute se, anche solo una di queste avesse una lunghezza inferiore alla zero (quindi vuota) vorrebbe dire che l'utente ha dimenticato di riempire un campo e quindi non si potrà procedere, verrà quindi inviata una AlertDialog come descritto nelle pagine precedenti. Infine ultima non meno importante, abbiamo la dichiarazione e implementazione del WebServiceTask, oggetto che permette di passare attraverso connessione Http attraverso una richiesta POST, quindi invierà le stringhe ricavate prima sotto forma di parametri all'URL dichiarato all'inizio, durante l'invio dei dati verrà visualizzata a schermo una ProcessDialog, in attesa del completamento dell'operzione.

Il risultato della nostra activity è quindi il seguente una schermata composta da 4 editText il cui primo contenente il numero di telefono sarà affiancato dallo Spinner che ci permetterà di selezionare il prefisso dato dalla nostra nazionalità, quindi avremo subito sotto l'EditText per inserire il nome ed il cognome, subito sotto abbiamo l'EditText della password.

Poi avremo lo spinner per la selezione del tipo di servizio che si vuole utilizzare, subito sotto troviamo il bottone per completare la registrazione, altrimenti se siamo già registrati possiamo passare alla schermata di Login.

Figura 2.4 -Signup Activity

2.1.2 CodeVerificationActivity

Tralasciando l'importazione delle dovute librerie esaminiamo gli attributi della classe che, naturalmente, estende la classe Android.App.Activity nel dettaglio.

```
private static final String SERVICE_URL =
"http://192.168.0.106:8080/Telly/rest/number/state";

      String PHONENUMBER,PREFIX;
```

Dichiarazione delle variabili globali necessarrie per la connessione HTTP al Webservice e per il passaggio dalla SignupActivity delle informazioni sul numero di telefono e il prefisso appenna registrati.

```
      AlertDialogManager alert = new AlertDialogManager();
```

la classe AlertDialogManager è stata creata per semplificare il processo di creazione di un alert-dialog in modo da mantenere il codice il più pulito possibile, la classe come descritto nelle pagine precedenti crea una piccola finestrella di allerta con un messaggio ed un semplice button con la scritta "OK".

```
      EditText txtCode;
      txtCode=(EditText) findViewById(R.id.code_verify);
```

l'oggetto EditText il quale permette l'immissione di testo da parte dell'utente. Quando l'utente clicca su un oggetto di tipo EditText, infatti, appare automaticamente una tastiera virtuale sullo schermo del dispositivo che permette l'immissione del testo.

Come detto la tastiera appare in maniera automatica non appena clicchiamo sull'oggetto EditText, ma è anche necessario farla sparire non appena l'utente termina l'immisione del testo.

```
public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            Bundle extras = getIntent().getExtras();
            if (extras != null) {
                  PHONENUMBER = extras.getString("NUMBER");
                  PREFIX=extras.getString("NUMBERPREFIX");

            }
```

Creazione della classe con definizione del file di layout.

L'oggetto extras invece permette invece di recuperare i dati passati dalla SignupActivity alla CodeVerificationActivity attraverso l'intent, infatti opzionalmente un intent permette di inviare dati addizzionali attraverso una istanza della classe Bundle che possono essere recuperati dall'intent stesso attraverso il metodo getExtras().

Quando infatti si chiama la creazione della suddetta Activity si possono inserire dati aggiuntivi attraverso il Bundle facendo l'overload del metodo putExtra() dell'oggetto intent.

Gli Extras passati sono di tipo key/value. La chiave sarà ovviamente sempre di tipo String, mentre il suo valore può assumere un tipo di dato primitivo (int,float, etc) e anche String.

L'Activity che deve ricevere i dati, in questo caso la CodeVerificationActivity potrà recuperare i dati inviati attraverso la chiamata getIntent().getExtras() recuperando così i dati extra.

```
TextView goBack = (TextView) findViewById(R.id.link_back);


            goBack.setOnClickListener(new View.OnClickListener() {
                  public void onClick(View arg0) {

                        Intent i = new Intent(getApplicationContext(),
                                    SignupActivity.class);
                        startActivity(i);
                        finish();

                  }
            });
```

In questo caso andiamo a dichiarare ed implementare una semplice TextView di nome goBack, quest'ultima infatti implementando il metodo goBack.setOnClickListener(new View.OnClickListener() ci permette al suo click di passare dalla schermata della CodeVerificationActivity a quella della SignupActivity.

```
Button btnVerify = (Button) findViewById(R.id.btnVerify);

            btnVerify.setOnClickListener(new View.OnClickListener() {

                  @Override
                  public void onClick(View arg0) {

                        String code=txtCode.getText().toString();

                        if (code.trim().length() > 0){

                              WebServiceTask wst = new
WebServiceTask(WebServiceTask.POST_TASK, CodeVerificationActivity.this,
"Posting data...","verify",PHONENUMBER,PREFIX);

                              wst.addNameValuePair("number", PHONENUMBER);
                              wst.addNameValuePair("code", code);

                              // the passed String is the URL we will POST
to
                              wst.execute(new String[] { SERVICE_URL });



                        }else{

                              alert.showAlertDialog(
                                          CodeVerificationActivity.this,
                                          "Verification Failed..",
                                          "Please enter a Code",
                                          false);
                        }
                  }
            });
```

Nelle linee di codice sovrastanti invece abbiamo la dichiarazione del btnVerify, ogetto di tipo Button.Quest'ultimo è il bottone che al suo click permette di inviare tutti i dati della EditText al WebService, infatti come possiamo vedere subito dopo la dichiarazione del onClickListener(), abbiamo la variabile code di tipo String che conterrà il valore della EditText, che poi andranno passati al WebService che verificherà se quest'ultimo è corretto o meno.Subito dopo abbiamo un controllo sulla lunghezza della stringha appena ottenuta se questa dovesse avere una lunghezza inferiore alla zero (quindi vuota) vorrebbe dire che l'utente ha dimenticato di riempire il campo e quindi non si potrà procedere, verrà quindi inviata una AlertDialog come descritto nelle pagine precedenti.

Infine ultima non meno importante, abbiamo la dichiarazione e implementazione del WebServiceTask, oggetto che permette di passare attraverso connessione Http attraverso una richiesta POST, quindi invierà la stringha contente il codice di verifica, durante l'invio dei dati verrà visualizzata a schermo una ProcessDialog, in attesa del completamento dell'operzione.

Il risultato della nostra activity è quindi il seguente una schermata composta da una editText nel quale si potrà inserire il codice di verifica, quindi subito sotto, avremo il button per l'invio dei dati al WebService. Altrimenti possiamo tornare alla schermata di SignUp.

