Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate
Corso di Laurea in Informatica

UNIVERSITÀ DEGLI STUDI
DELL'INSUBRIA

Progettazione ed implementazione di un sistema ottimizzato per l'utilizzo mobile di una rete sociale decentralizzata basata sul Cloud

Relatore: Prof.ssa Barbara Carminati

Tesi di Laurea di
Gaglio Federico
Matricola 719110

Anno Accademico 2016-2017

Indice

Elenco dei simboli e delle convenzioni

Entità

KMS	Key Manager Service
RMS	Rule Manager Service
OSN	Online Social Network
APP	Applicazione Android
PFS	Path Finder Service

Attributi

IDuser	Identicativo univoco dell'utente connesso alla rete sociale
IDuser(requestor/ed)	Identicativo associato a un utente richiesto/richiedente
IDresource	Identicativo univoco di una risorsa del sistema
n	Valore nonce (numero generato casualmente)
IV	Initialization Vector (parametro AES)
salt	Valore pseudo-casuale (parametro AES)
keySize	Dimensione dei blocchi dell'algoritmo AES
interationCount	Numero di iterazioni dell'algoritmo AES
passPhrase	Stringa segreta conosciuta solo dall'utente

Cifratura

$\mathbf{K^+_{KMS}/K^-_{KMS}}$	Chiave pubblica / privata associata a KMS
$\mathbf{K^+_{RMS}/K^-_{RMS}}$	Chiave pubblica / privata associata a RMS
$\mathbf{K^+_{PFS}/K^-_{PFS}}$	Chiave pubblica / privata associata a PFS
$\mathbf{K^+_{APP}/K^-_{APP}}$	Chiave pubblica / privata associata all'applicazione mobile
$\mathbf{K^+_s}$	Chiave simmetrica a conoscenza del servizio coinvolto e dell'utente

1

Introduzione

1.1 La potenza delle reti sociali

Due miliardi di persone, ogni mese, utilizza attivamente il più grande servizio di rete sociale conosciuto: Facebook[1]. Un numero che, comparato alle sette persone di cui era composto il team per la creazione di ciò che conosciamo ora come “Internet”, e alla popolazione mondiale, attualmente di circa 7.5 miliardi di persone, rende ancora più impressionante l'impresa in cui sono riusciti. La realtà delle reti sociali è, in verità, cosa ben nota da molto più tempo grazie al boom del nuovo millennio causato principalmente dalla possibilità di sottoscrivere connessioni internet molto più veloci e a prezzi accessibili a gran parte delle persone che, ormai da qualche anno, possiedono un computer che si scatena una sempre più esponenziale crescita di interesse ed attrazione rispetto a questo nuovo sistema di comunicazione, come fu il telefono decenni prima.
Gli anni tra il 1995 e il 2005 sono stati, infatti, gli anni d'oro per lo sviluppo delle prime reti sociali: la novità di una comunicazione istantanea con il mondo è qualcosa che attrae chiunque, che sia un neofita del Web o meno. Sono state decine le reti sociali che furono sviluppate e tutt'ora dominano o crollano nel Web: Sixdegrees.com, Friendster, MySpace, Twitter, LinkedIn, Bebo, Facebook sono solo alcune di queste. Ancora oggi la creazione di nuove reti sociali continua ad essere uno dei punti cardinali di internet, in grado di muovere decine di migliaia di persone, portandole a condividere esperienze, pensieri e foto.
Ma questi servizi, totalmente gratuiti, per raccogliere un maggior bacino di utenti devono in qualche modo mantenere innanzitutto i costi gestionali per offrire un servizio sempre disponibile e reattivo, senza contare un guadagno monetario che può servire a ripagare sia gli sviluppatori di tale risorsa che coloro che vi hanno investito inizialmente. Come è stato detto poco prima, migliaia (se non milioni) di persone che utilizzano questi servizi si trovano dunque a condividere informazioni nel cyber-mondo e, di conseguenza, al servizio di rete sociale di queste risorse. Informazioni che, solitamente, comprendono quelle personali (es. nome, cognome, età, etc.) oltre che a preferenze che, a livello di marketing, sono preziose dal punto di vista economico. Queste informazioni, di norma, sono di diritto cedute al sistema della rete sociale al momento dell'iscrizione tramite un Termine contrattuale (più conosciuto come T&C), ovvero un contratto che il servizio di rete sociale firma virtualmente con i suoi sottoscriventi. La privacy dell'utente, tipicamente, viene rispettata con la protezione della propria identità a coloro a cui vengono vendute queste informazioni, ma ciò non influisce sul capitalismo che l'azienda apporta nei confronti della persona che vuole utilizzare questo servizio, diventando quindi bersaglio di pubblicità mirate e alla profilazione da parte delle aziende di marketing. Negli ultimi anni, inoltre, sono sorti sempre più dubbi e studi riguardanti il rilascio di queste informazioni portando anche a veri e propri disastri tra cui il rilascio delle informazioni personali di utenti e la possibilità di risalire tramite alcune informazioni più “circostanziali” al relativo utente. E' per questo ed altri motivi che le reti sociali definite come OSN (Online Social Network) sono state addirittura riferite come effettivi pericoli per la società:

l'impossibilità di avere controllo totale su ciò che si dice e il poter limitare tutto ciò che viene reso pubblico in modo effettivamente non totale, ma bypassato dal gestore del servizio rende le reti sociali un pericolo che può creare veri e propri disastri culturali e sociali.

La soluzione dei problemi delle OSN vengono parzialmente risolti da un'altra architettura: le **DSN** (***Decentralized Social Network***) sono esattamente l'opposto di ciò che una OSN vuole, ovvero la decentralizzazione delle risorse su di altri domini, e dunque una totale oscurità tra ciò che il servizio della rete sociale conosce e ciò che l'utente vuole che il mondo conosca.
Una DSN, riassuntivamente, è una federazione di provider che possiedono, nella loro totalità, il servizio di rete sociale, ma che singolarmente non hanno informazioni necessarie per effettuare alcun tipo di modifica o di collegamento tra le risorse e i vari dati dell'utente, senza considerare tutte le azioni e modifiche che quest'ultimo apporta alle stesse. E proprio grazie a questo vantaggio l'utente finale si trova a poter gestire, in modo totalmente autonomo e privato, qualsiasi informazione carica sulla rete sociale, potendo dunque intervenire sulla privacy del proprio account e delle proprie risorse, con un'amplissima gamma di impostazioni.
Il tentativo di spostare il mondo delle reti sociali su questo tipo di architettura è nel suo picco di sviluppo al momento: vi sono decine di servizi che tentano di competere con i colossi di internet, progetti come Diaspora, Persona, SafeBook e molte altre, seppure il problema di data mining da parte dei provider continui a persistere, seppure molto più ridotto. Si è giunti dunque alla conclusione di sviluppare una rete sociale DSN tramite web che implementasse un robusto sistema di crittografia in grado di limitare al minimo le interazioni del provider con i dati dell'utente, mascherandoli e rendendoli il meno possibilmente oggetti al tentativo di marketing delle risorse di cui ogni provider è oggetto di discussione.

1.2 Breve introduzione sugli smartphone

E' stimato che il numero di smartphone nel mondo sia di 2.1 miliardi, in regolare crescita (2.32 per il 2017)[2]. Avere infatti un telefono capace di una connessione internet è reputato oggigiorno comune: i vecchi sistemi di comunicazione come SMS sono quasi totalmente deprecati, dato che viene facilmente riscontrato nelle offerte degli ISP che tendono a puntare su servizi a consumo di banda. E con l'avvento degli smartphone, definibili come “personal computer portatili”, il mondo delle reti sociali si è ritrovato a vivere una seconda era d'oro. Lo sviluppo di applicazioni in grado di fornire gli stessi servizi di un computer ma nel palmo della propria mano è ciò che ha portato una vera e propria rivoluzione nel mondo della programmazione per il produttore e della tecnologia per il consumatore.
Gli stessi produttori di telefonia infatti, tutt'oggi, installano spesso e volentieri applicazioni per accedere ai servizi di rete sociale, sviluppate da questi ultimi, in modo da invogliare tale utilizzo del telefono, liberando così da quest'incarico il cliente a cui viene venduto il prodotto.
E se queste applicazioni sono oggi cosa comune, inizialmente non lo erano altrettanto: l'avvento di App specifiche è una questione nuova (2013 c.a.) e prima di allora l'accesso al sito doveva essere (e tutt'ora può essere) effettuato via il browser integrato nel sistema operativo del dispositivo mobile.

Seppure i siti di servizi di rete sociale siano una delle grandi cause dello sviluppo del modo in cui la rete internet si approccerà al mobile, il problema che questa ha causato (specialmente in passato), mettono in luce come internet possieda delle grandi falle nel proprio sistema, e di come il mondo (specialmente per chi, come noi, che deve creare pagine web disponibili per questi dispositivi) si sia

dovuto adattare e applicare per trovare soluzioni in grado di soddisfare le esigenze di un evento sempre più in crescita.
Seppur i primi browser per mobile nascono nel 1999, bisogna attendere il 2007 con l'introduzione di schermi multi-touch per ottenere un migliore accesso al mondo del Web da dispositivi mobili e una previsione quasi sconcertante che nasce dalla ITU (International Telecommunication Union) nel 2010: entro 5 anni vi saranno più dispositivi mobili che computer nell'accesso a internet. Previsione che viene confermata nel 2014, quando negli Stati Uniti questo fenomeno si verifica, dimostrando la potenza e la versatilità di questi dispositivi.

Per chi ha lavorato nei reparti di sviluppo web in quegli anni la situazione venuta a crearsi è stata davvero difficile. Le soluzioni per correre ai ripari o sfruttare questo boom sono state svariate e non sempre rivolte con uno sguardo al futuro: purtroppo l'intera questione era stata già sottovalutata dalle autorità che dovevano occuparsi di standardizzare l'avvento di questa rivoluzione, basata già su alcuni linguaggi aventi linee guida non propriamente rigide e stringenti ha creato ben presto allo sviluppo di molteplici mal sviluppati siti web che hanno fornito “cattive abitudini” agli sviluppatori, rallentando così il progresso di una successiva standardizzazione. Nel 2013 è stato fatto il paragone tra la tecnologia moderna e quella passata[3]: seppur in quell'anno si poteva possedere un dispositivo con una capacità di calcolo superiore a quella del computer a bordo dell'Apollo 11, si era affetti da gravi problemi di performance del web, tali da rendere la connessione pari a quella del 1996, a causa della mentalità di sviluppo dei programmatori che non consideravano giustamente le risorse non più “illimitate” fornite dai dispositivi mobili. Fortunatamente la MWI (Mobile Web Initiative)[4] ha creato un W3C per standardizzare le pratiche e le tecnologie adottate per la creazione di siti web mobili, che vengono aggiornate di continuo, pur tenendo in considerazione gli standard attuati per il web e le validazioni che vengono fornite agli sviluppatori per creare un web corretto e dunque “pulito”.

Seppure in passato fosse richiesto di utilizzare un web browser per effettuare qualsiasi operazione che non fosse di telefonia mobile, si è arrivati a percepire che fosse uno dei maggiori problemi nello sviluppo di nuove tecnologie per smartphone.
Come era possibile creare nuove funzionalità che potessero essere implementate nel telefono (come lo è per sveglie, calendari e simili), ma che non fossero distribuite direttamente nel “pacchetto base” fornito all'utente inizialmente? Perché non dare la possibilità a sviluppatori di creare queste applicazioni, al posto di monopolizzare una risorsa che potrebbe offrire così tante possibilità e prospettive?
E' per questi motivi che le Applicazioni Mobili (più conosciute semplicemente come “App”) sono nate, ed è grazie a questi punti di forza (come varietà, facilità di distribuzione, etc.) che gli smartphone di oggi sono diventati uno dei grandi punti focali per la distribuzione di servizi via telefono, surclassando in alcuni campi le capacità limitate di dover passare attraverso un browser preinstallato sul telefonino. Le applicazioni mobili infatti rivestono un ruolo già coperto dai siti adattati per la visione su schermi più piccoli, con la differenza di funzionalità e una gestione migliorata delle richieste e risorse disponibili: invece che dover incaricare il browser di effettuare la gestione, la “lavorazione” e l'invio di queste richieste web, l'applicazione incarica lo smartphone stesso di effettuare queste operazioni, sfruttando con grandi migliorie le risorse disponibili e adattandosi alle condizioni del telefono, senza contare alla possibilità di sviluppare le funzionalità dell'applicazione stessa attorno alle capacità del telefono.

1.3 Scopo e struttura della tesi

Alla luce di queste due realtà si è venuta naturalmente a creare la necessità di sviluppare una nuova rete sociale ma di tipo decentralizzata in grado di fornire un controllo maggiore agli utenti della propria privacy, ma che, allo stesso tempo, sia compatibile con l'inarrestabile crescita del settore mobile. Il primo problema è stato affrontato nella tesi magistrale di DiGiacomo Erika Antonia e Magistrale Procopio Michele, i quali hanno implementato i moduli che compongono la base di una DSN e relativa interfaccia grafica da browser, creando dunque il progetto DSNProject. Purtroppo lo sviluppo per un ambiente mobile è stato tralasciato a causa della mancanza di tempo e risorse: in questa tesi si tratta la necessità di riuscire ad adattare il lato client del progetto DSNProject in modo tale da riuscirsi ad interfacciare con tutti i moduli già esistenti ed essere riprodotto su qualunque dispositivo mobile sino ad oggi supportato, il tutto senza influire in alcun modo sulla sicurezza e anonimato delle comunicazioni che lo coinvolgono.

Seppure nella prima fase di studio la soluzione trovata sembrava spingere verso una modifica del sistema di generazione delle pagine di OSN senza intaccare il sistema di criptazione CS, ben presto ci si è accorti dei problemi che quest'ultimo modulo possiede non erano né risolvibili né ignorabili: effettuando uno studio approfondito sulle soluzioni possibili si è dovuta cambiare la soluzione incentrando lo sviluppo verso un'applicazione mobile (App) che potesse sostituire CS, risolvendo il problema per l'ambiente mobile.

Come appena presentato, nel Capitolo 2, verranno trattati i due studi di fattibilità effettuati, spiegando tramite un'approfondita digressione i problemi relativi alla prima soluzione, per poi passare alle soluzioni fornite dal secondo studio. Nel capitolo successivo, Capitolo 3, viene invece mostrata la struttura di riferimento di DSNProject, andando a trattare i vari moduli e il loro funzionamento nella totalità del progetto. Dal Capitolo 4 si inizia effettivamente ad analizzare l'ambiente di sviluppo e l'architettura del progetto riguardante la soluzione proposta, ovvero di una applicazione per sistemi Android, mostrandone storia, struttura e funzionamento generico. Il Capitolo 5 riporta con maggior attenzione i protocolli di sicurezza e gli algoritmi crittografici utilizzati nel progetto, dai moduli già esistenti e dall'applicazione Android. Nel Capitolo 6 si può ritrovare l'implementazione effettiva del codice e delle funzionalità, il tutto accompagnato da brevi discussioni tecniche relative alle scelte di sviluppo e al funzionamento di alcune parti. In conclusione, nel Capitolo 7, si ritrovano le riflessioni relative allo sviluppo della tesi, le soluzioni alternative e future considerazioni di cui tener conto per un possibile sviluppo futuro del sistema. Come capitolo bonus vi è una breve lista degli attacchi crittografici trattati nella tesi, da utilizzare come riferimento nel caso di necessità.