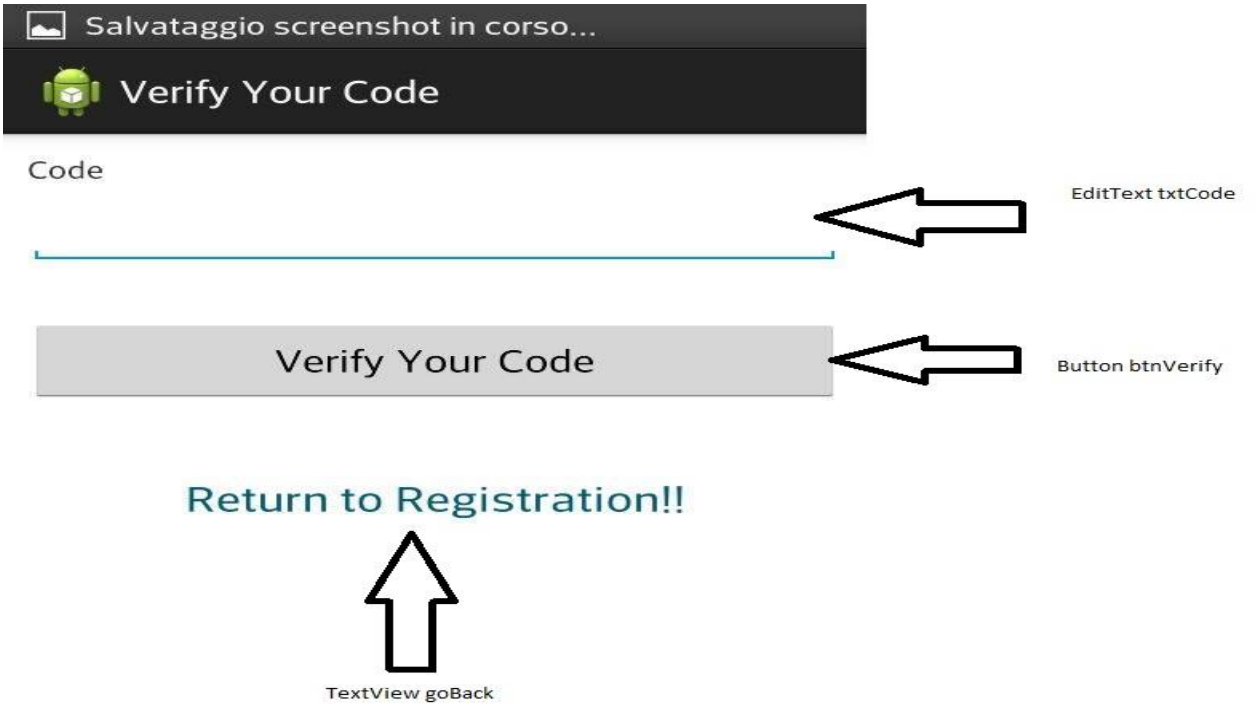


Figura 2.5 - Code Verification Activity

2.2 Classe WebServiceTask

La classe WebServiceTask, estende la classe AsyncTask e permette di inviare dati utilizzando una conessione HTTP attraverso il metodo POST, accettando come risposta una string di tipo Json, ottenuta attraverso metodo GET, durante tutto questo processo verrà visualizzata una ProgressDialog che non permetterà all'utente di eseguire altre operazioni. Andiamo ora ad analizzare le principali funzioni della nostra classe.

```
private String JsonReponse=null;
public static final int POST_TASK = 1;
public static final int GET_TASK = 2;
private static final String TAG = "WebServiceTask";
// connection timeout, in milliseconds (waiting to connect)
private static final int CONN_TIMEOUT = 4000;
// socket timeout, in milliseconds (waiting for data)
private static final int SOCKET_TIMEOUT = 6000;
private ProgressDialog ringProgressDialog;
private int taskType = GET_TASK;
private Activity mContext = null;
private String activityName=null;
private String processMessage = "Processing...";
private String Number,Prefix;
private ArrayList<NameValuePair> params = new ArrayList<NameValuePair>();
```

Queste che possiamo vedere sono le variabili globali della nostra classe, le principali sono:

- **CONN_TIMEOUT**, variabile che tiene in memoria il timeout di conessione massima possibile in millisecondi, quindi il tempo massimo in cui si rimane in attesa di una connessione.
- **SOCKET_TIMEOUT**, variabile che tiene in memoria il timeout di attesa del socket massima possibile in millisecondi, quindi il tempo massimo in cui si rimane in attesa dei dati.

- **params**, variabile che conterrà tutti i parametri che sono stati inseriti dopo aver dichiarato un oggetto di tipo WebServiceTask e che poi dovranno essere inviati al nostro WebService.
- **activityName**, tiene in memoria il nome dell'activity che ha chiamato il WebServiceTask in modo da poter differenziare le operazioni da eseguire nel caso essa sia una SignupActivity o una CodeVerificationActivity.
- **mContext**, contiene l'interfaccia globale per le informazioni sull' enviroment della nostra applicazione. Quest'ultima è una classe astratta la cui implementazione è affidata al sistema Android. Questa ci permette di accedere alla specifiche dell'applicazione, delle risorse e delle classi, come anche lanciare activity e gestire intent.

```
public WebServiceTask(int taskType, Activity mContext, String
processMessage,String activityName, String Number,String Prefix) {

        this.taskType = taskType;
        this.mContext = mContext;
        this.activityName=activityName;
        this.processMessage = processMessage;
        this.Number=Number;
        this.Prefix=Prefix;
    }
```

Questo è il costruttore della nostra classe WebServiceTask, come si può vedere le varibili che dobbiamo fornire alla creazione sono il tipo di Task 1 per POST o 2 per GET, il context della nostra applicazione, il nome della activity che ha richiamato il servizio, il messaggio da mostrare durante l'attesa, il numero di telefono e il prefisso.

```
    public String getJsonResponse(){
      return JsonReponse;
    }
```

Ritorna la stringa Json ottenuta dalla risposta del WebSerice.

```
    public void addNameValuePair(String name, String value) {

        params.add(new BasicNameValuePair(name, value));
    }
```

Questa funzione permette l'aggiunta di una coppia di valori key/value al nostro insieme di parametri che poi dovranno essere inviati attraverso il metodo POST.

```
@Override
    protected void onPreExecute() {

        showProgressDialog();

    }


   private void showProgressDialog() {

     ringProgressDialog=ProgressDialog.show(mContext, "Please Wait!",
processMessage,true,false);

    }
```

Mostra a schermo il ProgressDialog in attesa della conclusione della procedura di invio dei dati, impedendo così all'utente di interferire, magari cliccando su altri Button o modificando i dati inseriti. Il primo valore del nostro ProgressDialog è il context della nostra apllicazione, il secondo è il messaggio visualizzato nel titolo, il terzo è il messaggio visualizzato all'interno del riquadro, poi abbiamo il true che indica che la nostra ProgressDialog sarà intermedia mentre l'ultimo indica che la nostra ProgressDialog non sarà cancellabile dal tocco dell'utente.

```
protected String doInBackground(String... urls) {

        String url = urls[0];
        String result = "";

        HttpResponse response = doResponse(url);

        if (response == null) {
            return result;
        } else {

            try {

                result =
inputStreamToString(response.getEntity().getContent());

            } catch (IllegalStateException e) {
                Log.e(TAG+" DoBackGroundState", e.getLocalizedMessage(),
e);

            } catch (IOException e) {
                Log.e(TAG+" DoBackGroundIO", e.getLocalizedMessage(), e);
            }

        }

        return result;
    }
```

La funzione sopra attiva il processo in background di invio e ricezione dei dati dal WebService, si otterrà un oggetto di tipo HTTPResponse dalla funzione doResponse (che andremo a visionare tra breve) se questo response dovesse essere nullo, ci fermerò qui ritornando un valore nullo.
Se così non fosse trasformeremo il nostro response da stream in una stringa e di conseguenza in un Json.

```
private HttpResponse doResponse(String url) {
        // Use our connection and data timeouts as parameters for our
        // DefaultHttpClient
        HttpClient httpclient = new DefaultHttpClient(getHttpParams());
        HttpResponse response = null;

        try {
            switch (taskType) {
            case POST_TASK:
                HttpPost httppost = new HttpPost(url);
                // Add parameters
                httppost.setEntity(new UrlEncodedFormEntity(params));
                response = httpclient.execute(httppost);
                break;
            case GET_TASK:
                HttpGet httpget = new HttpGet(url);
                response = httpclient.execute(httpget);
                break;
            }
        } catch (Exception e) {
            Log.e(TAG+" doResponse", e.getLocalizedMessage(), e);
        }
        return response;
    }
```

doResponse effettua quello che è il vero e proprio invio dei dati tramite una stream di byte in una connessione di tipo POST accettando una risposta di tipo stream di byte attraverso una connessione di tipo GET.

```
    @Override
    protected void onPostExecute(String response) {
        handleResponse(response);
        ringProgressDialog.dismiss();
        JsonReponse=response;
        if(JsonReponse!=null && activityName.equalsIgnoreCase("signup")){

                Intent i = new
Intent(mContext,CodeVerificationActivity.class);
                i.putExtra("NUMBER", Number);
                i.putExtra("NUMBERPREFIX", Prefix);
                mContext.startActivity(i);
                mContext.finish();
            }

    }
```

La funziona sovrastante viene eseguita al termine di tutte le operazioni richieste, chiudendo come prima cosa il ProgressDialog e poi, se la nostra operazione ha dato successo e quindi siamo riusciti ad inviare i dati ottenendo una risposta di rimando, se l'activity che ci ha richiesto l'invio dei dati era la SignupActivity passiamo alla CodeVerificationActivity passando a questa anche i dati relativi al numero di telefono e del prefessi, necessari poi per autenticare il numero di telefono del nostro utente.

```
// Establish connection and socket (data retrieval) timeouts
    private HttpParams getHttpParams() {

        HttpParams htpp = new BasicHttpParams();

        HttpConnectionParams.setConnectionTimeout(htpp, CONN_TIMEOUT);
        HttpConnectionParams.setSoTimeout(htpp, SOCKET_TIMEOUT);

        return htpp;
    }
```

La funzione getHttpParams permette di impostare i parametri base necessari per la conessione HTTP con il nostro WebService, due di questi sono impostati come variabili globali e sono il tempo di attese della conessione e il tempo di attesa del socket.

```
private String inputStreamToString(InputStream is) {
        String line = "";
        StringBuilder total = new StringBuilder();

        // Wrap a BufferedReader around the InputStream
        BufferedReader rd = new BufferedReader(new
InputStreamReader(is));

        try {
            // Read response until the end
            while ((line = rd.readLine()) != null) {
                total.append(line);
            }
        } catch (IOException e) {
            Log.e(TAG+" inputStreamToString", e.getLocalizedMessage(),
e);
        }

        // Return full string
        return total.toString();
    }
```

La funzione inputStreamToString prende come input lo stream ottunuto dalla connessione HTTP con il WebService e lo trasforma in una stringa di caratteri che formano la risposta in JSON.

```
public void handleResponse(String response) {

      try {
      System.out.println("Handle response "+response);

         JSONObject jso = new JSONObject(response);

            String responseId = jso.getString("code");
             String responseMsg = jso.getString("message");


             Toast.makeText(
                       mContext,
                       responseId+" : "+responseMsg,
                       Toast.LENGTH_LONG
                   ).show();


    } catch (Exception e) {
        Log.e("Error", e.getLocalizedMessage(), e);

        Toast.makeText(
                mContext,
                "Server Not Responding or Wrong Response!!",
                Toast.LENGTH_LONG
           ).show();
    }

    }
```

La funzione handleResponse invece prende in input la stringa convertita grazie alla funzione inputStreamToString di conseguenza se questa non è vuota ed è del formato giusto, viene convertita dal JSON ottenendo i valori richiesti, quindi mostra un Toast con la risposta ottenuta.