2

Un primo approccio risolutivo

2.1 DSNProject

E' sulle basi di DSN e sulla necessità di impedire data mining di alcun tipo che si è voluti cercare di mettere in atto tutto ciò che, sino a questo momento, è teoricamente migliore e conveniente per ogni utente: la nascita del progetto DSNProject è la volontà di creare un'effettiva DSN che non sia basata o si appoggi su alcun tipo di rete sociale già esistente. Effettivamente parlando, questo approccio risulta comunque inesplorato a causa della natura stessa delle reti sociali.
DSNProject nasce come progetto effettivo da due ragazzi dell'università dell'Insubria che, sempre per una tesi, decidono di procedere nello sviluppo di una struttura decentralizzata specifica: ogni modulo che la compone deve essere a se stante, appoggiarsi su provider diversi e non è possibile che le informazioni che i vari provider possiedono vengano scambiate o unite tra di loro. La rete sociale è stata inizialmente basata, seppure questa scelta poi venga abbandonata successivamente per problemi relativi allo sviluppo, sul concetto definito come **Peer-To-Peer**, ovvero di una totale capacità di partizionare e distribuire le moli di lavoro ad ogni collaboratore della struttura: la realtà effettiva è ben diversa dalla definizione di P2P standard. La struttura infatti assomiglia decisamente di più ad un sistema **Server-Client** per varie ed ovvie ragioni: il collegamento alla parte Server, che varia a secondo delle operazioni, è sempre ben presente e rappresenta un caposaldo della struttura. Questo si occupa infatti della distribuzione e/o condivisione di una specifica risorsa (es. password, amicizie, immagini) senza cui sarebbe altrimenti impossibile riuscire ad effettuare qualsiasi tipo di autenticazione e salvataggio. Senza questa base sarebbe infatti necessario richiedere le varie risorse a coloro che la possiedono, causando vari problemi di “anonimia” (verrebbero di fatti svelate le amicizie e il grado di collegamento che le legano), problemi di perdita di dati (se coloro che possiedono una determinata risorsa la cancellano, questa sparisce totalmente dal sistema) e di rintracciamento istantaneo delle risorse (se nessuno di chi possiede tali risorse è attivo, l'utente che la richiede deve rimanere in attesa di qualcuno che la abbia).
Altro fattore che dimostra la mancata implementazione del sistema P2P è proprio il metodo di comunicazione scelto per la comunicazione tra client e server: la richiesta che il client iniziale effettiva non viene elaborata basandosi sulle risposte dell'altro peer a cui viene richiesta ma, bensì, ogni parte della struttura ha compiti ben precisi e diversi livelli di importanza; inoltre la comunicazione tra questi “peer” diventa sempre più complessa e stratificata, effettuando un vero e proprio “passaparola” per arrivare all'effettivo possessore della richiesta.

Ma, anche con questa effettiva struttura, lo scopo finale dell'applicazione non cambia: DSNProject nasce come social network sicuro in grado di condividere risorse, capace di fornire un pieno controllo della sicurezza all'utente, in modo tale da esser a disposizione di chi utilizza il servizio e non di chi lo ha creato.

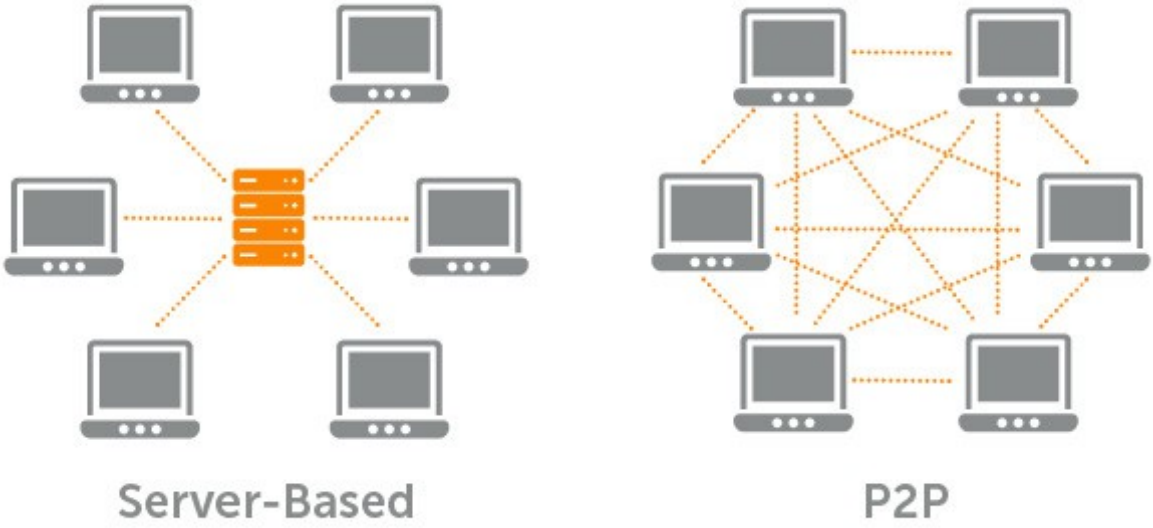

Figura 2.1.1: Struttura Server-Client e P2P a confronto

2.2 Progettazione

La seguenti tesi si basava su di un iniziale quesito: *è possibile adattare il Front-End dell'applicazione DSNProject in modo tale da ottenere una versione in grado di essere replicata senza alcuna forma di inconsistenza o utilizzo su un dispositivo mobile?*

Inizialmente, per eseguire tale cambiamento, si è decisi di essere fedeli il più possibile alla idea di base al progetto, evitando dunque qualsiasi tipo di modifica alla struttura dello stesso: sfruttare i **framework** che lo componevano (come **AngularJS**) e le classi già generate dall'ambiente sembrava la soluzione più adatta e semplice. Nell'effettivo, la risoluzione di tale problema apportava più domande dal punto di vista grafico e di design che effettivamente da quello logico.
Per creare un'applicazione web tale da non incorrere in alcun problema è infatti necessario seguire le **linee guida** stabilite dal **W3C** in materia di accessibilità mobile[5]. E' da sottolineare che le linee guida esposte sono in realtà convenzioni create e stabilite dall'ente **World Wide Web Consortium** per ripiegare al sempre più crescente problema di una mancanza totale di **standard** sullo sviluppo del web per mobile: nessuno impone ai programmatori di eseguire tutti i punti per filo e segno come descritti dalle regole che si possono ritrovare nel trattato, ma per poter avere una migliore **performance** tale da interagire con qualsiasi dispositivo e riuscire a raggiungere un punto di incontro comune per ogni programmatore in tutto il mondo è fortemente consigliato seguire ciò che viene suggerito dal W3C.
L'**accessibilità** al web dal punto di vista "**mobile**" in realtà è trattato in modo ben maggiore e più ampio: tutte le regole che si possono trovare nel link sopra fornito sono utilizzabili per -più genericamente parlando- tutti quegli apparecchi che utilizzano un sistema touch-screen o simile, replicando il funzionamento di quello che modernamente chiamiamo smartphone. La realtà dei fatti è che anche altre device utilizzano sistemi simili, che varino semplicemente in dimensione (es. Tablet) o che siano totalmente differenti (es. Smartwatch).
E' infatti sempre importante, nel pensare allo sviluppo dell'applicazione, a tentare di includere

quanto più bacino di utenza senza esclusione di alcun tipo: per questo si parla di accessibilità. **Accessibilità** significa poter garantire a qualsiasi soggetto di poter accedere ad un contenuto web a discapito di qualsiasi fattore che possa imporre un fattore di barriera divisoria, che siano tecnologiche o sociali.

Le liste del W3C sono molteplici e trattano in modo accurato molti punti, seppure alcuni non siano applicabili al nostro caso. Tra le liste si può trovare, oltre all'indipendenza del dispositivo dalla pagina web e agli standard sui contenuti e all'accessibilità dei dispositivi per tutte le pagine web universalmente, anche un trattato riguardante l'uso di CSS per applicazioni mobili[6].
Le soluzioni passate per mostrare una pagina web in mobile, inizialmente, comprendevano lo sviluppo totale e separato di pagine web dedicate ai vari dispositivi e che assomigliassero a applicazioni quasi fossero installate sul telefono. Ma dal 2014 è iniziata un'esponenziale corsa allo stile **CSS RWD**, ovvero **Responsive Web Design**, ampiamente supportato nel 2015 da Google in un cambio del sistema di ranking e rating per i siti listati, a cui fa riferimento il sistema di ottimizzazione per sviluppatori[7]. Le basi su cui si fonda il design responsivo è comparabile a quello dei liquidi e bicchieri di varia forma: seppure il dispositivo su cui viene mostrato il contenuto cambia, non si mantengono i vari elementi nelle stesse posizioni, ma questo viene semplicemente cambiato di forma o posizione pur mantenendo le stesse misure effettive finali.

Figura 2.2.1: Dimostrazione del funzionamento dello stile RWD su Desktop, Tablet e Smartphone

Alternativamente era persino disponibile un framework dedicato per l'interfaccia utente in mobile di AngularJS[8] che avrebbe persino semplificato maggiormente la situazione.

2.3 Limiti della soluzione

Messe in atto queste soluzioni si è iniziato lo sviluppo ma, effettuando uno studio più approfondito, si è incorsi immediatamente in **problemi** ben maggiori e gravanti sull'**integrità** stessa del progetto. Un qualsiasi web browser può eseguire codice Javascript che sia su desktop o su mobile. Naturalmente è necessario avere riguardo per lo sviluppo su di queste ultime piattaforme in quanto le capacità di calcolo e memoria, specialmente su dispositivi passati, non sono allo stesso livello di un computer. Ed è durante lo studio stesso del codice Javascript di CS e nella relativa ricerca della libreria utilizzata che è sorto il problema: la libreria Cripto-js è un progetto che risale al 2009 e

l'ultimo update è stato fornito nel 2012[9]. E' mantenuta nell'archivio di Google con la rimanente documentazione presente. Il fatto che questa libreria non sia stata sviluppata comporta molti problemi a interfacciarsi con le architetture moderne, ma non è questo il problema principale. Il vero problema risiede nel linguaggio stesso su cui è stata fondata: javascript[10].
La crittografia Javascript è, infatti, considerata persino pericolosa poiché fornisce un falso senso di protezione che in realtà può essere facilmente compromesso.

Partiamo dalla base che i messaggi che vogliamo inviare **non utilizzino** alcun tipo di criptazione **SSL/TLS**, come nel nostro caso, e vengano eseguiti dal web browser. Naturalmente nel caso in cui inviassimo i messaggi senza alcun tipo di criptazione saremmo soggetti con estrema facilità ad un **attacco Man-in-the-Middle**[atk2].
Codificando invece tramite browser javascript... la situazione non cambia in modo alcuno: ad un attaccante, invece che eseguire un attacco MITM, basterà ottenere e aggiungere codice crypto dopo aver eseguito quello standard[atk3].
Seppure il tipo di attacco sembri molto complesso e decisamente costoso a livello di risorse, in realtà è decisamente semplice e, secondariamente, non importa il livello di difficoltà. Effettivamente parlando se un attaccante può ottenere un segreto non criptato significa che può controllare nello stesso modo anche una qualsiasi richiesta web, alterandola e inserendo codici malevoli di ogni sorta nella richiesta javascript prima che questi vengano criptati. In secondo luogo, è necessario riaffermare un principio chiave e innegabile della crittografia. L'importanza di un attacco non si basa esclusivamente sulla capacità dell'attaccante di rompere la crittografia generata, ma bensì sulla capacità di replicazione di quest'ultima.

Il problema principale rimane l'ambiente in cui vengono sviluppate tale applicazioni: il web browser. Purtroppo i web browser (come Chrome, Firefox, Safari, etc.) non permettono una criptazione pura in quanto l'ambiente di esecuzione di Javascript è fortemente influenzabile e malleabile nella sua esecuzione. Tra i problemi più rilevanti vi è il codice per il controllo del contenuto: i browser costruiscono le pagine non da singole richieste o trasmissioni, ma da molteplici comunicazioni che sono costituite da Javascript puro o meno. Ma non possiamo nemmeno riprodurre un digest del codice per autoverificarsi da parte del server, questo perché creerebbe una pagina capace di fare da chiave universale e bypassare tutti i dati sui cui dipende la criptazione. Si vengono così a creare le condizioni ideali per un attacco **Cross-site scripting**[atk4].
Purtroppo è impossibile comunque disattivare il metodo di salvataggio del contenuto e di javascript per evitare il caricamento rapido delle pagine web: una volta che il codice “infetto” viene salvato non vi è via d'uscita per l'utente.

L'altro grande problema sopracitato riguardava l'**ambiente di esecuzione** adattabile: pur essendo una funzionalità unica e di grande aiuto per un'esecuzione dinamica, diventa uno dei più grandi problemi di crittografia quando parte delle funzioni da cui dipende può essere sovrascritta senza traccia, e non vi è modo alcuno di controllare l'ambiente durante l'esecuzione. I browser non sono stati creati con l'idea di avere controllo assoluto del loro ambiente, e l'unica soluzione possibile sarebbe di creare un'estensione in grado di garantire una sicurezza in runtime della nostra estensione... ma a questo punto, quest'estensione, potrebbe garantire la sicurezza di qualsiasi comunicazione, e diventerebbe un progetto di grandi dimensioni, senza contare che questo problema dovrebbe essere trattato prima di tutto da coloro che forniscono i web browser, ma per la mole di lavoro necessaria a creare un sistema rivoluzionario di runtime, è stato abbandonato o semplicemente evitato in favore di soluzioni migliori e meno dispendiose/faticose quali SSL.

Altro problemi problemi comprendono quello della **mancanza** totale di un **RNG** (**Random Number Generator**) **sicuro** su Javascript, di un sistema di gestione della memoria sicuro (JS è per lo più gestito da Garbage Collector) e funzioni con caratteristiche di tempistica. Inoltre Javascript manca totalmente di un metodo per salvare in modo sicuro le chiavi ricevute con una sicurezza maggiore di quella con cui salverebbe sul server di un altro.

Il problema non affligge solamente la libreria crypto-js ma anche librerie ben più famose e sicure quali **SJCL**[11], libreria dell'università di Stanford, che purtroppo non è più progetto attivo da un anno a giudicare dagli update.

2.4 Soluzione alternativa

Vedendo l'impossibilità di poter risolvere il problema, si è decisi di cambiare approccio per la risoluzione: se il problema risiede in Javascript e nell'ambiente del browser, la soluzione sta nel cambiare il linguaggio (e dunque l'ambiente) di sviluppo. Pensando ad uno smartphone principalmente, oltre al web browser, per accedere in modo veloce ad un servizio social solitamente si fa utilizzo di una **App**, ovvero di una **Applicazione mobile**. Un'app non è altro che un programma scritto per dispositivi mobili, adattato per l'utilizzo su strumenti dotati di schermi minori di quelli di un pc o di touchscreen.
Altro lato positivo di questo cambiamento sta nel fatto che le applicazioni sono scritte in altri linguaggi, come Java e Objective-C, escludendo dunque totalmente i problemi definiti da Javascript.

Per scegliere su quale ambiente sviluppare il nostro progetto si è decisi di basarsi sulle statistiche di utilizzo e del sistema operativo più utilizzato dai dispositivi mobili, scegliendo dunque, vista la grande varietà di piattaforme su cui è distribuita, Android come sistema di sviluppo[12].

Il sistema di sviluppo di Android è **open source** e, inoltre, è quasi completamente eseguito su di una **Virtual Machine** del linguaggio Java che è leggermente diversa dalla standard **JVM**, chiamata **Dalvik Virtual Machine**.
In parole povere è necessario sviluppare l'applicazione (almeno per quanto riguarda la parte di moduli e eseguibile) proprio nel linguaggio Java. E' dunque necessario controllare che Java possieda la possibilità di eseguire una crittografia sicura e rapida. E tutto ciò è vero: la libreria di default di Java, implementata nella versione 7 è sicura, affidabile e non è stato riscontrato, sinora, alcun tipo di attacco che possa intaccare e compromettere la sicurezza garantita dalla crittografia di Java[13]. Tutta questa sicurezza è inoltre affermata e persino sigillata dal fatto che tutto il codice, a differenza del sistema precedentemente utilizzato, verrà criptato in locale senza utilizzare in alcun modo un sistema Javascript che possa essere compromesso nelle molteplici richieste tra client e server, senza contare che l'ambiente di sviluppo è ora totalmente sicuro.