Capitolo 3

WebService REST

Come abbiamo detto nella descrizione delle tecnologie utilizzate il nostro WebService sarà un servizio di tipo REST, implementato utilizzando Tomcat 7.0;
quest'ultimo interagisce con il device attraverso una connessione HTTP e attraverso appositi URL vengono richiamate delle API appositamente implementate.
Inviando dati attraverso il metodo POST con una connessione HTTP inviamo risorse che poi il nostro WebService utilizzerà per inviare il messaggio di WhatsApp o Telegram all'utente, o per confermare il corretto inserimento del codice inviato al determinato numero di telefono.
Per fare questo utilizzeremo le librerie di Java JSR 311meglio conosciute come “jersey”.
Le API Jersey possono semplificare significativamente lo sviluppo e l'implementazione di un WebService RESTful.

Il nostro servizio si comporrà di diverse classi, e di diverse possibili chiamate a cui questo protrà rispondere.

3.1 Number_Resource

Questa classe è il corpo principale del nostro WebService e contiene tutte le possibili chiamate alle API attivabili dal nostro Device.
Andiamo ora a vedere nel dettaglio il codice che la compone, come sempre mostreremo solo il codice principale tralasciando le importazioni.

```
@Path("/number")
public class Number_Resource {
```

la dichiarazione della classe con @Path sopra indica che le nostre chiamate avranno tutte come padre /number nel nostro specifico caso durante lo sviluppo, siccome il mio "progetto" l'ho denominato Telly avremo che la path completa sarà 127.0.0.1/Telly/rest/number.

```
    private final static String NUMBER = "number";
    private final static String NAME = "name";
    private final static String SURNAME = "surname";
    private final static String PASSWORD = "password";
    private final static String CODE = "code";
    private final static String PREFIX = "prefix";
    private final static String MSGTYPE = "msgtype";
```

Queste sono le costanti che tengono in memoria il nome chiave per reperire i dati inviati tramite post dai parametri di tipo key/value.

```
     // Basic "is the service running" test
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String respondAsReady() {
        return "Telly REST service is ready!";
    }
```

Questa funzione semplicemente ritorna una stringa con la scritta "Telly REST service is ready!" quando si richiama la path Telly/rest/number, in modo da verificare il corretto avviamento del servizio REST.

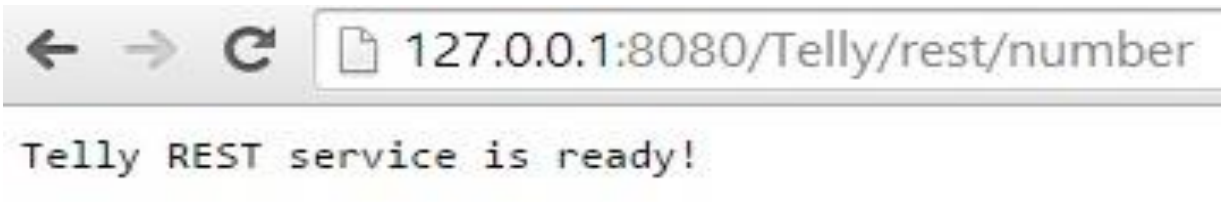


Figura 3.1 - Stampa REST /rest/number

```
    @GET
    @Path("/sample")
    @Produces(MediaType.APPLICATION_JSON)
    public Number getSampleNumber() throws IOException,
InterruptedException {

      Number number = new Number("3485929951","Alessandro",
"Brizzi","alexis91","2546","Pending","39","Whatsapp");

        System.out.println("Returning sample person: " + number.getName()
+ " " + number.getSurname()+ " " + number.getState());

        return number;
    }
```

Qui invece abbiamo una chiamata che risponde alla path Telly/rest/number/sample il quale restituisce una stringa in formato JSON contenente un prototipo di quello che saranno poi i dati utente necessari per registrarlo all'interno del Database, i dati vengono inseriti all'interno di un oggetto di tipo Number che conterrà appunto il numero di telefono, il nome,cognome,password,codice di verifica del numero, lo stato che potrà essere Pending o Verified, il prefisso ed infine il tipo di servizio scelto per inviare il codice di verifica.

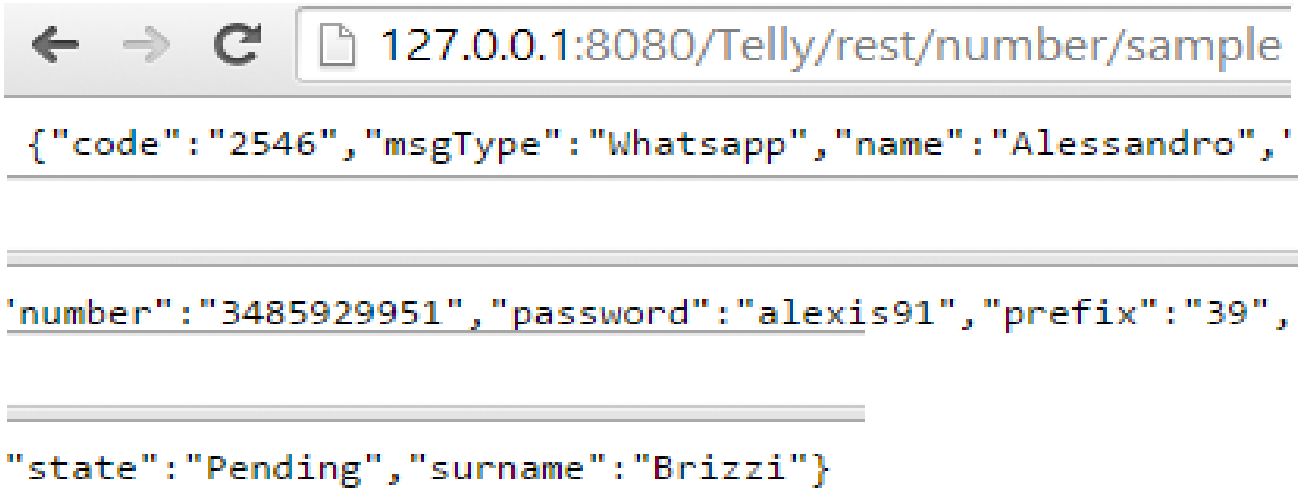


Figura 3.2 - Stampa JSON ottunuto dalla chiamata /rest/number/sample del REST

```
@GET
    @Path("/state/{num}")
    @Produces(MediaType.APPLICATION_JSON)
    public ResponseMessage getNumberState(@PathParam("num") String num)
throws IOException, InterruptedException {

      DbConnection connection=new
DbConnection("jdbc:sqlite:C:/Users/AlessandroBrizzi/Desktop/Telly/dbTelly
");

      ResponseMessage msg = connection.VerifyCode(num);

      connection.Close();

         return msg;
    }
```

Questa chiamata è di tipo variabile questo perchè l'ultima parte dell' URL varia in base all'inserimento da parte dell'utente, infatti nell'ultima parte della nostra path possiamo inserire un qualsiasi numero di telefono per verificarne lo stato di conseguenza avremo che l'ultima parte della path definita come {num} verrà passata come @PathParam("num") String num, quindi il suo valore verrà inserito all'interno di una stringa.

Ffatto questo attraverso la classe DbConnection che vedremo successivamente andiamo a verificare lo stato di verifica relativo al numero desiderato, di conseguenza ritorna un JSON contenente un ogetto di tipo ResponseMessage che indicherà il risultato della query.

```
@POST
    @Path("/registration")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Produces(MediaType.APPLICATION_JSON)
    public ResponseMessage postNumber(MultivaluedMap<String, String>
numberParams) {

        String phonenumber = numberParams.getFirst(NUMBER);
        String name = numberParams.getFirst(NAME);
        String surname = numberParams.getFirst(SURNAME);
        String password = numberParams.getFirst(PASSWORD);
        String prefix = numberParams.getFirst(PREFIX);
        CodeGenerator gen=new CodeGenerator(phonenumber,name+surname);
        String code=gen.getCode();
        String state="Pending";
        String msgtype = numberParams.getFirst(MSGTYPE);
```

Qui sopra invece abbiamo la prima parte di una delle chiamate principali delle nostro servizio REST, infatti questa chiamata serve per registrare un nuovo utente, qui abbiamo la dichiarazione e il recupero dei parametri necessari dalla connessione HTTP.

```
        System.out.println("Storing posted " + phonenumber + " " + name +
" " + surname + "   " + password+" "+code+" "+state+" "+prefix+"
"+msgtype);

        Number number = new
Number(phonenumber,name,surname,password,code,state,prefix,msgtype);

        System.out.println("person info: " + number.getName()+ " " +
number.getSurname() + " " + number.getPassword() + " " +
number.getNumber()+" "+number.getCode()+" "+number.getState()+"
"+number.getMsgType());
```

Qui stampiamo a console le info ottenute dai parametri inviati dal nostro Device, anche sotto forma di oggetto Number per essere sicuri di averli organizzati nel modo corretto.

```
      DbConnection connection=new
DbConnection("jdbc:sqlite:C:/Users/AlessandroBrizzi/Desktop/Telly/dbTelly
");
```

Creiamo un oggetto di tipo DbConnection che è una classe che andremo ad analizzare nel dettaglio nelle prossime pagine e che permette di aprire una connessione con il nostro Database per l'inserimento delle informazioni.

```
        int success=-1;
```

La variabile success di tipo int permette di identificare il tipo di messaggio che andrà restituito in base al tipo di problema riscontrato oppure sul corretto funzionamento.

```
            boolean msgsent=true;

            if(msgtype.toLowerCase().trim().contains("whatsapp")){

                  YowsupClass whatsappmessage = new
YowsupClass(prefix+phonenumber, code);
                 msgsent=whatsappmessage.sendWhatsappMessage();

            }
            if(msgtype.toLowerCase().trim().contains("telegram")){

                  TelegramClass telegrammessage = new
TelegramClass(prefix, phonenumber, code);
                  msgsent=telegrammessage.sendTelegramMessage();

            }
```

Qui andiamo a evidenziare due casi differenti in base al tipo di servizio di messaggistica scelto dall'utente.

Nel primo caso abbiamo un oggetto di tipo YousupClass che prende in ingresso il numero compreso il prefessi del nostro utente e il codice di verifica per inviare un messaggio di WhatsApp al numero indicato attraverso le librerie Yousup che andremo a vedere nei capitoli seguenti.

Nel secondo caso invece abbiamo un oggetto di tipo TelegramClass che prende in ingresso il prefisso e il numero di telefono, infine il codice di verifica per inviare un messaggio di Telegram al numero indicato attraverso i comandi del Telegram-cli che andremo a vedere nei capitoli seguenti.

```
            if(msgsent!=false){
                  success=connection.InsertNumber(phonenumber,
name,surname, password, code, state,prefix,msgtype);
                  if(success>1){
                  System.out.println("Number Inserted Correctly");
                  }else
                  {
                  System.out.println("Errore durante inserimento");
                  }
            }else{
                  success=2;
            }
```

A questo punto verifichiamo che il messaggio sia stato inviato senza errori, se è così andiamo ad inserire il numero all'interno del Database e stampiamo a console un messaggio di avvenuto inserimento, o di errore se non siamo riusciti ad inserire i dati nel Database.

```
        connection.Close();
        String id=null;
        String mess=null;
        switch(success){
           case 0:
                  id="403";
                  mess="Db ERROR!!";
                  break;
           case 1:
                  id="400";
                  mess="You are just registered!";
                  break;
           case 2:
                  id="405";
                  mess="Can't send message!!!";;
                  break;
           case 3:
                  id="406";
                  mess="Inserted, message sent";;
                  break;
        }
        ResponseMessage msg= new ResponseMessage(id, mess);
        System.out.println(msg.getCode()+" : "+msg.getMessage());
        return msg;
    }
```

Infine chiudiamo la connessione aperta all'inizio e in base al risultato ottenuto inserito nel variabile success, andiamo a creare un oggetto di tipo ResponseMessage che poi andrà restituito al Device.

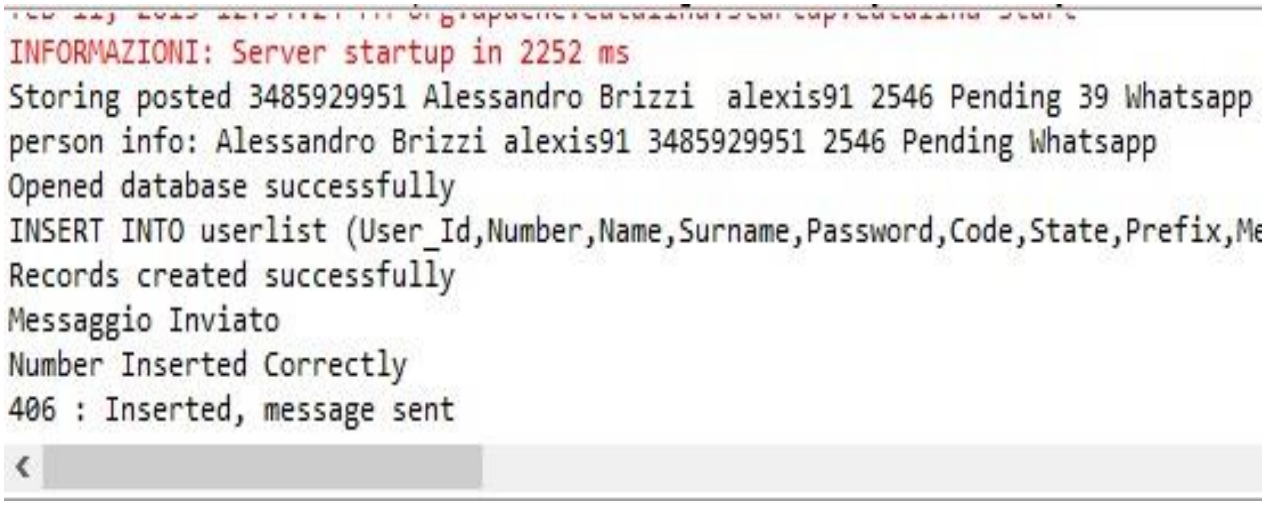


Figura 3.3 - Console Eclipse REST dopo aver richiesto una regitrazione

```
    @POST
    @Path("/state")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Produces(MediaType.APPLICATION_JSON)
    public ResponseMessage postNumberStateUpdate(MultivaluedMap<String,
String> numberParams) {
      System.out.println("Verifico Stato");
      String phonenumber=numberParams.getFirst(NUMBER);
      String code=numberParams.getFirst(CODE);

      DbConnection connection=new
DbConnection("jdbc:sqlite:C:/Users/AlessandroBrizzi/Desktop/Telly/dbTelly
");

      connection.UpdateNumberState(code, phonenumber);
      ResponseMessage msg = connection.VerifyCode(phonenumber);
      System.out.println("Messsage: "+msg.getCode()+"
"+msg.getMessage());
      connection.Close();
      return msg;
    }
```

Ultima ma non meno importante chiamata dato che permette al Device di verificare lo stato di iscrizione e nel caso il codice sia giusto di fare l'upload dello stato del numero di telefono sul Database da Pending a Verified.

3.2 DbConnection

Questa classe è la classe che contiene tutte le funzioni necessarie per accedere al Database in SQLite.

SQLite è una libreria software scritta in linguaggio C che implementa un DBMS SQL di tipo ACID incorporabile all'interno di applicazioni. Il suo creatore, D.

Richard Hipp, lo ha rilasciato nel pubblico dominio, rendendolo utilizzabile quindi senza alcuna restrizione. Permette di creare una base di dati (comprese tabelle, query, form, report) incorporata in un unico file, come nel caso dei moduli Access di Microsoft Office e Base di OpenOffice.org e Libre Office; analogamente a prodotti specifici come Paradox o Filemaker.

SQLite non è un processo standalone utilizzabile di per sé, ma può essere incorporato all'interno di un altro programma.