Purtroppo non è tutto ora ciò che luccica: come abbiamo presentato nel Capitolo 2, la parte di FE è composta da due moduli: OSN che è il sistema che si occupa delle richieste, salvataggi, parziali criptazioni, etc. … ma la seconda parte, ovvero CS, è la parte che l'utente vede e interagisce e viene generata esattamente dallo stesso sistema che gestisce OSN. Questo ci costringe a effettuare una scelta necessaria per la continuazione dello sviluppo in questa direzione, l'unica disponibile: trasformare tutto il codice che sino ad ora generava pagine .jsp in codice eseguibile dal sistema

Android.
Questo problema è in realtà ben più complesso e grave di quanto possa apparire inizialmente: ciò che l'utente percepisce sono pagine dinamiche di linguaggi HTML (con CSS) e codici JS per eseguire operazioni varie (grafiche, comunicazioni HTTP o crittografiche che siano) che distano decisamente molto rispetto a un linguaggio statico come Java e fondato su di una forte tipizzazione. Inoltre la parte di OSN, pur essendo in linguaggio Java, a causa dell'architettura su cui è stato basato non può essere semplicemente trasportata su di Android, anche perché torneremmo sui problemi di generazione Javascript del framework iniziale, senza contare che lo stesso modulo OSN è problematico di principio e, si può notare, che sia stato ideato come placeholder per poter mostrare le funzionalità del progetto. La realtà dei fatti è ben più grave: OSN non dovrebbe quasi esistere se non per piccole comunicazioni tra le parti finali, non dovrebbe salvare alcun dato che possa portare il recupero di informazioni aggiuntive e dunque datamining nella rete e non deve esser un collo di bottiglia insormontabile che rallenti le prestazioni del sistema intero.
Sarà dunque necessario eseguire un **reverse engineering** del framework corrente per poter ottenere del codice Java “puro”, da cui successivamente togliere le funzioni e trasportarle al CS che in questo modo potrà comunicare con gli altri moduli direttamente senza preoccuparsi di passare dal sistema OSN.

3

DSNProject

3.1 Architettura di riferimento di DSNProject

Come gran parte dei programmi, l'effettiva struttura che al di sotto elabora e fornisce dati all'utente viene mascherata e nascosta: DSNProject non fa eccezione e, seppure nell'effettivo l'utente riesca solamente a percepire la parte di Front-End, il progetto è diviso in molti più moduli.

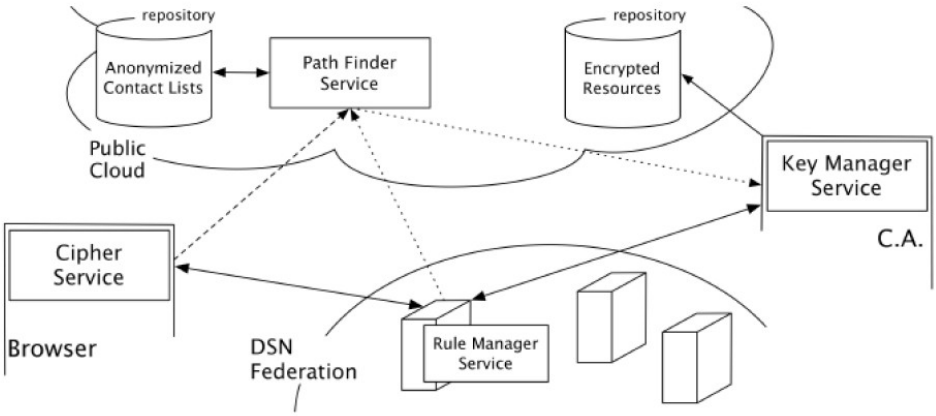

Figura 3.1.1: Architettura originale del progetto DSNProject

La divisione si può effettuare effettivamente in 4 moduli, che spigherò brevemente a seguito.

- **FE: CS e OSN**

  La parte di **Front-End** *è*, da definizione, *l*a porzione di progetto destinata ad interagire con l'utente e permettergli di effettuare specifiche operazioni. Nel nostro caso le parti che si occupano di tali compiti sono due, seppure nell'effettivo si vedrà che queste non sono poi così separate come si potrebbe credere.
  CS (**Cipher Service**) è il servizio di criptazione e codifica di ogni richiesta e messaggio da parte del client. Il ruolo che svolge nell'applicazione è in pratica tra i più importanti: senza questo modulo qualsiasi messaggio non verrebbe criptato e, di conseguenza, non esisterebbe più nessuna sicurezza in fattore di invio e ricezione dei messaggi. E' l'unico modo che, infatti, il client possiede per effettuare qualsiasi operazione in modo sicuro e poter trasferire informazioni. Questo servizio, originariamente, è stato implementato tramite web browser.

OSN (**Online Social Network**) è invece il servizio di gestione di qualsiasi operazione della rete sociale, e assume un ruolo molto ambiguo ai fini dei compiti che compie per la comunicazione della rete: teoricamente parlando è solamente un "**repository**". Un repository è un sistema ideato per tenere traccia di vari dati per effettuare più velocemente ricerche senza effettivamente ripetere le operazioni molteplici volte. Purtroppo nell'effettivo dei fatti i dati che vengono tenuti in memoria spaziano dalle autenticazioni, ai profili, sino alle amicizie ed i file che vengono inviati ad un qualsiasi utente. Già solamente tale mole di informazioni rappresenta non solamente un rischio per quanto riguarda una vera e propria miniera di dati che, attraverso vari attacchi (trattati più avanti) rischiano di compromettere totalmente l'anonimato di un individuo, infatti è anche un problema a livello gestionale in quanto OSN rappresenta un **collo di bottiglia** per le operazioni: se ogni richiesta del CS deve essere salvata o accertata da OSN diventa ben chiaro che una grande mole di dati causerà sicuramente scompensi nelle prestazioni portando, infine, all'offerta di un servizio incapace di mantenere ciò che ha promesso e per cui è stato ideato. Oltre a ciò, inoltre, OSN è anche un gestore delle richieste da parte dell'utente del servizio di rete sociale. Di partenza questo servizio è nato sul web, e dunque qualsiasi azione viene eseguita su web browser: l'utente va su di un URL prestabilito, richiede una pagina di login o registrazione, nel caso in cui sia loggato allora procede con l'aspetto social della rete. Ma la generazione delle pagine web viene gestita da OSN che si trova a gravarsi ulteriormente di compiti. E' proprio questo aspetto della rete sociale che verrà maggiormente trattato nella tesi corrente.

- **BE: RMS**
  Il **RMS** (abbreviativo di **Rule Manager Service**) è un modulo destinato a fare da **proxy**, ovvero da intermediario mascherando l'identità di chi effettua le richieste, tra il CS e il KMS.
  Queste ultime richieste consistono nell'ottenere i messaggi che contengono i dati relativi ai segreti dell'utente che forniranno la capacità allo user di creare messaggi univoci a se soltanto, in modo da evitare attacchi esterni. RMS si occupa dunque di fornire supporto per la memorizzazione dei dati passati dal client, senza però tenere traccia effettiva della loro posizione di salvataggio, stabilendo un collegamento tra il richiedente (ovvero lo user che effettua la richiesta e che viene salvato dal modulo tramite un ID fittizio) e il modulo che si occupa di ottenere e rintracciare le risorse che vengono salvate sul database esterno: questa è una comunicazione unidirezionale tra RMS e PFS.

- **BE: KMS**
  Il **KMS** (abbreviativo di **Key Manager Service**) è il modulo invece dedito a due compiti di vitale importanza nella rete sociale: il primo è il salvataggio diretto delle risorse presso il database, costituendo l'effettivo servizio di upload e download dell'intera rete; il secondo invece (e più importante) è quello di dover generare e gestire le chiavi asimmetriche utilizzate nel sistema, ma non del loro utilizzo per criptare i messaggi. Questa è una delle parti più delicate e che richiedono che tale servizio sia totalmente sicuro, per questo viene implementato in un'autorità di certificazione.
  Un'**autorità di certificazione**, per intendersi, è un soggetto esterno di fiducia che viene abilitato a distribuire certificati digitali tramite procedure standard e regolamentate. Nel nostro caso ciò che viene distribuito, come detto già precedentemente, sono una coppia di chiavi asimmetriche per ogni utente che ne fa richiesta: il KMS oltre a tale distribuzione prende la responsabilità di tenere traccia delle chiavi pubbliche in modo tale da essere reperibili da tutti.

- **BE: PFS**
  PFS (abbreviativo di **Path Finder Service**) è l'ultimo modulo del back-end e si occupa di controllare le regole di controllo d'accesso per le risorse, ovvero si preoccupa di rintracciare nel grafo il collegamento che collega il possessore del dato digitale al requestor della richiesta, confrontando poi questo legame con quelli impostati dall'utente e restituendo l'accesso o la negazione dello stesso.

3.2 Frameworks

Seppur la nostra decisione di effettuare un reverse engineering sia posta a "cancellare" l'architettura corrente per riportarci ad un livello di sviluppo minore, è necessario comunque effettuare uno studio dell'architettura corrente in modo tale da ottenere un approccio in grado di replicare la struttura in atto e non dimenticare alcuna parte. Nella porzione di progetto interessato vi è solamente la presenza di un unico framework: **Spring** e **AngularJS**.

3.3 Spring

Spring è un framework dedito allo sviluppo di applicazioni tramite svariati moduli assemblati in modo tale che, nel caso in cui non sia richiesto, non vengano sfruttati tutti se non quelli strettamente necessari e richiamati ai fini dell'applicazione stessa.

La struttura che deve seguire un progetto Spring viene fortemente influenzata da quali moduli vengono richiesti, nel caso di DSNProject questi:

- **DAO**, **Data Access Object** è un modulo di spring creato per facilitare la lavorazione di grandi mole di dati (salvati solitamente in database locali o esterni)
- **AOP**, **Aspect Oriented Programming** è il modulo che si occupa di suddividere le diverse parti del progetto Spring in sottomoduli che devono occuparsi di specifici punti, ogni punto lavorato viene gestito da una "Factory" che si occupa di lavorare dei dati in ingresso e produrre risultati che vengono definiti proprio da questo modulo.
- **Web**, modulo che si collega alla parte MVC (Model View Control) che spiegheremo anche successivamente ma con AngularJS.
- **Core**, modulo centrale che si occupa di far funzionare il framework e delle DI (Dependency Injection) ovvero delle parti di codice che non sono eseguite dalla JVM ma dal Core del framework.

Gran parte di questi moduli, è facile intendere, vengono utilizzati dal server per lo sviluppo del sistema OSN che si prende in carico di gestire molteplici compiti all'interno della rete e che dovrebbero essere smantellati.

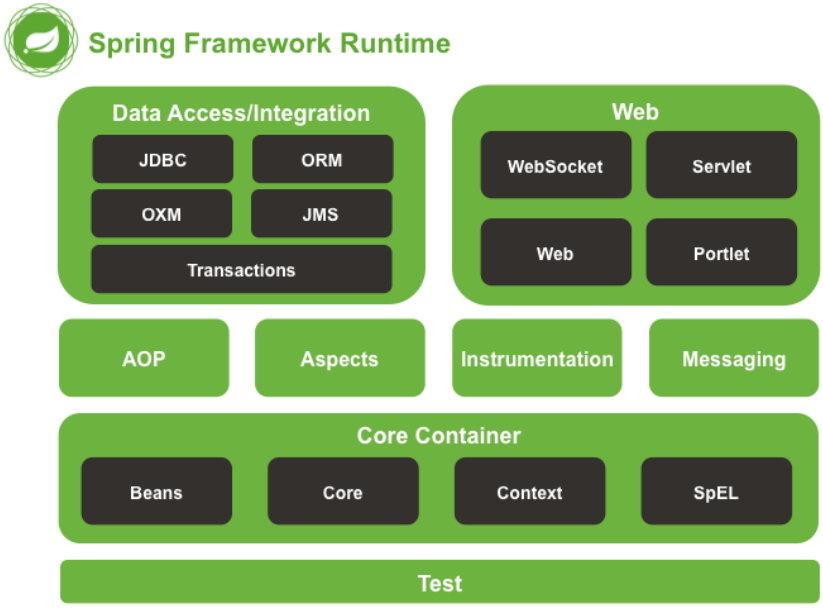

Figura 3.3.1: Rappresentazione grafica dei moduli appartenenti all'architettura Spring

Inoltre purtroppo, come si può notare, la parte Web è generata da lato server e, come già prima riscontrato, il client si trova nella situazione di non poter effettuare alcuna comunicazione in JS in modo sicuro, men che meno se viene distaccato dal server. Il problema opposto si verifica invece lasciando questa integrazione sul server: questo è a conoscenza delle procedure che il client esegue e, potenzialmente, potrebbe essere in grado di ottenere tutte le informazioni per ricavare le informazioni dal client e ottenere dunque accesso a ciò che volevamo proteggere sino dall'inizio.

3.4 AngularJS

Seppur collegato a Spring nel suo utilizzo, AngularJS è il secondo framework del progetto e prende l'incarico di come elaborare e mostrare la pagina all'utente finale, prendendosi l'incarico di gestire ogni operazione per l'utente. Segue il modello di sviluppo MVC (**Model View Controller**), infatti possiede:

- **View**, che non sono altro che le pagine HTML che erano state generate precedentemente ma ora arricchite di codice generato da AngularJS a seconda dei comportamenti richiesti.
- **Controller**, ovvero una specie di collegamento tra operazioni sui vari oggetti arricchiti e i comportamenti (funzioni) a cui devono rispondere
- **Servizi (Model)**, ovvero le operazioni stesse che deve eseguire il codice richiamato, come ad esempio chiamate di AJAX

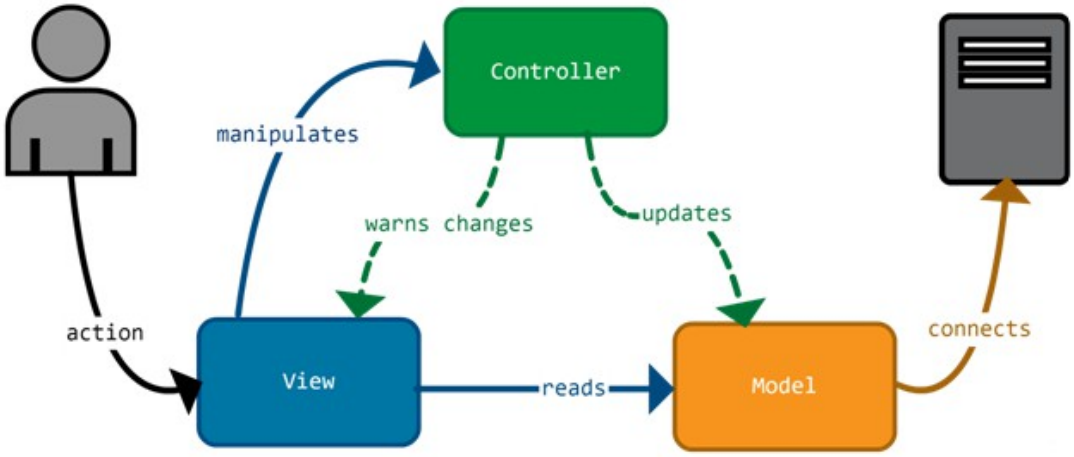

Figura 3.4.1: Rappresentazione grafica della struttura MVC usata anche dall'architettura AngularJS

4

Android

Figura 4.0.1: Logo di Android

4.1 Breve storia

La realtà di Android, nell'ottobre del 2003, era molto lontana da quella che conosciamo. I suoi quattro creatori (Adry Rubin, Rich Miner, Nick Sears e Chris White) inizialmente pensavano di creare un sistema operativo avanzato per fotocamere digitali ma, ben presto, si sono accorti che il solo traguardo sarebbe stato troppo raggiungibile e semplice da realizzare per le loro capacità. Così insistendo con gli investitori, nel 2005, Google acquisì la società che si limitò a definirsi il lavoro svolto come "software per telefoni". Fino al 2007 lo sviluppo del sistema operativo procedeva a rilento: lo sviluppo dell'hardware touchscreen aveva messo in difficoltà i progressi fatti sino a quel momento e l'entrata in campo di Apple aveva creato una vera e propria competizione nel campo: i touchscreen non vennero considerati sino al 2008 e lo sviluppo del sistema operativo fu prettamente realizzato con in mente l'idea di bottoni fisici.
Fu solamente ed effettivamente nel 2008 che Android, in un vero e proprio sprint, riuscì a fornire una serie di update capaci di portare il semplice sistema operativo a un livello assurdamente competitivo: gli update più importanti, che delimitavano una nuova versione del OS, vennero basati su nomi di dolci alfabeticamente.