Analizziamo ora nel dettaglio la classe.

```
public DbConnection(String dbpath){

           try {
                Class.forName("org.sqlite.JDBC");
                c = DriverManager.getConnection(dbpath);
                c.setAutoCommit(false);


                System.out.println("Opened database successfully");

              } catch ( Exception e ) {
                System.err.println( e.getClass().getName() + ": " +
e.getMessage() );
                System.exit(0);
              }
      }
```

Questo è il costruttore della nostra classe e permette di aprire una connessione con il nostro database in SQLite.

```
public int InsertNumber(String phonenumber,String name, String
surname,String password,String code,String state,String prefix,String
messagetype){
                String sql = "INSERT INTO userlist
(User_Id,Number,Name,Surname,Password,Code,State,Prefix,Message_Type) " +
                             "VALUES ( null , '"+phonenumber+"',
'"+name+"', '"+surname+"', '"+password+"', '"+code+"', '"+state+"',
'"+prefix+"', '"+messagetype+"' );";
                System.out.println(sql);
                try {
                   Statement stmt = c.createStatement();
                   stmt.executeUpdate(sql);
                   stmt.close();
                   c.commit();
                } catch ( Exception e ) {
                   System.err.println( e.getClass().getName() + ":"
+e.getMessage());
                   if(e.getMessage().contains("is not unique"))
                   {
                         return 1;
                   }else{
                         return 0;
                   }
                }
          System.out.println("Records created successfully");
            return 3;
      }
```

Permette di inserire un nuovo utente all'interno del nostro database, come output abbiamo un intero che potrà assumere 3 possibili valori, 1 se il nostro utente è già presente, 0 per un errore generale, o 3 se il nostro nuovo utente è stato inserito correttamente.

```
public boolean JustRegistered(String Phonenumber){
            try {
                  Statement stmt = c.createStatement();
                  ResultSet rs = stmt.executeQuery( "SELECT * FROM
userlist WHERE Number='"+Phonenumber+"';");
                  if(rs.next()){
                        while ( rs.previous() ) {
                              int id = rs.getInt("User_Id");
                        String  name = rs.getString("Fullname");
                        String  state = rs.getString("State");
                        String  code = rs.getString("Code");
                        System.out.println( "ID = " + id );
                        System.out.println( "NAME = " + name );
                        System.out.println( "State = " + state );
                        System.out.println( "Code = " + code );
```

```
                    System.out.println();
                    }
                    rs.close();
                    stmt.close();
                    return true;
                }

        } catch ( Exception e ) {
          System.err.println( e.getClass().getName() + ": " +
e.getMessage() );
        }
            return false;
      }
```

La funzione JustRegistered prende in input un numero di telefono e verifica se questo è già presente nel database, se così fosse ritornerà un valore true, altrimenti false.

```
      public ResponseMessage VerifyCode(String num){
            ResponseMessage msg = null;
            String st="";
        try {
            Statement stmt = c.createStatement();
          ResultSet rs = stmt.executeQuery( "SELECT * FROM userlist WHERE
Number='"+num+"';");
          while ( rs.next() ) {
             int id = rs.getInt("User_Id");
             String  name = rs.getString("Name");
             String surname = rs.getString("Surname");
             String  state = rs.getString("State");
             String  code = rs.getString("Code");
             System.out.println( "ID = " + id );
             System.out.println( "NAME = " + name );
             System.out.println( "SURNAME = " + surname );
             st=state;
             System.out.println( "State = " + state );
             System.out.println( "Code = " + code );
             System.out.println();
          }
          rs.close();
          stmt.close();
        } catch ( Exception e ) {
          System.err.println( e.getClass().getName() + ": " + e.getMessage() );
          msg=new ResponseMessage("404","Error");
          return msg;
        }
        System.out.println("Operation done successfully");
        if(st=="Verified"){
            msg=new ResponseMessage("401",st);
        }else{
            msg=new ResponseMessage("402",st);
        }
        return msg;     }
```

Questa funzione permette di recuperare lo stato del relativo al numero di telefono e lo ritorna sotto forma di ResponseMessage che poi sarà visualizzato dall'utente attraverso un Toast per confermare l'avvenuta conferma.

```
public boolean UpdateNumberState(String verifycode,String phoneNumber){
            String  code=null;
            try {
                  Statement stmt = c.createStatement();
                 ResultSet rs = stmt.executeQuery( "SELECT * FROM userlist
WHERE Number='"+phoneNumber+"';");
                 while ( rs.next() ) {
                    int id = rs.getInt("User_Id");
                    String  name = rs.getString("Name");
                    String surname=rs.getString("Surname");
                    String  state = rs.getString("State");
                    code = rs.getString("Code");
                    System.out.println( "ID = " + id );
                    System.out.println( "NAME = " + name );
                    System.out.println( "SURNAME = " + surname );
                    System.out.println( "State = " + state );
                    System.out.println( "Code = " + code );
                    System.out.println();
                 }
                 rs.close();
                 stmt.close();
               if(code.equalsIgnoreCase(verifycode)){
                    System.out.println( "Codice Corretto!!!" + verifycode );
                    stmt = c.createStatement();
                    String sql = "UPDATE userlist set State = 'Verified'
where Number='"+phoneNumber+"';";
                    stmt.executeUpdate(sql);
                    c.commit();
               }else{
                    System.out.println( "Codice Sbagliato!!!" + verifycode
);
               }
             } catch ( Exception e ) {
               System.err.println( e.getClass().getName() + ": " +
e.getMessage() );
               return false;
            }
            return true;
      }
```

Questa funzione invece permette di fare l'update dello stato relativo alla registrazione di un determinato numero di telefono, se e solo se il codice inserito dall'utente coincide con quello già presente nel Database.

```
	public boolean Close(){

		try {
				c.close();
		  } catch ( Exception e ) {
				System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
				 return false;
		  }

		return true;
	}
```

Questa funzione invece permette di chiudere la connessione al database e ritorna un boolean che permette di comprendere se questa operazione è stata conclusa con successo o meno.

3.3 TelegramClass e WhatsAppClass

Queste due classi, prendono come input i valori necessari per inviare un messaggio del suddetto servizio, quindi ci serviranno il prefisso e il numero di telefono per comporre così l'effettivo numero del destinatario e il codice di verifica da inviare al dispositivo del nostro utente.

Le due classi quindi si assomigliano molto a parte per il fatto che richiamano due script totalmente diversi che fanno riferimento ovviamente a due tipologie di comandi totalmente diversi in base al servizio scelto dall'utente.
Infatti le librerie che andiamo ad utilizzare e che andremo a descrivere nel dettaglio nei prossimi capitoli, permettono di inviare messaggi di WhatsApp o Telegram attraverso specifici comandi dalla shell Unix.

Quindi per richiamare questi comandi sono stati creati 2 script differenti in base al tipo di messaggio richiesto, ai quali vengono dati in input il numero del nostro utente (compreso di prefisso) e il codice da inviare come testo, quindi le due classi del REST richiamano gli script che andranno a richiamare i comandi necessari all'invio del messaggio.

Andiamo ora ad esaminare nel dettaglio la funzione principale per ogni classe che permette questo operazione.

```
public boolean sendTelegramMessage(){
           String message="Your_Activation_code_is_"+code;
           System.out.println(message);
        try {
           String
command="/home/alessandro/Telegram/bin/SendMessageTelegram.h "+ prefix +"
"+number+" "+message;
           System.out.println(command);
            Process p = Runtime.getRuntime().exec(command);
            // you can pass the system command or a script to exec
command.
            BufferedReader stdInput = new BufferedReader(new
                    InputStreamReader(p.getInputStream()));
            BufferedReader stdError = new BufferedReader(new
                    InputStreamReader(p.getErrorStream()));
            // read the output from the command
            String s="";
            while ((s = stdInput.readLine()) != null) {
                System.out.println("Std OUT: "+s);
            }
            while ((s = stdError.readLine()) != null) {
                System.out.println("Std ERROR : "+s);
                return false;
            }
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Errore invio Whatsapp: "+e.getMessage());
            return false;
        }
        System.out.println("Messaggio Inviato");
        return true;
      }
```

Questa che vediamo sopra è la funzione della classe TelegramClass che permette il richiamo del suo script, tutto avviene attraverso la chiamata.
Runtime.getRuntime().exec(command) che permette eseguire uno specifico comando e i suoi argomenti in un processo separato con uno specifico ambiente e una specifica directory. Nel nostro caso passiamo al nostro exec solo la stringa contenente il comandi da eseguire, che poi verrà eseguito da un processo separato.
Il metodo controlla anche che il comando sia una valido comando di sistema, e i comandi validi dipendono solo dal sistema stesso.
Nel nostro caso siccome non inseriamo nessuno ambiente in cui eseguire il nostro comando il processo semplicemente eredita le proprietà che gli servono dal processo che lo ha invocato.

Infine dopo aver eseguito il comando implementiamo due buffer reader che ci serviranno per andare a recuperare i comandi che stanno venendo eseguiti in background a shell e gli eventuali errori, in modo da poter controllare che sia stato tutto eseguito correttamente.

In ogni caso la funzione ritorna un valore True se il processo è andato a buon fine o False se si sono registrati degli errori.

```
public boolean sendWhatsappMessage(){
           String message="Your_Activation_code_is_"+code;
        try {
            Process p =
Runtime.getRuntime().exec("/home/alessandro/Yowsup/src/sendMessage.sh "+
yowsupPhoneNumber+" "+message);
            // you can pass the system command or a script to exec
command.
            BufferedReader stdInput = new BufferedReader(new
                    InputStreamReader(p.getInputStream()));
            BufferedReader stdError = new BufferedReader(new
                    InputStreamReader(p.getErrorStream()));
            // read the output from the command
            String s="";
            while ((s = stdInput.readLine()) != null) {
                System.out.println("Std OUT: "+s);
            }
            while ((s = stdError.readLine()) != null) {
                System.out.println("Std ERROR : "+s);
                return false;
            }
        } catch (IOException e) {
            e.printStackTrace();
            System.out.println("Errore invio Whatsapp: "+e.getMessage());
            return false;
        }
        System.out.println("Messaggio Inviato");
        return true;
    }
```

Questa che vediamo sopra invece è la funzione principale all'interno della WhatsAppClass e come possiamo vedere è praticamente identica alla funzione descritta precedentemente solo per il fatto che exec questa volta esegue uno script totalmente diverso che vuole in input anche delle variabili un poco diverse dato che il numero di telefono e il prefisso in questo caso sono uniti mentre per quello di Telegram dovevano essere separati.

Il risultato comunque sarà sempre quello di Figura 11 che come si può vedere dall'ultima riga dice anche se il messaggio è stato spedito o meno.

Capitolo 4

Client WhatsApp e Telegram

Innanzi tutto iniziamo con il dire che cosa sono WhatsApp e Telegram.

WhatsApp e Telegram sono delle app di messaggistica mobile multi-piattaforma che consentono di scambiarsi messaggi coi propri contatti senza dover pagare gli SMS. WhatsApp e Telegram sono disponibile per iPhone, BlackBerry, Android, Windows Phone e Nokia.
Dato che entrambi si servono dello stesso piano dati Internet usato per le e-mail e la navigazione web, non vi sono costi aggiuntivi per mandare messaggi e restare in contatto coi propri contatti.

Oltre alla messaggistica di base gli utenti di WhatsApp possono creare gruppi, scambiarsi messaggi illimitati, video e messaggi audio multimediali, l'unica differenza è che Telegram offre un servizio che permette di creare gruppi di chat contenenti fino a 200 persone mentre WhatsApp fino ad un massimo di 50, Telegram offre anche un servizio di crittazione dei dati che rende più sicure le conversazioni.

Andiamo ora a vedere nel dettaglio i due client che offrono una serie di comandi eseguibili a shell e che implementano le API di WhatsApp e Telegram per l'invio dei messaggi, entrambi sono client che implementano librerie utilizzabili solo su sistemi Unix (Ubuntu, Fedora, etc).

Per poter utilizzare questi servizi bisogna avere un numero per il mittente valido, visto che andrà autenticato tramite SMS, nel mio caso ho utilizzato il mio numero persona di cellulare, si noti che i servizi elencati sopra possono essere attivi solo su un device alla volta, quindi una volta registrato il numero per il nostro servizio non potrà essere utilizzato in nessun'altro device.

4.1 Yowsup-Cli

Il client utilizzato per inviare messaggi da shell UNIX di WhatsApp si chiama Yousup-cli ed è un programma a linea di comando in python ed implementa le API di WhatsApp, permette all'utente di accedere e usare WhatsApp, offrendo tutti i possibili servizi di un client WhatsApp standard.

Per poter utilizzare i comandi di Yowsup bisognerà impostare un file contente le configurazioni, che contiene le credenziali di login.
Questo file si trova nel nostro caso sarà /home/alessandro/Yowsup/src/config.example, che sarà poi passato al nostro comando attraverso l'opzione '-c '.

Il protocollo di WhatsApp è basato su una versione modificata di XMPP (Extensible Messaging and Presence Protocol), che non è altro che un insieme di protocolli aperti di messaggistica istantanea e presenza basato su XML. Il client si serve di una combinazione di JID (numero di telefono compreso il prefisso) e una password per il login. Il client di Yowsup è in grado di generare la stessa password che genera il client WhatsApp su un qualsiasi device, questo ci permette di fare login sui server di WhatsApp da una piattaforma fissa come se fosse un qualunque device mobile.

Il nostro file di configurazione conterrà le seguenti informazioni:

```
cc=
phone= 39348*******
id=
password= 1MWlKhbvqHJx/+geHWbwD8iHTsI=
```

La riga che presenta il "cc=" conterrà le informazioni sul prefisso del nostro numero di telefono, nel mio caso è già stato unito al numero di telefono per un errore che veniva restituito da shell quando si tentava l'esecuzione da uno script del comando per l'invio del messaggio, in questo modo si è bypassato il problema.

Il campo "phone=" conterrà il numero di telefono di telfono da cui vorremo inviare i messaggi WhatsApp.

Infine per ultimo abbiamo il campo "password=" che contiene la password per il login. La password viene ottenuta quando si registra il nostro numero attraverso yowsup-cli, attraverso il procedimento di registrazione che andremo a descrivere.

4.1.1 Registrare un numero attraverso Yowsup

La registrazione viene effettuata attraverso due passi, simile a come funziona sul WhatsApp per i device mobili, per prima cosa dovremo fare richiesta per un codice di autenticazione.

alessandro@Linux-Ale:~/Yowsup/src$ python yowsup-cli -c config.example -r sms

attraverso il seguente comando si fa richiesta di un codice di autenticazione tramite SMS al numero inserito dentro il file di configurazione che abbiamo precedentemente settato. L'output di riposta al nostro comando sarà il seguente:

```
Detected cc: 39
status: sent
retry_after: 1805
length: 6
method: sms
```

A questo punto sul numero selezionato arriverà un SMS contenente un codice di verifica per il numero.

A questo punto possiamo procedere con il secondo passaggio cioè fare richiesta di registrazione al servizio tramite il comando:

alessandro@Linux-Ale:~/Yowsup/src$ python yowsup-cli -c config.example -R 829-570

le ultime 6 cifre in grassetto del comando sono il codice che nel mio caso avevo ottenuto tramite messaggio.
A questo punto otterremo una risposta dal nostro servizio come la seguente:

Detected cc: 39
status: ok
kind: free
pw: 1MWlKhbvqHJx/+geHWbwD8iHTsI=

Il campo pw andrà inserito manualmente all'interno del file di configurazione che così ci permetterà di autenticarci al server di WhatsApp tramite la combinazione numero di telefono, password.

4.1.2 Inviare messaggi WhatsApp e lo script bash

A questo punto inviare messaggi di WhatsApp attraverso Yowsup è relativamente semplice infatti basterà digitare il seguente comando nella shell UNIX.

```
python /home/alessandro/Yowsup/src/yowsup-cli -c
/home/alessandro/Yowsup/src/config.example -s $1 "$2"
```

il comando che vediamo sopra prende in input due valori il primo indica il numero, compreso il prefisso senza il '+' del destinatario, il secondo invece è una stringa che indicherà il messaggio da inviare a quest'ultimo.

Come possiamo vedere dovremo sempre richiamare il client di Yowsup per fare questo e dovremo passare anche il file contenente la configurazione relativa al nostro numero di telefono.

Il codice dello script che abbiamo visto essere richiamato dal nostro servizio REST quindi sarà come segue:

```
#!/bin/bash
 python /home/alessandro/Yowsup/src/yowsup-cli -c
/home/alessandro/Yowsup/src/config.example -s $1 "$2"
```

4.2 Telegram-Cli

Come per Youwsup anche Telegram-Cli è un client utilizzato per inviare messaggi da shell UNIX di Telegram anche questo è programma a linea di comando ed implementa le API di Telegram, permette all'utente di accedere e usare Telegram, offrendo tutti i possibili servizi di un client Telegram standard.

Per poter utilizzare il servizio dovremo servirci di una chiave pubblica, che nel nostro caso si trova /home/alessandro/Telegram /tg-server.pub, questa chiave ci permetterà di accedere al server pubblico di Telegram.

Anche qui l'autenticazione dell'utente avviene mediante la coppia di valori numero di telefono e password.