Ecco un breve riassunto delle versioni rilasciate e dei loro principali contributi:

- **Android 1.6 – Donut**
  - *Ricerca rapida su internet integrata nella home*
  - *Dimensioni dello schermo variabili supportate*
- **Android 2.1 – Eclair**
  - *Google Maps su mobile*

- *Home personalizzata tramite widget e link a app*
- *Digitazione vocale*

- **Android 2.2 – Froyo**
  - *Comandi vocali*
  - *Hotspot Wi-Fi*
  - *Introduzione del compilatore Dilvik JIT basato su Java*
- **Android 2.3 – Gingerbread**
  - *API audio, grafica e memoria*
  - *NFC*
- **Android 3.0 – Honeycomb**
  - *Supporto tablet*
  - *Barre di sistema, simili al pc*
  - *Impostazione rapida (a cascata)*
- **Android 4.0 – Ice Cream Sandwitch**
  - *Migliorie di usabilità*
  - *Introduzione del controllo dei dati*
- **Android 4.1 – Jelly Bean**
  - *Notifiche interattive a cascata*
  - *Collegamento di funzionalità online (multi-account e Google Now)*
- **Android 4.4 – Kit Kat**
  - *Usabilità e grafica maggiore*
- **Android 5.0 – Lollipop**
  - *Design essenziale*
  - *Notifiche su schermata di blocco*
  - *Possibilità di collegarsi a più dispositivi (PC, Tablet, TV, Smartwatch)*
- **Android 6.0 – Marshmallow**
  - *Autorizzazioni per applicazioni*
  - *Risparmio batteria*

4.2 Una breve riflessione

Come si può notare, inizialmente il progetto era concentrato con grande forza e importanza sulla risoluzione dei problemi, migliorie a livello di prestazioni e l'introduzione di nuove funzionalità capaci di stare al passo con l'utente medio, in grado di interfacciarsi con i telefoni moderni basandosi su standard di hardware e software. Con l'avanzare nel tempo, però, ci si è spostati sul campo più grafico e di usabilità: una volta messe in campo le funzioni necessarie si è capito che quelle aggiuntive dovevano essere sviluppate esternamente con l'introduzione delle applicazioni. Si può notare questo cambio drastico dalla versione 4.0 in poi: l'utente avendo l'accesso a internet è diventato meno dipendente dal produttore stesso di telefoni o dal sistema operativo che lo “limita”, sono le applicazioni, gli aggiuntivi, che determinano le necessità di ogni persona che si vede diversa dalle altre.

Basandoci su questo ragionamento ne possiamo introdurre interessanti spunti per le conclusioni

sull'esperienza avuta.

4.3 Architettura di un progetto Android

Le App sono uno dei più rilevanti punti di forza dell'ultimo decennio e, seppure tra i corsi facoltativi dell'Università dell'Insubria di Informatica ve ne sia presente uno riguardante la programmazione su Android purtroppo, nel mio caso, sono dovuto ricorrere a internet e alle guide ufficiali di Android per imparare la struttura di questo framework su cui lavorare, sfruttando la grande quantità di informazioni e spiegazioni reperibili sul web. A seguito si trova un breve riassunto della struttura di un progetto Android, in modo da effettuare un'immersione nei concetti di base per capire, successivamente, il codice creato.

Quando si crea un progetto Android utilizzando Android Studio, una volta compilate le semplici o più complesse richieste (quali nome del progetto, supporti per altri linguaggi e l'impostazione della SDK minima da utilizzare per effettuare retro compatibilità), verrà creato il progetto che potrà essere visualizzato nel menù a tendina corrispondente.

Come si potrà subito notare, la suddivisione del progetto stesso è automatica e presenta due componenti principali: da una lato abbiamo una “semplice” cartella App mentre, dall'altra, abbiamo una raccolta di scripts sotto il nome Gradle Scripts.

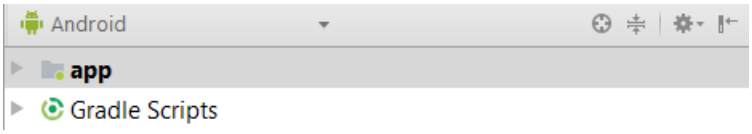

Figura 4.3.1: Esempio di un basilare progetto Android

4.4 Gradle Scripts

Gradle è un **sistema di sviluppo** che si occupa di **gestire** i **comandi a basso livello** (altrimenti usati dall'utente in linea di comando) in modo autonomo, sostituendo la precedente necessità di dover creare una propria APK e lasciando l'intero compito al sistema sviluppato da Google.

In questo modo, utilizzando Gradle, non è più necessario creare da zero varie cartelle e file necessari allo sviluppo perché questi verranno direttamente generati dagli script, senza contare che sarà proprio Gradle a preoccuparsi di effettuare la **compilazione** e il **deployment** degli stessi: infatti il sistema è basato su di una build della JVM e così tutti i file di sorgente vengono trattati dai tool appropriati e predefiniti, per poi essere compressi nella nostra specifica APK, pronta per essere distribuita.

Altro grande punto di forza di Gradle è la capacità di poter **implementare** qualsiasi **plug-in** disponibile con semplicità, potendo ottenere JAR anche da web per poi essere installati nel progetto direttamente come dipendenze e facilitare lo sviluppo dunque.

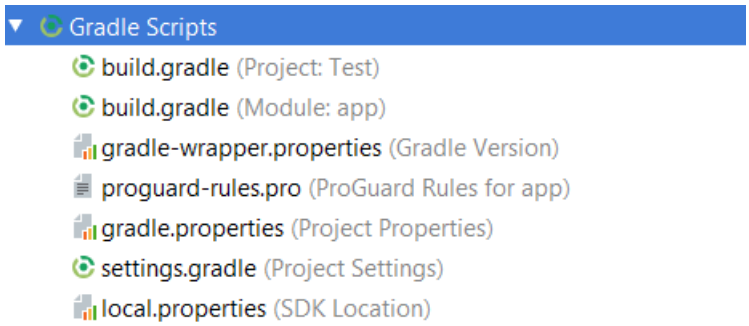

Figura 4.4.1: Esempio della parte Gradle di un progetto

4.5 App

Oltre alla parte Gradle del progetto possiamo trovare una parte definita semplicemente come **App**.

In realtà App è un semplice contenitore per tutte le altre cartelle che compongono il progetto.

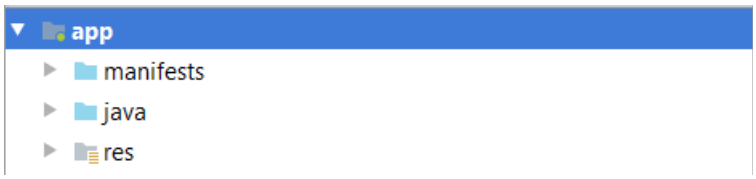

Figura 4.5.1: Esempio della parte App di un progetto

Come possiamo notare vi sono contenuti altri "moduli" dello stesso:

- **manifest**
- **java**
- **res**

manifest

manifest contiene il file **AndroidManifest.xml** che ha un'importantissima funzione: descrivere i **componenti** dell'applicazione che, a loro volta, includono attività, servizi, etc. informando il sistema di come ogni componente deve essere trattata. Oltre a ciò vi sono anche le liste dei **permessi** che l'applicazione richiede e sarà costretta ad ottenere per funzionare, oltre che al livello di **Android API** necessario per farla funzionare e alle **librerie** allegate.

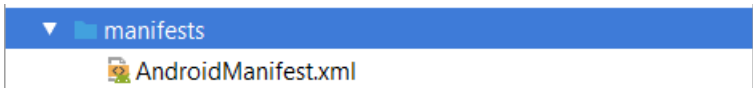

Figura 4.5.2: Esempio di manifest di un progetto

java

java contiene invece le cartelle che definiscono il progetto e gli ambienti di sviluppo su cui viene eseguito.
Per essere precisi java contiene tre package distinti:

- **applicazione**, ovvero il package di sviluppo e che effettivamente verrà utilizzato e rilasciato nella versione finale. Qui dentro possiamo trovare diversi package dello sviluppo e relative classi.
  L'applicazione può essere divisa principalmente in due categorie in base se estende una classe o meno:
  - non estendono activity, ovvero classi studiate per il funzionamento generale del progetto
  - estendono activity, ovvero quelle classi studiate per gestire la parte grafica del progetto, che gestiscono le operazioni e funzioni legate alla stessa
- **Instrumentation test**, ovvero il package utilizzato per effettuare i test avendo **controllo** sui **cicli di vita** e degli **eventi** dell'applicazione in real time.
- **Unit test**, ovvero il package utilizzato per **testare** una **singola unità separata** dalle proprie **dipendenze**. Solitamente l'unit test è abbinata al metodo instrumentation.

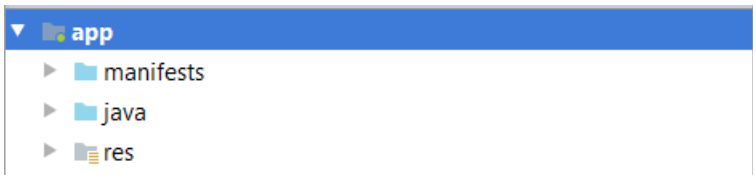

Figura 4.5.3: Esempio della parte App di un progetto

res

Abbreviativo di **resources**, è la cartella specifica alle **risorse**, per l'appunto, fornite all'applicazione. E' divisa in quattro sottocartelle principali di default, seppure ne siano disponibili molte di più:

- **drawable**, che è una collezione di immagini, o XML che eseguono la stessa funzione
- **layout**, l'intera collezione della parte grafica del progetto, che definisce l'interfaccia utente dello stesso.
- **mipmap**, collezione delle icone di varia dimensione, per adattarsi al dispositivo su cui si riproduce.
- **values**, lista di vari file XML che memorizzano ogni preferenza definita in fase di sviluppo. Esempi di questi file sono **string.xml**, che è la lista di ogni stringa definita per i componenti utilizzati nell'interfaccia utente, oppure **style.xml**, che è la lista degli stili da utilizzare per i vari bottoni, operazioni o transizioni.

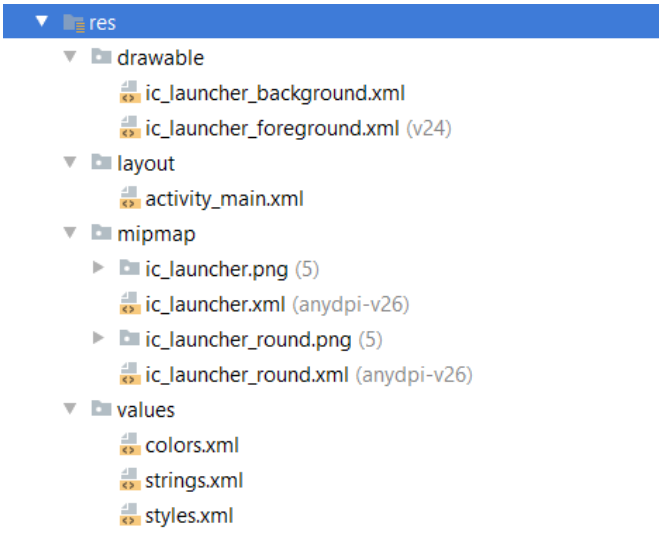

Figura 4.5.4: Esempio della parte App di un progetto

5

Crittografia

La base stessa del progetto è stata fondata sullo studio di una delle più importanti materie in fattore di protezione dei dati e che, senza, avrebbe reso inutile e impossibile in ogni modo la realizzazione stessa di questo progetto: la crittografia.

Per osservare il comportamento dei vari moduli e come questi ultimi riescono a scambiarsi i messaggi è necessario spiegare le tecniche crittografiche utilizzate, partendo dalle relative basi.

5.1 Tipologie

Esistono due categorie di cifratura per qualsiasi algoritmo sviluppato sino ai giorni nostri: simmetrica e asimmetrica.

Simmetrica
Nel caso di una cifratura simmetrica lo scopo dell'algoritmo è di cifrare il messaggio che vogliamo inviare utilizzando solamente una chiave per eseguire il lavoro di criptazione e relativa decriptazione. Il grande vantaggio di questa tecnica è la **velocità** delle comunicazioni oltre alla sicurezza, infatti tutte le operazioni sono semplici e di basso livello, ideali per messaggi di elevata dimensione, ma con un enorme svantaggio: entrambe le parti devono essere dotate della stessa **identica chiave** precedentemente scambiata tramite altro tipo di algoritmo, poiché non esiste modo alcuno di effettuare questa procedura utilizzando una tecnica simmetrica di principio. Se questo problema non fosse abbastanza, si aggiunge il fatto che, all'aumentare delle comunicazioni che utilizzano un algoritmo simmetrico, il numero delle chiavi da generare deve aumentare sempre più, finendo ad essere uno per ogni comunicazione tra due estremi che non hanno mai comunicato prima.

Ma ecco un grafico con relativa spiegazione del funzionamento:

1. E' possibile vi sia un messaggio di comunicazione iniziale per stabilire per ambo per parti quale algoritmo di criptazione utilizzare per la comunicazione corrente.
2. Dopo aver criptato il messaggio in chiaro (*Plaintext - P*) tramite la chiave simmetrica (*key*) stipulata precedentemente tramite altre vie, l'utente che ha iniziato la comunicazione invierà il messaggio criptato (*Chipertext – C*) su di un canale di comunicazione sicuro, facendo attenzione a non rivelare mai la propria chiave.
3. L'utente ricevente una volta ottenuto il messaggio in arrivo, utilizzerà la stessa chiave (*key*) per decriptare il *Chipertext C* e ottenere quindi, alla fine, il messaggio in chiaro che solo lui (e chiunque altro possiede tale chiave) può leggere.

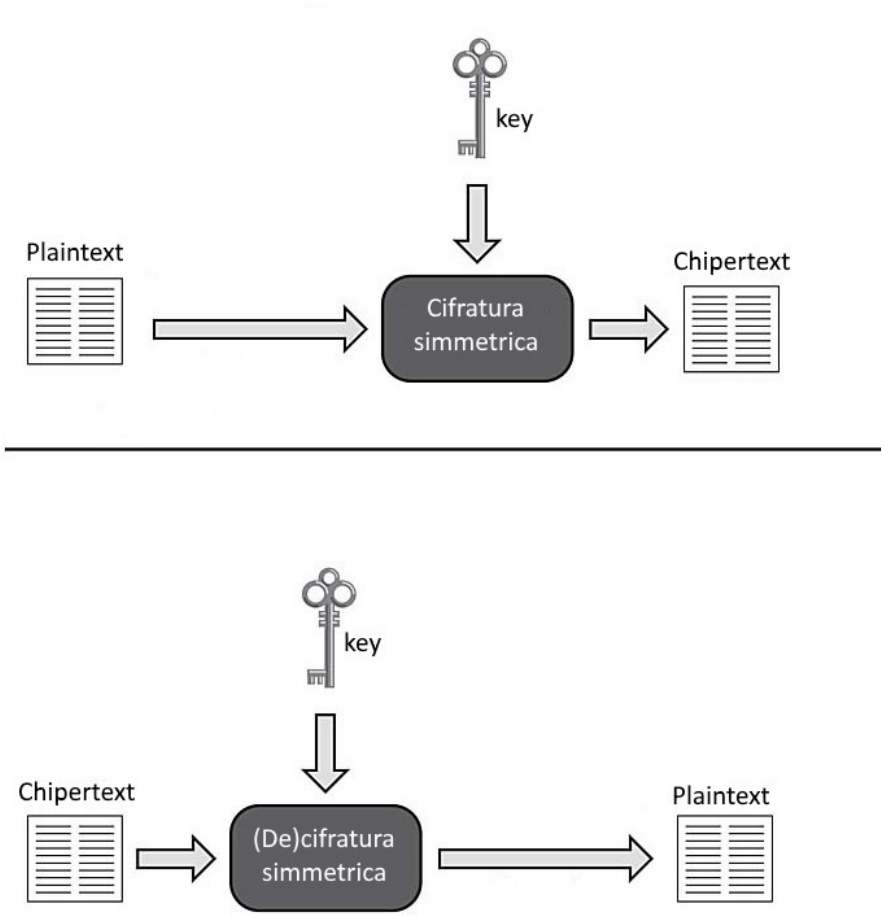

Figura 5.1.1: Esempio generico di cifratura simmetrica: Criptazione e decriptazione

Asimmetrica

La cifratura asimmetrica permette di cifrare un messaggio che potrà essere decodificato solamente dalle parti che comunicheranno utilizzando una **coppia di chiavi** (rispettivamente chiamate chiave **pubblica** e **privata**) che oltre a effettuare la semplice codifica permette di garantire che solamente il destinatario otterrà i messaggi a lui inviati. Il grande vantaggio di questa tecnica, opposto a quello della cifratura simmetrica, è la possibilità di poter scambiare messaggi senza effettuare in precedenza alcun tipo di scambio di chiavi tramite altri canali di comunicazione privati o accordi presi in precedenza in altro modo. Altro lato molto importante è il fatto che, pur non conoscendosi, è possibile effettuare messaggi di scambi nel caso in cui le chiavi pubbliche, come vedremo successivamente, è reperibile da un servizio di gestione e salvataggio di chiavi. Inoltre, tra gli altri fatti, il numero di chiavi non cambia ed è di due per ogni utente coinvolto: la sua chiave pubblica e quella privata.