Oltretutto per poter utilizzare il nostro client ci serviranno anche le librerie readline o libedit, openssl, libconfig (se vogliamo fare uso della configurazione) e liblua, queste librerie si possono installare facilmente attraverso il comando di Ubuntu:

sudo apt-get install libreadline-dev libconfig-dev libssl-dev lua5.2 liblua5.2-dev libevent-dev make

4.2.1 Registrare un numero attraverso Telegram

Al contrario di Yowsup, Telegram-Cli è in grado di eseguire più comandi grazie al fatto che quando lo avviamo questo permette di inviare più comandi alla volta senza dover sempre richiamare il client stesso.

alessandro@Linux-Ale:~/Telegram/bin$ ./telegram-cli -k ../tg-server.pub

Quindi quando avvieremo il client attraverso il comando precedente per la prima volta avremo che il client ci chiederà di inserire il nostro numero di telefono, più precisamente quello che ci verrà chiesto sarà posto nel seguente modo.

Telegram-cli version 1.1.0, Copyright (C) 2013-2014 Vitaly Valtman
Telegram-cli comes with ABSOLUTELY NO WARRANTY; for details type `show_license'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show_license' for details.
I: config dir=[/home/alessandro/.telegram-cli]
Telephone number (with '+' sign): +39348*******

Come possiamo vedere dovremo inserire il nostro numero di telefono con il prefessi anticipato dal segno '+' una volta convalidato il comando avremo il seguente responso.

User is not registered. Do you want to register? [Y/n] y

A questo punto una volta che avremo dato come input 'Y' avremo che il nostro client ci restituirà a video:

Ok, starting registartion.
First name: Alessandro
Last name: Brizzi
Code from sms (if you did not receive an SMS and want to be called, type "call"): 25782

Come possiamo vedere dopo aver affermato di voler continuare con la registrazione il client ci chiede il nome e il cognome che poi verranno associati al nostro numero di telefono che abbiamo inserito, dopodiché invia un SMS al nostro numero contente il codice di autenticazione che sopra ho evidenziato in grassetto.
Completato questo passaggio saremo in grado di utilizzare tutti i comandi relativi al client.

4.2.2 Inviare messaggi Telegram e lo script bash

Una volta completati tutti i settaggi sarà possibile inviare messaggi dal numero di telefono registrato, attraverso gli opportuni comandi.
Al contrario di Yowsup, però Telegram-cli ha alcuni difetti che ne complicano l'integrazione con il Webservice.

Prima di tutto andiamo a descrivere come si inviano dei messaggi di Telegram attraverso il servizio Telegram-cli, per fare questo come primo passaggio dobbiamo andare a "registrare" dentro la "rubrica" del nostro client, l'utente a cui vogliamo inviare il messaggio.

add_contact 3934664***** Arianna Brizzi

Attraverso questo comando andiamo a creare un "record" all'interno del nostro client che assocerà al numero inserito un determinato nome e cognome.
Ora non resta che inviare il messaggio e il comando per farlo è il seguente.

msg Arianna_Brizzi $1

dove con $1 si intende la stringa contente il messaggio.
Ora sorgono i problemi relativi all'automatizzazione di questo servizio.

Infatti come possiamo notare se noi registriamo un numero questo dovrà avere associato un nome ed un cognome, quindi per poter inviare il messaggio a quella determinata persona bisognerà passare al comando msg i parametri nome_cognome dell'utente destinatario e la stringa contente il messaggio.
Come sappiamo bene però il servizio non può conoscere anche il nome ed il cognome dell'utente dato che per inviare il messaggio il valori essenziali sono la stringa contente il messaggio e il numero di telefono preceduto dal prefisso.

Per risolvere il problema noi possiamo anche registrare il numero nel modo seguente:

add_contact 3934664***** 39 34664*****

ovviando così al problema del nome e del cognome.
Ora sapendo che il numero viene "registrato" nel nostro client come nome_cognome per inviare il messaggio, basterà richiamare il comando msg nel seguente modo:

msg 39_34664***** $1

dove con $1 si intende la stringa contente il messaggio.
In questo modo utilizzeremo sempre e solo il numero ed il prefisso come identificativi e non avremo bisogno di dati aggiuntivi.
Il codice dello script invece sarà il seguente:

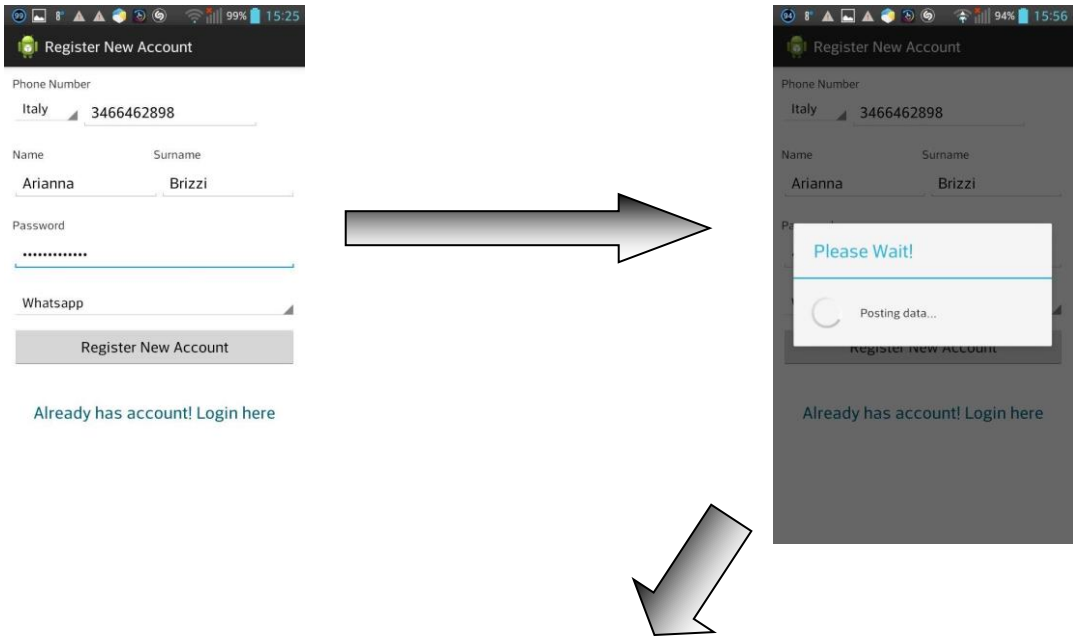
```
#!/bin/bash
prefix=$1
number=$2
subject="$3"
phonenumber="$1$2"
contact="$1_$2"
tgpath="/home/alessandro/Telegram"
cd ${tgpath}
${tgpath}/bin/telegram-cli -k ${tgpath}/tg-server.pub -WR -e "add_contact $phonenumber $prefix $number"
${tgpath}/bin/telegram-cli -k ${tgpath}/tg-server.pub -WR -e "msg $contact ${subject}"
```

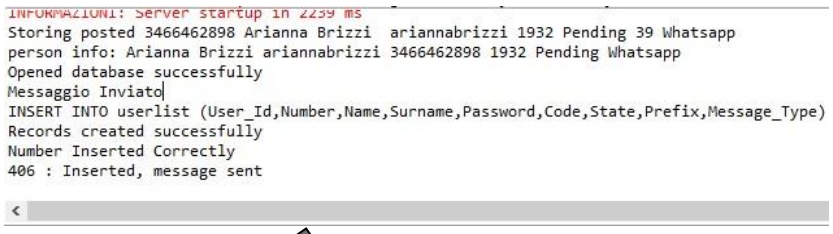
Nello script come possiamo vedere utilizziamo accanto alla chiamata del client le opzioni '-WR' che permettono di non ricevere output a schermo e '-e $1', dove $1 la stringa contente il comando da eseguire, che permette di eseguire il comando passato al client e una volta eseguito con successo esce automaticamente dal client stesso.
In questo modo il nostro Webservice REST potrà richiamare i comandi per inviare i messaggi Telegram attraverso la chiamata del metodo exec che abbiamo visto in precedenza.