E' necessario spiegare il principio base della crittografia asimmetrica prima di procedere alla spiegazione del grafico.
Le due chiavi che vengono generate per ogni utente hanno funzioni diverse:

- **Chiave privata**, non viene rivelata a nessuno e la conosce solamente l'utente a cui è assegnata che la utilizzerà per decriptare i messaggi a lui inviati.

- **Chiave pubblica**, viene distribuita in modo tale che chiunque voglia inviare un messaggio ad un utente possa utilizzare questa chiave per criptare il messaggio da inviare poi all'utente selezionato tramite una funzione unidirezionale (ovvero che con le stesse procedure non può essere risolta). Solamente l'utente in possesso della chiave privata potrà decifrare e dunque leggere il contenuto del messaggio

.

Purtroppo la generazione e, specialmente, la risoluzione dei messaggi è molto **pesante** e **lenta**, ancor di più se vi si aggiunge un sistema di **autenticazione** del messaggio. Per autenticazione del messaggio si intende la capacità dell'utente che invia di utilizzare funzioni di **Hash** e tramite la propria chiave privata di generare una "**firma digitale**" che può essere assegnata solamente a lui stesso in modo univoco.

Ecco il funzionamento con relativo grafico:

1. La persona che vuole inviare il messaggio cerca nella repository rispettiva la chiave pubblica del destinatario. (*Kpud*)

2. Una volta trovata utilizza tale chiave per cifrare il messaggio in chiaro (*P - Plaintext*) tramite un algoritmo di criptazione asimmetrico. La procedura varia a seconda dell'algoritmo scelto e genererà il messaggio criptato. (*C – Chipertext*)

   2.1 E' possibile che venga richiesto un servizio di autenticazione, in questo caso l'utente utilizzerà la propria chiave privata (*Kprm*) per generare una firma digitale univoca a se stesso, allegandola al messaggio prima che questo venga criptato.

3. Ora il messaggio verrà inviato al destinatario attraverso un canale di comunicazione qualsiasi al destinatario.

4. Il destinatario dunque utilizzerà la propria chiave privata (*Kprd*) per decifrare il messaggio, in modo da ottenere il *Plaintext* originale, che solo lui (e chi l'ha generato) può leggere.

   4.1 Nel caso in cui era stata utilizzata una firma digitale è possibile confermare che colui che ha generato il messaggio non è un esterno utilizzando la chiave pubblica del mittente (*Kpbm*), ottenendola nello stesso modo iniziale, per poi applicare l'algoritmo e confermarne la provenienza.

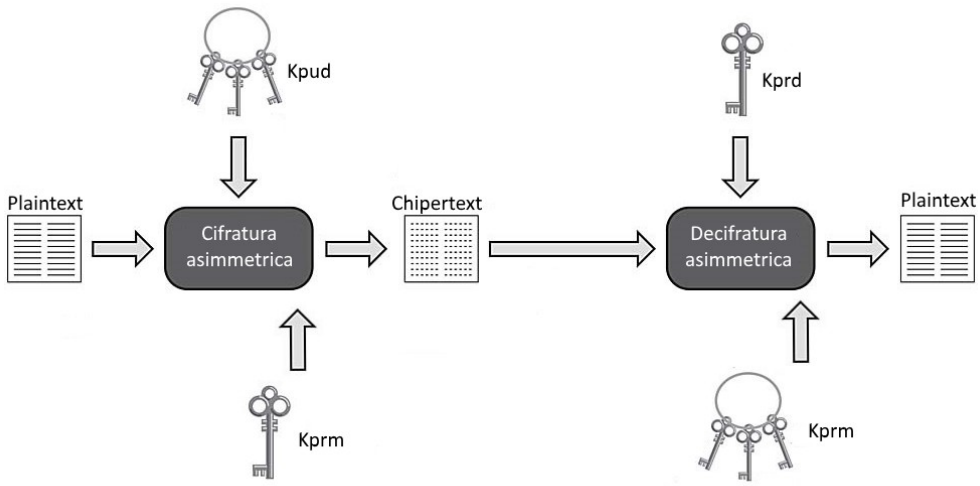

Figura 5.1.2: Esempio generico di cifratura asimmetrica comprensiva di Autenticazione

5.2 Algoritmi utilizzati

Ora che sono stati spiegati i principi utilizzati negli algoritmi della crittografia è necessario studiare i casi specifici utilizzati nel nostro progetto.
Si fa riferimento infatti, nella documentazione e nel relativo utilizzo, due algoritmi riportati a seguito che sono stati utilizzati per due ruoli diversi.

5.2a AES

AES, ovvero **Advance Encryption Standard**, è un algoritmo **block chiper** (ovvero di cifratura a blocchi) basato sull'implementazione dell'**algoritmo** di **Rijndael**.
Per **cifratura a blocchi** si intende che l'algoritmo utilizzato per la cifratura suddivide il messaggio in entrata (i suoi bit per l'esattezza) in sottosegmenti di una determinata lunghezza che verranno elaborati in quello stato. Questa operazione è tipica degli algoritmi di cifratura simmetrica.
AES è un algoritmo che sfrutta una rete a sostituzione e permutazione dotato di caratteristiche ottime a livello implementativo di risorse, sicurezza e complessità, inoltre è possibile impostare la grandezza dei blocchi per la cifratura che varia tra 128 e 256 bit.

La suddivisione dei bit in ingresso in blocchi che vengono disposti in una **matrice** 4x4 (nel caso di AES-128bit) subisce una serie di passaggi sempre identici:

1. **AddRoundKey**, ovvero il passaggio iniziale che combina ogni byte della tabella con la chiave di sessione, ovvero una chiave temporanea generata all'inizio del processo dalla chiave primaria tramite semplici operazioni decise dal **Key Scheduler** di Rijndael.

A questo punto le operazioni successive vengono effettuate più volte, in modo **ciclico** (chiamati **round**) a seconda sempre della dimensione (nel nostro caso 10 volte):

2. **SubBytes**, ovvero la sostituzione dei byte non lineari rimpiazzati utilizzando una tabella specifica, chiamata **S-box**. Le S-box, recentemente[14], sono state scoperte esser soggette ad attacchi tempistici, ovvero sul tempo di elaborazione necessario per processare l'informazione.
3. **ShiftRows**, ovvero lo spostamento dei blocchi di byte in **diagonale** rispetto alla matrice originale.
4. **MixColumns**, ovvero utilizzando un **polinomio fisso** si combinano ad una colonna della matrice, effettuando una trasformazione lineare invertibile.
5. **AddRoundKey**, ovvero quando si combina con l'operatore **XOR** la **chiave di sessione** con la nuova matrice, per ottenere il nuovo blocco.

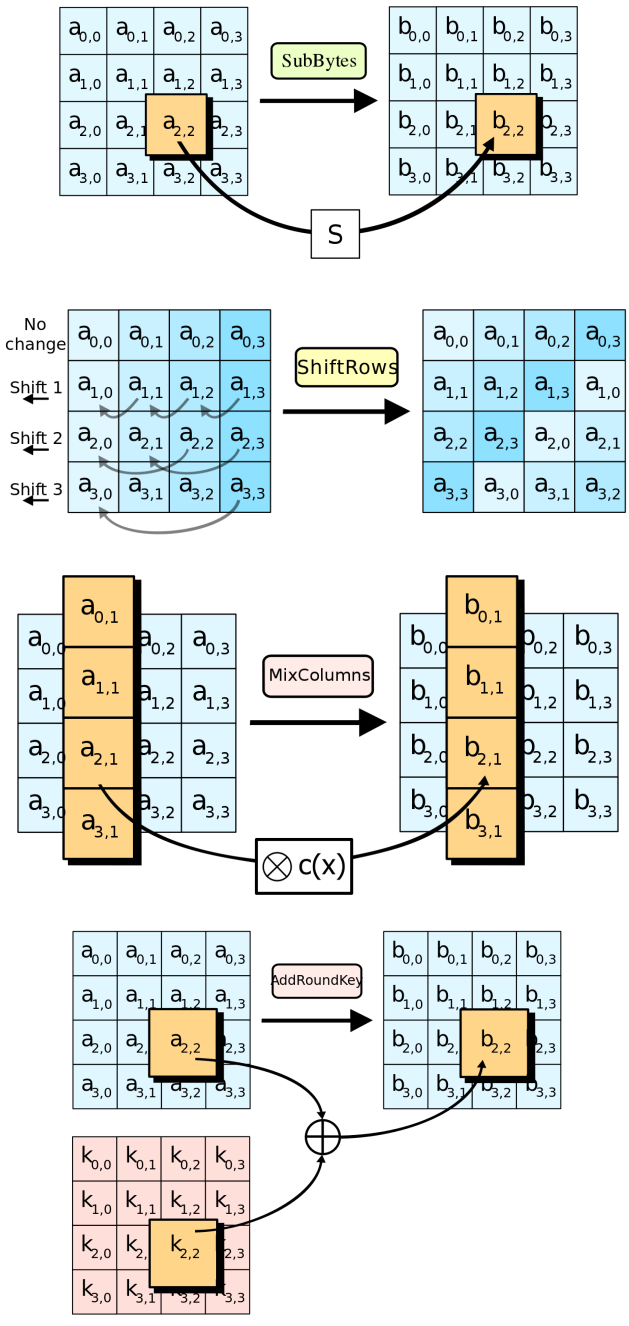

Figura 5.2.1a: Operazioni sulla matrice di AES

Tra le note interessanti si può sottolineare che i **punti 3 e 4** rispettano la **teoria di Shannon relativa alla confusione e diffusione** dell'algoritmo stesso, rendendo chiave e testo cifrato totalmente non correlati e escludendo qualsiasi tipo di relazione con l'alfabeto di criptazione.

Purtroppo AES è un algoritmo in uno stato incerto: seppur diffusissimo e utilizzatissimo è stato dimostrato che è debole ad attacchi XLS[15] seppure questi richiedano troppo tempo per essere elaborati allo stato attuale delle macchine, ma il vero problema è l'attacco correlato alla chiave sui blocchi 192 e 256 che, seppur sia solo teorico al momento, potrebbe in futuro prossimo garantire problemi di sicurezza dell'algoritmo AES.

5.2b RSA

RSA è l'acronimo degli inventori dell'algoritmo di crittografia asimmetrica, e utilizza le basi già sopra citate.
L'algoritmo utilizzato da RSA si basa sulla **fattorizzazione** in numeri primi della chiave e sull'elevata **complessità computazionale** che ne comporta.
Ecco i passaggi necessari per la generazione e codifica:

1. Vengono scelti a caso due numeri primi p e q tali da garantire la sicurezza stessa (solitamente superiori alle 300 cifre, in modo da ottenere numeri superiori ai 2048 bit, vedi fine per motivazioni), che saranno segreti e conosciuti solo da chi li ha generati.
2. Si calcola il prodotto ($n = pq$) e questo viene definito come modulo.
3. Si sceglie un esponente pubblico e che sia coprimo con il valore $z = (p-1)(q-1)$ e *minore* di quest'ultimo.
4. Si calcola l'esponente privato d a questo punto tale che $ed \equiv 1$ (che è definibile come *e congruo a 1 modulo z*)

A questo punto si ottengono due chiavi: quella **pubblica** che è composta dalla coppia *(n, e)* e quella **privata** che è composta da *(n, d)*.
Il collo di bottiglia di questo algoritmo che non permette l'inversione delle operazioni e dunque scoprire la chiave privata è la mancanza del numero z, di cui è necessaria la fattorizzazione per essere scoperto, operazione complessa e decisamente pesante.
Per cifrare un *messaggio P* si esegue la seguente operazione: $P^e \pmod n = C$, ovvero il **messaggio cifrato**; mentre per **decifrarlo** si esegue questa: $C^d \pmod n = P$.
Tutto ciò comunque si basa sull'assunzione, definita **RSA assumption**, di un fattore non dimostrato: $\sqrt[e]{c} \bmod n$ è **computazionalmente intrattabile**.
Per quanto riguarda invece la **firma digitale**, non è altro che il **digest dell'hash** del messaggio stesso: è l'applicazione di una **funzione iniettiva** non invertibile che prende “parti” ovvero blocchi di bit e li converte in una stringa di lunghezza arbitraria inferiore, restituendo il risultato, ovvero il digest.

E' inoltre stato definito come **standard** che le chiavi RSA generate devono essere almeno di **2048 bit**, visto che utilizzandone di più piccole rende ogni utente possibilmente suscettibile ad **attacchi bruteforce**[atk1].

6

Architettura e implementazione

L'architettura dell'applicazione è varia e, per permettere di essere compresa velocemente e con maggiore tracciabilità e semplicità, si è effettuata una divisione in package delle varie parti che la costituiscono.

Tutto il codice, anche quello citato, è reperibile presso il seguente link: SocialCloudAPP .

6.1 Modelli

Il primo package da trattare è sicuramente quello dei Model: qui è possibile trovare tutte le classi che rappresentano i principali oggetti utilizzati nell'applicazione e che sono trattati nelle varie comunicazioni.

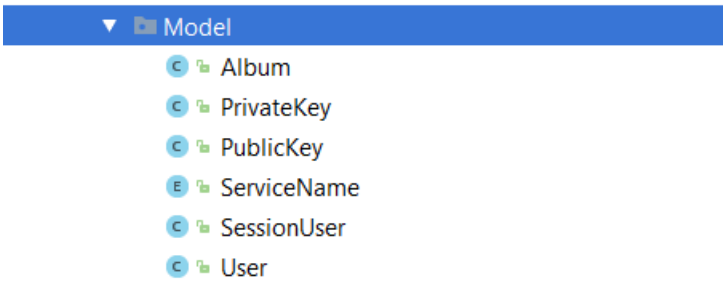

Figura 6.1.1 Package Model

User

E' la classe che descrive un utente all'interno della rete sociale, che sia l'utente che si collega alla rete oppure un utente creato per memorizzare (temporaneamente) le informazioni ottenute in risposta da un'altra procedura. L'utente non ha più collegate a se le informazioni relative alla propria chiave pubblica o privata, in quanto queste ultime vengono sempre richieste al server tramite messaggi per eseguire le procedure di cifratura/decifratura. Altro dato *non presente* nella classe utente è quella della **passphrase**: ogni inserimento di questa importantissima informazione è infatti manuale e si è cercati di tenere traccia il meno possibile di questo dato estremamente rilevante ai fini della sicurezza del sistema, in modo tale che anche se un utente riesce ad accedere ai dati applicativi, l'unico modo per ottenere la passphrase è di utilizzare *bruteforce* o *keyloggers*, problemi di cui tutti i sistemi di sicurezza sono afflitti e che non possono essere risolti.

La classe utente contiene i seguenti campi con relativi Setter e Getter:

- ***ID***, identificativo assegnato dal sistema di lato server
- ***firstname***, ovvero il nome dell'utente
- ***lastname***, ovvero il cognome dell'utente

- ***email***, indirizzo mail associato all'utente
- ***city***, ovvero la città dell'utente, che è un dato opzionale
- ***birth_day***, il giorno di nascita dell'utente
- ***pw***, la password dell'utente, utilizzata solo nel login, inoltre viene memorizzata in chiaro inizialmente il tentativo di accesso (dunque la password può esser giusta o errata) ma, successivamente alla richiesta, viene sovrascritta in codifica bcrypt
- ***photo***, ovvero un array di byte da essere codificato/modificato per ottenere il formato desiderato (che sia jpg/png per il server o bitmap per l'applicazione) rappresentante l'immagine di avatar dell'utente

Vi sono inoltre tre costruttori per le varie casistiche, diversi a seconda dei parametri in accettazione.

SessionUser

Per permettere di tenere traccia a livello dell'intera applicazione dell'utente utilizzato è stata scelta una soluzione semplice ma efficace: creare una classe contenente alcuni **parametri statici** tali che, in qualsiasi punto del sistema, questi possono essere ottenuti come riferimento per ottenere i dati dell'utente che effettua il login localmente sul dispositivo. Infatti, poiché il client mobile che si collega alla rete sociale è solamente uno per apparecchio, è possibile stabilire questa condizione che verrà sempre rispettata, in quanto esiste un singolo punto di modifica di questo valore, ovvero nel momento del login stesso. Per poter accedere con un altro account sarà necessario uscire sino alla schermata di login, dove potrà essere cambiato il valore effettuando un nuovo accesso.

I campi presenti in questa classe sono:

- ***sessionID***, ovvero l'identificativo della sessione che viene generata nel momento del login
- ***user***, ovvero le informazioni dell'utente loggato, che fanno riferimento alla classe User

Album

Oltre a tenere traccia degli utenti è necessario tenere traccia anche degli album, ovvero delle foto che sono state caricate da uno specifico utente con un determinato metatag per avvantaggiare un futuro algoritmo di ricerca. La classe Album prende il compito di tener traccia di queste risorse in alcune delle attività, per poter effettuare dopo la ricerca di tale risorsa e poterla scaricare in locale.

I campi che la compongono sono:

- ***IDalbum***, ovvero l'identificativo dell'immagine
- ***metaTag***, ovvero il tag associato all'immagine, che è una stringa solitamente che descrive in contenuto della stessa
- ***user***, ovvero l'utente che ha caricato tale risorsa
- ***fileName***, il nome del file, formato compreso

PublicKey e PrivateKey

Seppure esistano già delle classi di Java chiamate PublicKey e PrivateKey si è deciso di creare due classi proprie in grado di fornire un metodo di salvataggio temporaneo di queste risorse con piccole aggiunte che, nel futuro, potrebbero giovare ulteriormente la creazione del sistema.

Per fare un esempio, nella classe **PublicKey** è presente un campo, oltre a quello del modulo ed esponente, che descrive a quale servizio è associata la chiave. Idealmente potrebbe essere possibile salvare questi dati in modo tale da essere riutilizzati, con l'unica sconvenienza nella possibilità di un cambio di valori delle chiavi pubbliche e private del servizio a cui si fa richiesta durane la sessione dell'utente stesso.

Campi di PublicKey:

- ***service***, ovvero il servizio utilizzato *(OSN, RMS, KMS o PFS)*
- ***modulus***, ovvero il modulo per la decriptazione RSA della chiave pubblica
- ***exponent***, ovvero l'esponente per la decriptazione RSA della chiave pubblica

PrivateKey non appartiene ad alcun modulo: la chiave privata ricevuta sarà sempre dell'utente loggato, e verrà trasmessa tramite una combinazione degli algoritmi di RSA e AES, dunque è molto simile alla sua controparte di Java.

Campi di PrivateKey:

- ***modulus***, ovvero il modulo per la criptazione RSA della chiave privata
- ***public_exponent***, ovvero l'esponente pubblico per la criptazione RSA della chiave privata
- ***private_exponent***, ovvero l'esponente privato per la criptazione RSA della chiave privata

ServiceName

Oltre alle classi normali è disponibile anche una **enum** (enumerazione), ovvero uno speciale set di valori che possono essere assunti da una variabile del tipo dell'enum creata.

Questa classe è stata creata con lo scopo di tenere traccia dei vari servizi della rete sociale e dei loro identificativi.

Valori che possono essere assunti: *OSN("Online Service Network"), RMS("Rule Manager System"), PFS("Path Finder Service"), KMS("Key Manager System").*

6.2 Algoritmi di criptazione e utilità di progetto

In questo package, nominato **Utility**, è possibile accedere alle implementazioni degli algoritmi di criptazione creati per il progetto oltre ad alcune classi create per effettuare operazioni utili a ridurre la duplicazione di codice ed avvantaggiare dunque qualsiasi modifica a livello dell'intera

applicazione.

Questi ultimi sono le classi che verranno riportate successivamente, spiegando la funzione che detiene ognuna di esse.

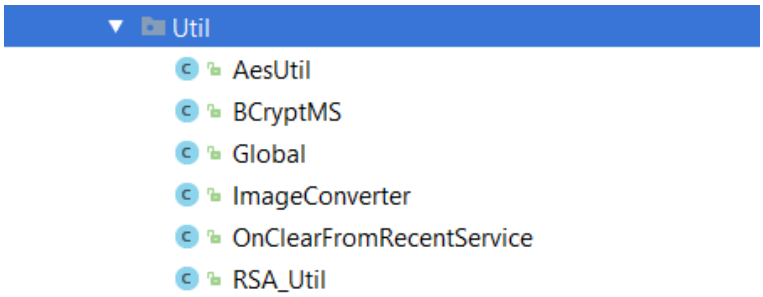

Figura 6.2.1 Package Util

Global

La classe Global è più definibile come una **repository statica** in grado di fornire **accesso globale** a tutti gli indirizzi del progetto, in modo tale da permettere una modifica veloce su variabili per evitare che vengano hard-coded.

In altre parole questa classe è una serie di metodi statici che ritornano l'indirizzo del servizio corrispondente concatenato alla subdirectory a cui sarà effettuata la richiesta HTTP.

ImageConverter

La classe di ImageConverter permette la **conversione** delle **immagini** in ingresso in formato JSONArray a quello di array di byte e, successivamente, è possibile convertire l'immagine ricevuta in formato di array di byte in una immagine bitmap, formato consigliato per la riproduzione su dispositivi mobili.

OnClearFromRecentService

Questa classe, che estende la **classe astratta Service** di Android, esegue le funzionalità di registrare le operazioni eseguite sull'applicazione nel momento in cui viene lanciata la schermata home postuma al login. Infatti i Service di Android sono, come suggerisce il nome, **operazioni** a **lungo termine** eseguibili in background prive di interfaccia grafica che ascoltano ciò che succede all'applicazione a cui sono annessi. Nel nostro caso, l'utilizzo che viene effettuato, è di controllare che, nel momento procedente alla chiusura dell'applicazione tramite gestore delle applicazioni integrato di Android, venga effettuato l'operazione di logout della sessione se presente.

Una volta elencati quelle classi che svolgono funzioni di agevolazione delle operazioni, è possibile parlare anche delle classi che implementano invece la codifica e decodifica tramite i diversi algoritmi utilizzati nel progetto.

BCryptMS

La classe **BcryptMS** implementa una versione **OpenBSD** di **Blowfish**. L'utilizzo di questa classe

avviene per una singola comunicazione al server nel momento del controllo della corrispondenza tra la password in input e quella ricevuta, rendendo l'identificazione un'operazione lato-client, fatto insicuro. Nel funzionamento di questo tipo di identificazione viene utilizzato un confronto tra la versione criptata della password presente nel modulo OSN e quella inserita dall'utente utilizzando l'algoritmo Blowfish, per l'appunto, che è un **algoritmo simmetrico** di **cifratura** a **blocchi** molto simile a AES, che con il tempo lo ha sostituito, in quanto quest'algoritmo (ideato inizialmente per sostituire DES) utilizza blocchi di 64bit che si sono rivelati con il tempo essere una misura debole contro gli attacchi crittografici (**birthday attacks e known-plaintext attacks**).

AESUtil

La classe AESUtil è stata studiata per implementare l'algoritmo **AES in** codice **Java**, effettuando alcuni cambiamenti per poter accettare comunicazioni pari a quelle su Javascript.

Per effettuare la criptazione è necessario creare un'istanza della classe e passare i dati richiesti (*IV, salt, Parola di codifica (es. Passphrase) e Messaggio in chiaro*) in una seconda chiamata per effettuare criptazione o decriptazione.

```
//Utility class made to implement the AES cryptography algorithm.
public class AesUtil {

    ...

    /**
     * Constructor for variable sizes
     * @param keySize           Size of the key that needs to be generated (2^n)
     * @param iterationCount    Number of iteration needed to be done
     */
    public AesUtil(int keySize, int iterationCount) {
        this.keySize = keySize;
        this.iterationCount = iterationCount;
        try {
            this.cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
            throw fail(e);
        }
    }

    /**
     * Constructor with DEFAULT_KEYSIZE & DEFAULT_ITERATIONCOUNT
     */
    public AesUtil() {
        this.keySize = DEFAULT_KEYSIZE;
        this.iterationCount = DEFAULT_ITERATIONCOUNT;
        try {
            this.cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
            throw fail(e);
        }
    }

    /**
     * Method that encrypts an image using salt and iv, in combination with a passphrase
     * @param salt          salt used in the AES algorithm
```

```
     * @param iv            IV used in the AES algorithm
     * @param passphrase    passphrase input by the user
     * @param plaintext     text needed to be converted
     * @return              Returns an encrypted string
     */
    public String encrypt(String salt, String iv, String passphrase, String plaintext) {
        try {
            SecretKey key = generateKey(salt, passphrase);
            byte[] encrypted = doFinal(Cipher.ENCRYPT_MODE, key, iv, plaintext.getBytes("UTF-
8"));
            return base64(encrypted);
        }
        catch (UnsupportedEncodingException e) {
            throw fail(e);
        }
    }

    /**
     * Method that decrypts an image using salt and iv, in combination with a passphrase
     * @param salt          salt used in the AES algorithm
     * @param iv            IV used in the AES algorithm
     * @param passphrase    passphrase input by the user
     * @param ciphertext    text that needs to be decrypted
     * @return              Returns a decrypted string
     */
    public String decrypt(String salt, String iv, String passphrase, String ciphertext) {
        try {
            SecretKey key = generateKey(salt, passphrase);
            byte[] decrypted = doFinal(Cipher.DECRYPT_MODE, key, iv, base64(ciphertext));
            return new String(decrypted, "UTF-8");
        }
        catch (UnsupportedEncodingException e) {
            throw fail(e);
        }
    }

    /**
     * Method that applies the AES algorithm on the text, both ways
     * @param encryptMode   Mod of operation for the Cipher
     * @param key           Key generated by the combination of salt and passphrase
     * @param iv            Initialization Vector
     * @param bytes         Array of bytes created from the original text
     * @return              Array of bytes resulted from applying the AES algorith on bytes
     */
    private byte[] doFinal(int encryptMode, SecretKey key, String iv, byte[] bytes) {
        try {
            cipher.init(encryptMode, key, new IvParameterSpec(hexStringToByteArray(iv)));
            return cipher.doFinal(bytes);
        }
        catch (InvalidKeyException
                | InvalidAlgorithmParameterException
                | IllegalBlockSizeException
                | BadPaddingException e) {
            throw fail(e);
        }
    }

    /**
     * Generation of the Key for the AES PBKDF2 with Hmac SHA1
     * @param salt          salt used in the AES algorithm
     * @param passphrase    passphrase input by the user
```

```
     * @return              Common key used for both encryption and decryption
     */
    private SecretKey generateKey(String salt, String passphrase) {
        try {
            SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
            KeySpec spec = new PBEKeySpec(passphrase.toCharArray(), hexStringToByteArray(salt),
iterationCount, keySize);
            return new SecretKeySpec(factory.generateSecret(spec).getEncoded(), "AES");
        }
        catch (NoSuchAlgorithmException | InvalidKeySpecException e) {
            throw fail(e);
        }
    }

    ...
}
```

RSA_Util

La classe RSA_Util è studiata invece per implementare l'algoritmo **RSA in** codice **Java**, ottenendo in ingresso i parametri della chiave pubblica nel caso di criptazione o di quella privata nel caso di decriptazione dove, tramite l'algoritmo standard utilizzato, modificato per generare e ottenere messaggi esadecimali in carattere UTF-8 come per Javascript, sarà possibile ottenere il messaggio elaborato di ritorno alla corrispettiva funzione.

```
//Class that implements the RSA Algorithm
public class RSA_Util {

   ...

    /**
     * Encryption of a message using the modulus and public exponent of the receiver
     * @param modulus   modulus, base 16
     * @param exponent  public exponent, base 16
     * @param rawData   data that needs to be converted
     * @return          a string encrypted using RSA
     */
    public static String encryptPublic(String modulus, String exponent, String rawData){
        KeyFactory keyFactory;
        String encryptedResult = null;
        try {
            keyFactory = KeyFactory.getInstance("RSA");

            //Decode from string hex (exp and mod) to byte[]
            byte[] crypted_modulus = new byte[256];
            BigInteger cryptedmod = new BigInteger(modulus, 16);
            if (cryptedmod.toByteArray().length > 256) {
                for (int i=1; i<257; i++) {
                    crypted_modulus[i-1] = cryptedmod.toByteArray()[i];
                }
            } else {
                crypted_modulus = cryptedmod.toByteArray();
            }
```

```
            byte[] crypted_exponent = new byte[256];
            BigInteger cryptedexp = new BigInteger(exponent, 16);
            if (cryptedexp.toByteArray().length > 256) {
                for (int i=1; i<257; i++) {
                    crypted_exponent[i-1] = cryptedexp.toByteArray()[i];
                }
            } else {
                crypted_exponent = cryptedexp.toByteArray();
            }

            //Generator of public keys
            RSAPublicKeySpec pubKeySpec = new RSAPublicKeySpec(cryptedmod, cryptedexp);
            RSAPublicKey key = (RSAPublicKey)keyFactory.generatePublic(pubKeySpec);

            Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
            cipher.init(Cipher.ENCRYPT_MODE, key);

            byte[] rawUTF8 = rawData.getBytes("UTF-8");

            //encryption
            byte[] encryptedRawData = cipher.doFinal(rawUTF8);
            encryptedResult = bytesToHex(encryptedRawData);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeySpecException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return encryptedResult;
    }

    public static String decrypt(String modulus, String private_exponent, String rawData){
        KeyFactory keyFactory = null;
        try {
            //Decoder from HexString to byte[]
            byte[] crypted_bytes = new byte[256];
            BigInteger cryptedmsg = new BigInteger(rawData, 16);
            if (cryptedmsg.toByteArray().length > 256) {
                for (int i=1; i<257; i++) {
                    crypted_bytes[i-1] = cryptedmsg.toByteArray()[i];
                }
            } else {
                crypted_bytes = cryptedmsg.toByteArray();
            }

            //generator of private keys
            keyFactory = KeyFactory.getInstance("RSA");
            RSAPrivateKeySpec prvKeySpec = new RSAPrivateKeySpec(new BigInteger(modulus, 16),
new BigInteger(private_exponent, 16));
            RSAPrivateKey key = (RSAPrivateKey) keyFactory.generatePrivate(prvKeySpec);
```

```
            Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
            cipher.init(Cipher.DECRYPT_MODE, key);

            byte[] cipherData = cipher.doFinal(crypted_bytes);
            return new String(cipherData);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeySpecException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        }
        return null;
    }

    ...
}
```

6.3 Servizi HTTP REST

Dopo una breve ricerca delle varie architetture in grado di fornire un metodo di comunicazioni tra client (applicazione android, nel nostro caso) e server e su come queste devono essere effettuate è stata presa la decisione di non utilizzare nessuna architettura implementativa ma di sfruttare i **servizi** di **connessione** e **comunicazione** predefiniti di **HTTP**, sfruttando l'architettura **REST** per le richieste.

Seppure questo possa apparire come una complicazione inutile nel progetto, nel controllo dei messaggi tra client e server si è venuta a creare una necessità di dover avere totale controllo sulle chiamate e su come i messaggi di risposta venivano ottenuti: alcune delle comunicazioni utilizzate nel progetto di base sono contrarie agli standard non scritti che si dovrebbero seguire per la comunicazione di un dato. Esempi di questi problemi sono la differenza tra l'intestazione della richiesta HTTP POST che viene eseguita (*application/json*) con i dati ricevuti (*singoli dati o text stream*) oppure imporre che indici di JSON siano scritti solo in minuscolo/maiuscolo/camel case.

E' per questo motivo che si è voluti utilizzare le comunicazioni standard **RESTful Web Service** (**Representational state transfert**) che impongono come queste richieste vengano effettuate tramite protocollo HTTP senza incaricare il compito a livelli superiori.

REST è definibile come un'**architettura logica per** le **comunicazioni HTTP** che implementano vari metodi (tra cui GET per l'ottenimento delle risorse e POST per la creazione, ovvero le due trattate nel progetto) in modo da definire le risorse utilizzate e la modalità in cui queste interagiscono tra di loro.