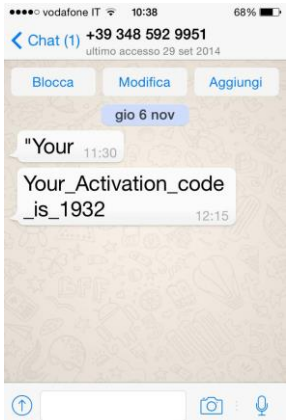
Capitolo 5

Dimostrazione

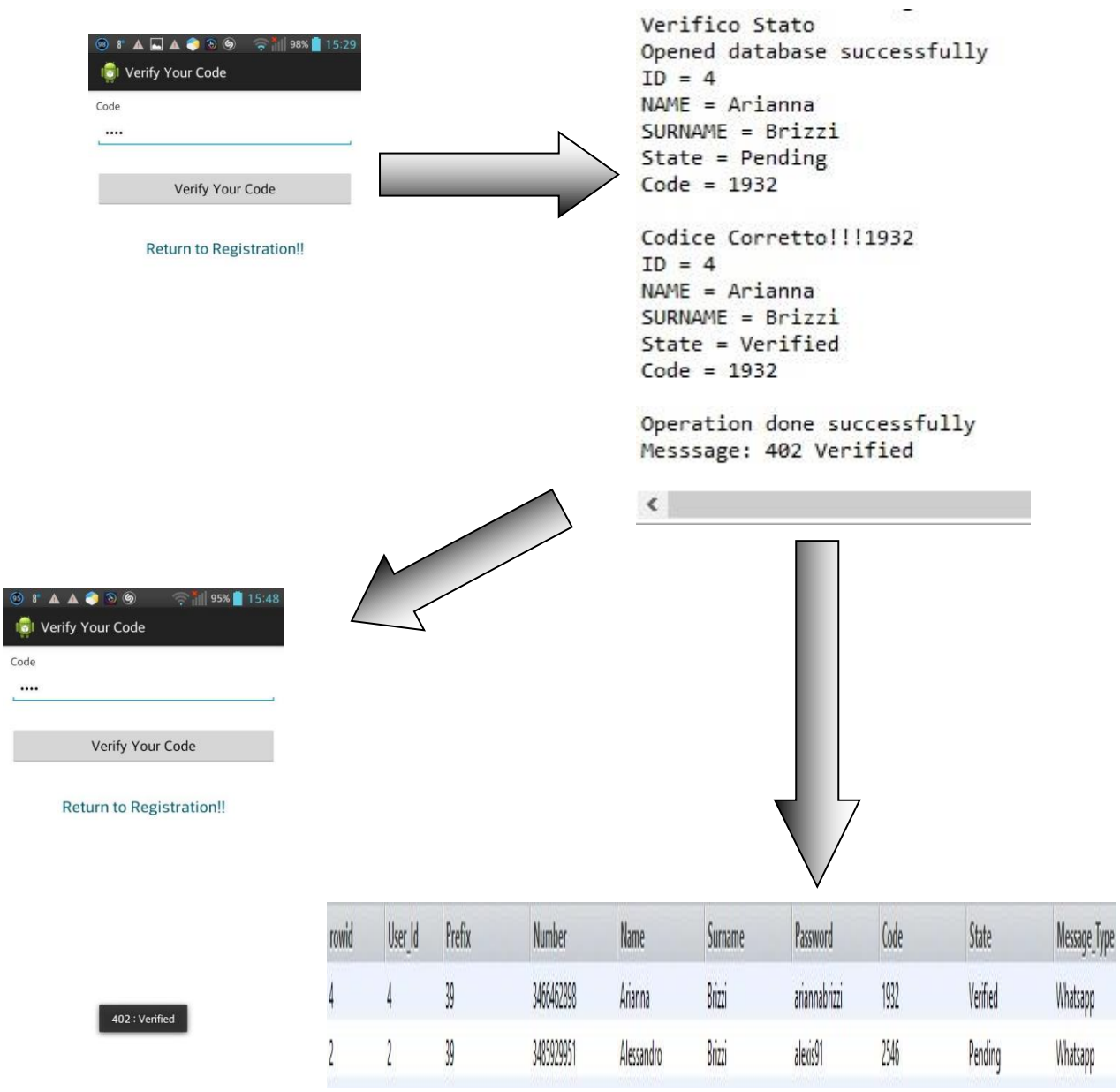
In questo capitolo verrà riportato l'output dell'applicativo con esplicitate le classi (activities) utilizzate e i file XML di layout implementati.







rowid	User_Id	Prefix	Number	Name	Surname	Password	Code	State	Message_Type
4	4	39	3466462898	Arianna	Brizzi	ariannabrizzi	1932	Pending	Whatsapp
2	2	39	3485929951	Alessandro	Brizzi	alexis91	2546	Pending	Whatsapp



Come possiamo vedere dalle immagini la nostra applicazione parte dall SignupActivity che permette di inviare i dati al WebService REST che risponderà con un output che ne indica il corretto funzionamento o meno, nel frattempo attraverso le opportune funzioni che richiamano i servizi di Yowsup e Telegram-Cli, inviamo un messaggio al nostro utente e aggiungiamo una Tupla al nostro Database contenente le informazioni principali dell'utente appena inserito. Nella fase successiva invece ci troviamo davanti alla CodeVerificationActivity nella quale il nostro utente inserirà il codice appena ricevuto, inviandolo al WebService che aggiornaerà così la Tupla inserita precedentemente nel Database, e restituirà al nostro device una stinga JSon che indicherà la corretta registrazione o meno del nostro utente.

Capitolo 6

Conclusioni e futuri sviluppi

L’obiettivo dell’elaborato di tesi è stato lo sviluppo di un applicativo per l'autenticazione di Sim attraverso messaggistiche che esulano dal classico SMS.
Il lavoro svolto porta un contributo innovativo nella sua realizzazione e, cioè, una fornitura di servizi secondo i canoni del WEB 3.0.
Il WEB 3.0 ha alla base l’idea di “connessione semantica”; considerare cioè il WEB come una rete di dati che descrivono dati (metadati); ovvero la connessione tra informazioni, collegamenti fra i dati gestiti direttamente dai computer, in termini immediati "macchine che parlano fra macchine", ovvero servizi documentati che possono essere combinati fra loro creando funzionalità innovative.
La tesi è stata sviluppata su sistemi Unix e su sistemi Mobile Android, per la sua larga diffusione attuale nel mercato e per la sua facile fruibilità, nonché vasta documentazione disponibile sul Web.
La soluzione proposta ha fatto utilizzo di API appositamente create in un WebService REST; sono anche state utilizzate API rese disponibili attraverso Client per l'invio di messaggi online attraverso comandi shell, i quali poi sono stati implementati nel progetto attraverso script esterni creati appositamente.
Per un immediato futuro è auspicabile la realizzazione di funzionalità che permettano il rinvio del codice qualora questo non sia stato ricevuto dall'utente, il che comprende le apposite API che andranno create sul servizio REST per poter verificare il codice richiesto e se l'utente è effettivamente registato.
Si potrebbe anche implementare un servizio che permetta ad un utente di cambiare il suo numero di telefono associato, dato che nella vita di tutti i giorni potrebbe capitare di dover cambiare numero di telefono.
Anche questo dovrebbe essere fatto creando le opportune API che permettano di fare richiesta del cambio di numero verificando che questo non sia già presente in modo da evitare furti d'identità.

Il lavoro, per come è stato progettato e implementato, è facilmente integrabile con nuove funzionalità, questo è reso possibile dalla struttura delle API a disposizione che hanno un formato standard (JSON) per la rappresentazione delle risorse. Risulterebbe, quindi, semplice, attraverso la creazione di nuove activity e l'uso del metodo di esecuzione delle API, opportunamente standardizzato.
Questo metodologia di autenticazione potrebbe anche essere utilizzata in diverse tipologie di progetti che spaziano dal social a una tipologia più aziendale, dato che l'idea alla base del progetto è la semplice verifica di appartenenza di un determinato numero di telefono ad un determinato utente.
Altri importanti sviluppi potrebbero essere dati dal implementare applicazioni che permettano l'utilizzo delle suddette api attraverso sistemi diversi da Android, come IOS o WinPhone, anche questo viene reso molto più semplice dalle API di tipo REST che permettono ad un qualsiasi sistema mobile di accedervi utilizzando sempre un standard HTTP comune ad un qualsiasi device avente connessione internet.
Al momento la sua più auspicabile implementazione, nonché motivo originale dello studio della suddetta tesi, è un progetto interno il cui obbiettivo primario è creare un social network basato sui numeri cellulare

Bibliografia

[1] Cesare Pautasso, Olaf Zimmermann e Frank Leymann, "*RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision*" in 17th International World Wide Web Conference (WWW2008), Beijing, China, 2008-04.

[2] Nicholas C.Zakas, "*High Performance JavaScript*.": O' Reilly Yahoo Press.

[3] Christophe Porteneuve, "*Sviluppare applicazioni Web 2.0*".: Apogeo, 2007.

[4] ECMA-404 The JSON Data Interchange Standard.

[5] Massimo Carli, "*Sviluppare applicazioni per Android*".: Apogeo, 2010.

[6] Canalys Società Internazionale per ricerche di mercato. http://www.canalys.com/.

[7] Reto Meier,.: John Wiley & Sons.

[8] Activity, la "prima pagina" dell'applicazione: http://www.html.it/pag/19499/le-attivit-activity/

[9] Il ciclo di vita di un'Activity: http://www.html.it/pag/48652/il-ciclo-di-vita-di-unactivity/