Oltre a tutto ciò è necessario inoltre specificare che ogni chiamata REST effettuata in Android, per definizione dal produttore del linguaggio, deve essere effettuata in una **AsyncTask**, ovvero un

"**incarico**" **asincrono** che non fermi il processo principale del sistema, in modo tale da garantire che non prenda il sopravvento dell'applicazione ma sia sempre e solamente un'operazione concorrente.

Seppure siano molte di più, nel progetto vengono utilizzati solo due tipi di operazioni HTTP 1.1 (versione che viene attualmente utilizzata generalmente nel web): **GET** e **POST**. Gran parte delle chiamate sono POST in quanto sono richieste che, dati dei parametri, ritornano in risposta dei risultati.

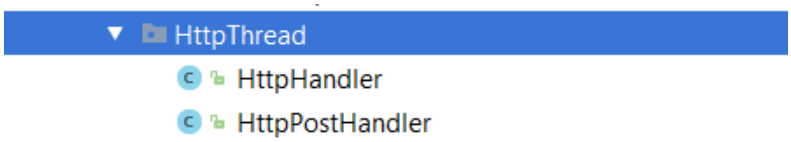

Figura 6.3.1 Package HttpThread

Al di sotto viene mostrato il codice per effettuare richieste di quest'ultimo tipo.

```
//Class that has the role of serving any POST request to all other servers, plus it needs to get
the response from them
public class HttpPostHandler {

    //Reference to the class itself
    private static final String TAG = HttpPostHandler.class.getSimpleName();

    /**
     * Void constructor
     */
    public HttpPostHandler() {
    }

    /**
     * Caller of the POST service via HTTP 1.1
     * NOTE: This should be the only method, but we need a "singledata" method too
     * @param url   URL that i send the request to
     * @param json  param data that i pass to the URL
     * @return      the response from the URL
     */
    public String makeServiceCall(URL url, JSONObject json) {
        String json_response = "{\"error\":\"error\"}";

        try {
            //Create the connection and set the input and output + request property and method
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setDoInput (true);
            conn.setDoOutput (true);
            conn.setUseCaches (false);
            conn.setRequestProperty("Content-Type","application/json");
            conn.connect();
            conn.setRequestMethod("POST");

            //Start of the POST input
            DataOutputStream printout;
            printout = new DataOutputStream(conn.getOutputStream ());
            printout.writeBytes(json.toString());
            printout.flush();
            printout.close();
```

```
            //Read the response code
            json_response="";
            //Read POST output
            InputStreamReader in = new InputStreamReader(conn.getInputStream());
            BufferedReader br = new BufferedReader(in);
            String text = "";
            while ((text = br.readLine()) != null) {
                json_response += text;
            }
        } catch (MalformedURLException e) {
            Log.e(TAG, "MalformedURLException: " + e.getMessage());
        } catch (ProtocolException e) {
            Log.e(TAG, "ProtocolException: " + e.getMessage());
        } catch (IOException e) {
            Log.e(TAG, "IOException: " + e.getMessage());
        } catch (Exception e) {
            Log.e(TAG, "Exception: " + e.getMessage());
        }
        return json_response;
    }

    /**
     * Caller of the POST service via HTTP 1.1
     * @param url           URL that i send the request to
     * @param singledata    single data that i pass to the URL
     * @return              the response from the URL
     */
    public String makeServiceCall(URL url, String singledata) {
        String json_response = "{\"error\":\"error\"}";

        try {
            //Create the connection and set the input and output + request property and method
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setDoInput (true);
            conn.setDoOutput (true);
            conn.setUseCaches (false);
            conn.setRequestProperty("Content-Type","application/json");
            conn.connect();
            conn.setRequestMethod("POST");

            //Start of the POST input
            DataOutputStream printout;
            printout = new DataOutputStream(conn.getOutputStream());
            printout.writeBytes(singledata);
            printout.flush();
            printout.close();

            //Read the response code
            json_response = "";
            //Read POST output
            InputStreamReader in = new InputStreamReader(conn.getInputStream());
            BufferedReader br = new BufferedReader(in);
            String text = "";
            while ((text = br.readLine()) != null) {
                json_response += text;
            }
        } catch (MalformedURLException e) {
            Log.e(TAG, "MalformedURLException: " + e.getMessage());
        } catch (ProtocolException e) {
            Log.e(TAG, "ProtocolException: " + e.getMessage());
```

```
        } catch (IOException e) {
            Log.e(TAG, "IOException: " + e.getMessage());
        } catch (Exception e) {
            Log.e(TAG, "Exception: " + e.getMessage());
        }
        return json_response;
    }
}
```

6.4 Protocolli di comunicazione

Per poter descrivere al meglio i protocolli di comunicazione è necessario dimostrare il funzionamento di un protocollo ideale prima di procedere nelle descrizioni dei singoli casi.
Un **protocollo di comunicazione** in questo progetto Android è definibile come **tutte** le **operazioni svolte** da una classe che estende la classe AsyncTask in modo tale da poter effettuare una o più operazioni **REST** manipolando una serie di dati in ingresso per effettuare modifiche al sistema della rete sociale e, se necessario, restituire i dati ricevuti che sono richiesti dal sistema.
I protocolli, nell'effettivo della programmazione, sono contenuti in due package e sono divisibili a seconda del ruolo che svolgono nel progetto.
Da un lato troviamo le **Task**, ovvero i "lavori" che devono essere effettuati attraverso operazioni asincrone e costituiscono un contenitore effettivo per le **Operations**.

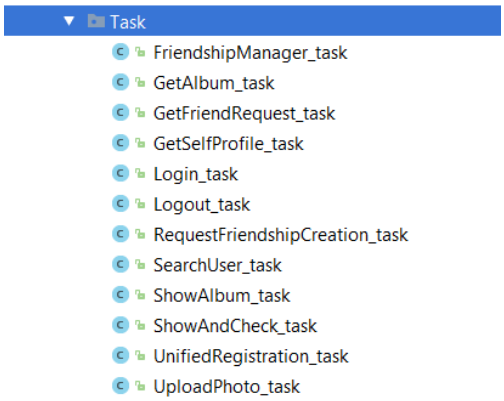

Figura 6.4.1 Package Task

Le Task, come già citato, implementano la classe **AsyncTask** in quanto devono effettuare una o più chiamate HTTP e forniscono la funzione di "**wrapper**" del contenuto incaricandosi l'accettazione dei parametri in ingresso e in uscita dalla chiamata asincrona. L'unico vero problema di questo sistema si riscontra quando si vuole che i risultati delle operazioni vengano passati alla classe chiamante. A questo punto è necessario che queste classi abbiano accesso a parametri pubblici implementati nelle classi chiamanti: questo non è esattamente vero perché grazie all'implementazione di un'interfaccia che sovrascrive la funzione della classe AsyncTask di termine delle operazioni è possibile incaricare un delegato della riposta asincrona (classe **AsyncResponse**) per ritornare questi valori nella classe chiamante tramite l'implementazione di un'interfaccia identica anche nella classe chiamante.
In questo modo il passaggio di risultati risulta totalmente isolato dal contesto del chiamante e ogni ambiente risulta isolato con semplici passaggi di risposte quando il metodo viene chiamato.

Prima sono state citate inoltre anche le **Operations**: queste si possono vedere come diverse implementazioni della classe astratta **FormatGenerator**, creata con l'intenzione di fornire una base implementabile da ogni operazione. FormatGenerator infatti possiede due parametri principali: una URL e un JSONObject.

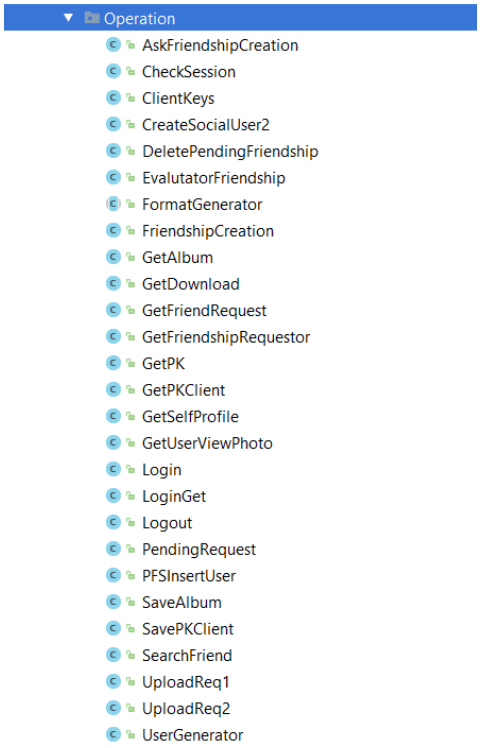

Figura 6.4.2 Package Operation

E' stato pensato, infatti, che ogni operazione che viene effettuata nei confronti di un modulo dovrebbe seguire un determinato schema: una chiamata REST deve essere indirizzata a un indirizzo specifico (l'**URL**) e contiene un **messaggio application/json**. E' stato successivamente riscontrato che non è questo il caso per ogni comunicazione, ma la soluzione non cambia: ogni implementazione di questa classe effettua un setup del proprio URL prendendolo dalla repository Global in modo autonomo. A questo punto, se vengono forniti dei parametri in ingresso, si procede alla creazione di un JSON sfruttando il metodo creato nella classe FormatGenerator. Questo potrà essere recuperato, insieme all'URL, utilizzando i metodi Getter di base.
Nel qual caso in cui non vi siano parametri in ingresso allora il messaggio da inviare non sarà un JSON ma un'altra stringa qualsiasi composta da un dato unico o da un messaggio criptato, il che renderà il parametro JSON inutilizzato per quei casi, ma con l'intenzione futura di standardizzare i messaggi. A seguito si trova il codice di FormatGenerator, esteso a tutte le classi del package Operations.

```
//Abstract class that creates a die to be used in all other operations
public abstract class FormatGenerator {

    protected static URL url;
    private JSONObject JSON;

    /**
     * Constructor
```

```
     */
    public FormatGenerator(){
        JSON = new JSONObject();
    }

    /**
     * Method to create a static URL from a string
     * @param url string of the complete URL
     * @return URL object of that string
     */
    protected static URL makeStaticURL(String url) {
        try {
            return new URL(url);
        } catch (MalformedURLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Getter of URL
     * @return URL
     */
    public URL getURL(){
        return this.url;
    }

    /**
     * Method to add to the JSON(Object) any Object with relative identifier (or key)
     * @param identifier String of the key/identifier
     * @param JSONData   Object that can be saved into a JSONObject
     */
    public void addObjectToJSON(String identifier, Object JSONData){
        try {
            JSON.put(identifier, JSONData);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    /**
     * Getter of the JSON
     * @return JSONObject
     */
    public JSONObject getJSON(){
        return JSON;
    }
}
```

Un esempio di una classe di Operations che estende la classe astratta FormatGenerator.

```
public class ExampleOperation extends FormatGenerator{

    public PendingRequest(Foo foo1, Foo foo2){
        super();
        url = makeStaticURL(Global.getExampleURL_path());
         Object elaboratedFoo = elaborate(foo2);
        addObjectToJSON("param1", foo1);
```

```
        addObjectToJSON("param2", elaboratedFoo);
    }
}
```

Di seguito si potrà trovare una lista di tutti i protocolli di comunicazione usati e relativi messaggi con risposte a cui fare riferimento. Seppure queste funzionalità esistevano di principio nel progetto iniziale (in un altro linguaggio, con una interfaccia diversa e con l'ausilio di un framework quale AngularJS) è necessario specificare che alcune di queste sono state rielaborate e modificate in modo tale da funzionare al meglio evitando riassegnazioni inutili oppure concatenando alcune funzionalità totalmente separate che avevano bisogno di duplicazione dei dati.
Anche se le comunicazioni sembrino ideali nella rappresentazione dei protocolli, in realtà vi sono alcuni dati inutilizzati passati tra le richieste di un modulo e l'altro, ma la modifica di queste da effettuarsi su tutti i moduli coinvolti: da un lato vi è un sovraccarico di dati mentre, dall'altra, vi è una possibilità di maggiore sviluppo nell'effettuare un numero minore di richieste allo stesso "prezzo" di comunicazione, se ottimizzato, o di implementare nuove funzionalità.
Viene lasciata scelta ai futuri sviluppatori la possibilità di un passaggio maggiore di parametri o il ritorno di dati precedentemente ignorati tramite questa applicazione, poiché si è già predisposti per questa eventualità.

Al di sotto verranno riportati tutti i protocolli creati e utilizzati nel progetto, implementando solamente il codice completo di quelli più complessi in modo da dimostrarne l'implementazione.

6.4.1 Registrazione di un nuovo utente

La registrazione di un nuovo utente deve avvenire su tutte e quattro le piattaforme del sistema in quanto su ognuno di essi deve essere registrata una risorsa diversa associata all'utente. Infatti per ogni utente sono necessarie una coppia di chiavi (pubblica e privata) utilizzate per la criptazione e decriptazione delle risorse nei successivi protocolli, senza contare che i servizi quali PFS e OSN devono tenere traccia di una lista degli utenti che sono registrati al servizio stesso.

La task che si occupa di tale operazione si chiama UnifiedRegistration_task.

1. L'applicazione Android (*APP*) ottiene i dati necessari dall'utente richiedendo una compilazione dei campi ed effettuando controlli riguardo la conformità di questi.
2. APP invia una prima chiamata POST a OSN contenente tutti i dati (nome, cognome, data di nascita, città, email, password e foto) fatta esclusione per la passphrase. Il risultato che ottengo è l'utente come inviato con incluso il suo identificativo unico, che lo rappresenta all'interno della rete sociale. (1)
3. Vengono effettuate due richieste POST ai servizi KMS e RMS in modo tale da ottenere le loro chiavi pubbliche per effettuare successivamente criptazione via RSA. (2 – 3)
4. A questo punto, viene effettuata una successiva richiesta POST a RMS dove richiedo la generazione di una coppia di chiavi pubblica e privata per l'utente, effettuando una criptazione incapsulata di un messaggio che verrà inviato successivamente a KMS dal modulo superiore, il quale genererà le chiavi e le rinvierà a RMS e, a sua volta, alla APP. (4)

5. La risposta di RMS sarà criptata via AES e la passphrase verrà utilizzata nella decriptazione da parte dell'utente in modo tale da ottenere la coppia di chiavi (pubblica e privata) dell'utente stesso. La coppia pubblica verrà resa disponibile senza criptazione a OSN ma quella privata invece dovrà essere criptata utilizzando la passphrase. (5)
6. E' necessario infine creare un utente anche su RMS sempre tramite chiamata POST in cui vengono salvati gli stessi dati ma criptati tramite la chiave pubblica di RMS e i parametri della cifratura AES utilizzata. (6)
7. Viene effettuata una ultima richiesta POST a PFS dell'aggiunta dell'ID dell'utente al sistema. (7)

1. $APP \rightarrow OSN : [name; surname; birthdate; city; photo; email; password]$

2. $APP \rightarrow OSN : ['RMS']$
   $OSN \rightarrow APP : [K^{+}_{RMS}]$

3. $APP \rightarrow OSN : ['KMS']$
   $OSN \rightarrow APP : [K^{+}_{KMS}]$

4. $APP \rightarrow RMS : [\ ID_{user};\ [\ ID_{user};\ iv;\ salt;\ keysize;\ iterationCount;\ passPhrase]]_{K^{+}KMS}$
   $RMS \rightarrow APP : [\ clientPublicExponent;\ clientModulus;\ clientPrivateKey;\ ]_{K^{+}S}$

5. $APP \rightarrow OSN : [\ IDuser;\ clientPublicExponent;\ clientModulus;\ [\ clientPrivateKey;]]_{K^{+}S}$

6. $APP \rightarrow RMS : [\ iv;\ salt;\ keySize;\ iterationCount;\ passPhrase;[\ clientPublicExponet,\ clientModulus,\ Iduser]^{+}_{KS}]^{+}_{KRMS}$

7. $APP \rightarrow PFS : [Iduser]$

```
/**
 * Asynchronous Task to make a registration in one go, by creating the user on all platforms (RMS
and KMS)
 */
public class UnifiedRegistration_task extends AsyncTask<HashMap<String, Object>, Void,
HashMap<String, Object>> {

    ...

    /**
     * Asynchronous operation that calls the handler to create a pending friendship
     * @param hashmap Hashmap<String, Object> where you can get the params from. It's always
one, so it can be found at index 0.
     *                Params contained:
     *                id: "user" -> object: an User object containing all data of the new user,
except for the ID
     *                id: "passphrase" -> object: an String containing the input passphrase
     * @return       Hashmap<String, Object> result of query.
     *               Params contained:
     *               id: "photo" -> object: byte array representing the photo
     */
    @Override
```

```
    protected HashMap<String, Object> doInBackground(HashMap<String, Object>... hashmap) {

        HashMap<String, Object> data = new HashMap<>();

        User new_user = (User)hashmap[0].get("user");
        String passphrase = (String)hashmap[0].get("passphrase");

        JSONObject json = null;

        //Save the user into the OSN for the first time, i receive an Autoincrement identifier
that will be the user's ID
        UserGenerator UG = new UserGenerator(new_user);
        String result = new HttpPostHandler().makeServiceCall(UG.getURL(), UG.getJSON());
        if(result==null){
            return null;
        }
        try {
            json = new JSONObject(result);
            new_user.setId_user((Integer) json.get("id"));
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //I perform the login to get the session going serverside
        LoginGet LG = new LoginGet();
        result = new HttpPostHandler().makeServiceCall(LG.getURL(), new_user.getEmail());

        //Getting the public key of KMS
        GetPK GPKkms = new GetPK();
        String service = null;
        String modulus = null;
        String exponent = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(GPKkms.getURL(),
ServiceName.KMS.toString()));
            service = (String) json.get("service");
            modulus = (String) json.get("modulus");
            exponent = (String) json.get("exponent");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        PublicKey KMS_pk = new PublicKey(service, modulus, exponent);

        //Getting the public key of RMS
        GetPK GPKrms = new GetPK();
        service = null;
        modulus = null;
        exponent = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(GPKrms.getURL(),
ServiceName.RMS.toString()));
            service = (String) json.get("service");
            modulus = (String) json.get("modulus");
            exponent = (String) json.get("exponent");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        PublicKey RMS_pk = new PublicKey(service, modulus, exponent);

        //Generating the new user for KMS, by sending a JSON like this one
```

```
        //var
message={"iduser":idUser,"iv":iv,"salt":salt,"keySize":keySize,"iterationCount":iterationCount,"p
assPhrase":passphrase};
        String IV = AesUtil.random();
        String salt = AesUtil.random();
        ClientKeys CK = new ClientKeys(new_user, KMS_pk.getModulus(), KMS_pk.getExponent(),
passphrase, IV, salt);
        String response = null;
        //TODO: MALFORMED E IOEXCEPTION
        JSONObject CKjson = CK.getJSON();
        response = new HttpPostHandler().makeServiceCall(CK.getURL(), CKjson);

        //the response i get is encrypted via AES and the salt + IV and passphrase i used
        AesUtil AES = new AesUtil();
        String decrypted = AES.decrypt(salt, IV, passphrase, response);
        String client_public_exponent = null;
        String client_modulus = null;
        String client_private_exponent = null;

        //Here i get the JSON result of the previous decryption that contains my public and
private key pairs
        PrivateKey PrK = null;
        try {
            json = new JSONObject(decrypted); //Decrypted = ""
            client_public_exponent = (String)json.get("client_public_exponent");
            client_modulus = (String)json.get("client_modulus");
            client_private_exponent = (String)json.get("client_private_exponent");

            PrK = new PrivateKey(client_modulus, client_public_exponent,
client_private_exponent);

        } catch (JSONException e) {
            e.printStackTrace();
        }

        //i need to encrypt my private key using the public key, so i can send it to RMS
        String cpe_encrypted = AES.encrypt(salt, IV, passphrase, PrK.getPrivateExponent());

        //i generate the message to save the pair key of the user to OSN for future retrieving
        SavePKClient SPKC = new SavePKClient(new_user.getId_user(), PrK.getPublicExponent(),
PrK.getModulus(), cpe_encrypted, salt, IV);
        new HttpPostHandler().makeServiceCall(SPKC.getURL(), SPKC.getJSON());

        AesUtil AES2 = new AesUtil();
        String IV2 = AesUtil.random();
        String salt2 = AesUtil.random();
        String OTPassphrase = AesUtil.random();
        //here instead i save the client key associated to the ID of the user to RMS via JSON
that needs to be encrypted
        //var clientPubKeyToRMS={"client_pub_exp": client_exponent, "client_mod":
client_modulus, "idu": idUser};
        JSONObject clientPubKeyforRMSJSON = new JSONObject();
        try {
            clientPubKeyforRMSJSON.put("client_pub_exp", client_public_exponent);
            clientPubKeyforRMSJSON.put("client_mod", client_modulus);
            clientPubKeyforRMSJSON.put("idu", new_user.getId_user());
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //encryption done via AES
```

```
        String encrypted_CPKforRMS = AES2.encrypt(salt2, IV2, OTPassphrase,
clientPubKeyforRMSJSON.toString());

        //generation of the symmetric key that is going to be used for the decryption and JSON
of the data
        JSONObject symmetricKey = new JSONObject();
        //var symmKey={
        //"iv": iv2,
        //"salt": salt2,
        //"keySize": keySize,
        //"iterationCount": iterationCount,
        //"passPhrase": x,
        //}
        try {
            symmetricKey.put("iv", IV2);
            symmetricKey.put("salt", salt2);
            symmetricKey.put("keySize", AesUtil.DEFAULT_KEYSIZE);
            symmetricKey.put("iterationCount", AesUtil.DEFAULT_ITERATIONCOUNT);
            symmetricKey.put("passPhrase", OTPassphrase);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Encryption via RSA using RMS public key
        String encrypted_symmKey = RSA_Util.encryptPublic(RMS_pk.getModulus(),
RMS_pk.getExponent(), symmetricKey.toString());

        //creation of the user on RMS
        CreateSocialUser2 CSU2 = new CreateSocialUser2(encrypted_CPKforRMS,
encrypted_symmKey.toString());
        new HttpPostHandler().makeServiceCall(CSU2.getURL(), CSU2.getJSON());

        //insertion of the ID of the user on PFS
        PFSInsertUser PFSIU = new PFSInsertUser();
        new HttpPostHandler().makeServiceCall(PFSIU.getURL(), new_user.getId_user().toString());

        data.put("user", new_user);
        return data;
    }

    ...
}
```

6.4.2 Login

Il login è un semplice tentativo di accesso utilizzando le credenziali di email e password per controllare se è presente nel sistema un utente che possiede tale caratteristiche. Purtroppo l'autenticazione della password è effettuata lato client. Seppure ciò non abbia senso è stata mantenuta questa specifica data dal progetto principale, seppure si richieda una modifica futura di questo sistema di autenticazione.

La task che si occupa di tale operazione si chiama Login_task.

Il login viene effettuato in due semplici richieste POST:

1. La prima richiesta è quella di controllare se nel sistema di OSN qualcuno possiede tra le proprie credenziali un indirizzo email come quello specificato. Nel caso vi sia, si ottiene in risposta parte delle informazioni dell'utente.

2. A questo punto viene effettuata una seconda richiesta all'OSN in cui si richiede di creare una sessione per l'utente, e il tutto verrà effettuato sul lato client.

1. *APP → OSN : [email; password]*
OSN → APP : [email; password; IDuser; name; surname]

2. *APP → OSN : [email]*

6.4.3 Ottenere il proprio profilo

Altra operazione trattata è quella per ottenere il proprio profilo dal sistema: seppure l'operazione è alquanto semplice è in questo punto, appena successivo al login, che viene effettuato il setup della variabile che ricorderà al sistema quale utente è loggato. Vi è infatti una sola unica chiamata a OSN per ottenere tramite l'IDuser i dati dell'utente.

La task che si occupa di tale operazione si chiama GetSelfProfile_task.

1. La richiesta viene effettuata a OSN tramite POST ottenendo in risposta le informazioni relative all'utente che vengono salvate in locale in una variabile statica disponibile per tutta l'applicazione. Inoltre si ottiene anche l'ID della sessione che era stata creata nel login.

1. *APP → OSN : [IDuser]*
OSN → APP : [IDuser; email; password; city; name; surname; birthDate; photo, session]

6.4.4 Ricerca di un altro utente

Per effettuare la ricerca di un utente è necessario effettuare una richiesta a OSN in cui viene inserito parte del nome o cognome dell'utente ricercato, in modo da effettuare una richiesta al database di OSN che restituirà una ricerca simile a quella utilizzata in SQL tramite l'operatore LIKE s%, dove s è la stringa da noi inserita per la ricerca. Una volta effettuata tale richiesta verrà mostrata una lista degli utenti che possiedono nel nome tali valori.

La task che si occupa di tale operazione si chiama SearchUser_task.

1. Richiesta a OSN di una stringa (totale o parziale del nome/cognome) da ricercare, il risultato sarà un array in formato JSON di tutti gli utenti che corrispondono ai valori dati in ingresso.

1. *APP → OSN : [namesurnamePartial]*
OSN → APP : [{IDuser(requested), name, surname, city}]

6.4.5 Ottenere le richieste di amicizia

Il metodo per ottenere le richieste di amicizia effettuate da altri utenti a chi effettua il login è tramite una singola chiamata a OSN che tiene traccia delle richieste di amicizia in sospeso nel proprio database.

La task che si occupa di tale operazione si chiama GetFriendRequest_task.

1. Richiesta a OSN costituita dall'identificativo dell'utente loggato, con risultato un array in formato JSON di tutti gli utenti che hanno effettuato la richiesta al richiedente.

1. *APP → OSN : [IDuser]*
OSN → APP : [{IDuser(requestor), name, surname, city}]

6.4.6 Gestione di una richiesta di amicizia

Quando si ottiene la lista delle richieste di amicizia è possibile scegliere se accettare o rifiutare tale richieste. Seppure inizialmente la gestione di queste operazioni era separata, è stato possibile unire le operazioni in un'unica sequenza. L'operazione iniziale consiste sempre nel controllo della sessione e, di seguito, si procede a una richiesta POST di cancellazione della richiesta in sospeso di amicizia tra il richiedente e l'utente corrente. Successivamente, a seconda della selezione effettuata tramite l'interfaccia grafica, verranno svolte più o meno operazioni: nel qual caso in cui sia stato scelto di rifiutare la richiesta di amicizia allora non verrà svolta più alcuna operazione e la funzione terminerà.
In alternativa bisognerà accettare la richiesta: ciò consiste in tre successive chiamate per ottenere il richiedente di tale richiesta, la chiave pubblica di PFS e inviare una richiesta di creazione di amicizia con quest'ultimo da parte dell'utente loggato.

La task che si occupa di tale operazione si chiama FriendshipManager_task.

1. Inizialmente viene effettuato un controllo sullo stato della sessione dell'utente, accertandosi che ne abbia una. Si riceve in risposta un token di sessione equivalente a quello di quella attiva, se la si ha. (1)
2. A seguito viene effettuata una chiamata di cancellazione della richiesta di amicizia in sospeso presente in OSN, inserendo il richiedente dell'amicizia e l'utente collegato al sistema di rete sociale. (2)

   Solamente se si accetta la richiesta verranno svolte le seguenti operazioni:

3. Ottenere il profilo del richiedente della ricerca di amicizia, ricercando suddetto utente tramite l'ID del richiedente che viene ottenuto dalla richiesta. La risposta ottenuta dal POST sarà un profilo parziale dell'utente. (3)
4. Si procede ad effettuare una chiamata di richiesta della chiave pubblica di PFS per effettuare una criptazione asimmetrica tramite RSA dei dati necessari per la creazione di un nuovo collegamento di amicizia nel sistema di PFS. (4)
5. Vengono criptati alcuni dati, quali l'ID dei due utenti, email, nome e cognome del richiedente e nome e cognome del richiesto (l'utente con l'accesso) tramite RSA con chiave PFS e inviato allo stesso servizio per la creazione di una nuova amicizia. (5)

1. *APP → OSN : [IDuser]*

$$OSN \rightarrow APP : [session]$$

2. $APP \rightarrow OSN : [IDuser(requestor), IDuser]$

3. $APP \rightarrow OSN : [IDuser(requestor)]$
$$OSN \rightarrow APP : [IDuser(requestor), email(requestor), name(requestor), surname(requestor)]$$

4. $APP \rightarrow OSN : ['PFS']$
$$OSN \rightarrow APP : [K^{+}_{PFS}]$$

5. $APP \rightarrow PFS : [IDuser(requestor), IDuser, email(requestor), name(requestor), surname(requestor), name, surname]_{K^{+}_{PFS}}$

```
/**
 * Asynchronous Task to manage any type or friendship request, that can be accepted or refused.
 * In both cases you need to do the same operations, except that after you need to save the
friendship if you accept it.
 */
public class FriendshipManager_task extends AsyncTask<HashMap<String, Object>, Void,
HashMap<String, Object>> {

    ...

    /**
     * Asynchronous operation that calls the handler manage the friendship request (both accept
and refuse)
     * @param hashmap Hashmap<String, Object> where you can get the params from. It's always
one, so it can be found at index 0.
     *                Params contained:
     *                id: "accept" -> object: a Boolean that tells if the request is accepted or
refused-
     *                id: "id_requestor" -> object: an Integer containing the ID of the user
that i have to link the friendship to.
     * @return        empty
     */
    @Override
    protected HashMap<String, Object> doInBackground(HashMap<String, Object>... hashmap) {

        HashMap<String, Object> data = new HashMap<>();

        Boolean isAccepted = (Boolean)hashmap[0].get("accept");

        Integer id_requestor = (Integer)hashmap[0].get("id_requestor");

        //Here I pick the user from the current session
        User current_user = SessionUser.getUser();

        //On both cases the session has to be checked and the friendship request needs to be
deleted from pending
        //Checking session
        JSONObject json;
        CheckSession CS = new CheckSession();
        String session = "";
        //The response from the POST request is a JSON containing a single identifier called
"session" that echos out the ID of my current session
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(CS.getURL(),
current_user.getId_user().toString()));
            session = (String)(json.get("session"));
        } catch (JSONException e) {
```

```
            e.printStackTrace();
        }

        //if the session is empty than i don't have one, i call a new exception
        if(session.length()==0){
            try {
                throw new EmptySessionException();
            } catch (EmptySessionException e) {
                e.printStackTrace();
            }
            return null;
        }

        //Deleting pending request
        DeletePendingFriendship DPF = new DeletePendingFriendship(id_requestor,
current_user.getId_user());
        new HttpPostHandler().makeServiceCall(DPF.getURL(), DPF.getJSON());
        //Response is void, cannot be handled

        //Here needs to be checked if the request is accepted. If not, then i don't do anything
else and skit to return
        if(isAccepted){
            //If the request is accepted then I trigger the procedure for a new friendship:
            //I ask the who is the friend i want to add and store all data in a new local user
            GetFriendshipRequestor GFR = new GetFriendshipRequestor();
            User requestor = new User();
            try {
                json = new JSONObject(new HttpPostHandler().makeServiceCall(GFR.getURL(),
id_requestor.toString()));
                requestor.setId_user((Integer) json.get("id"));
                requestor.setEmail((String) json.get("email"));
                requestor.setFirstname((String) json.get("firstname"));
                requestor.setLastname((String) json.get("lastname"));
            } catch (JSONException e) {
                e.printStackTrace();
            }

            //Getting the PFS keys and saving them
            GetPK GPKpfs = new GetPK();
            String service = null;
            String modulus = null;
            String exponent = null;
            try {
                json = new JSONObject(new HttpPostHandler().makeServiceCall(GPKpfs.getURL(),
ServiceName.PFS.toString()));
                service = (String) json.get("service");
                modulus = (String) json.get("modulus");
                exponent = (String) json.get("exponent");
            } catch (JSONException e) {
                e.printStackTrace();
            }
            PublicKey PFS_pk = new PublicKey(service, modulus, exponent);

            //Here I create the message that needs to be sent to PFS, it's a JSON that follow
the sequent format:
            //var
messagetoPFS={"idRequestor":id,"idOwner":id_acceptor,"emailRequestor":emailRequestor,"nameRequest
or":nameRequestor,
            //
"surnameRequestor":surnameRequestor,"nameSearched":self.user.firstname,"surnameSearched":self.use
r.lastname};
```

```
            JSONObject msgToPFS = new JSONObject();
            try {
                msgToPFS.put("idRequestor", requestor.getId_user());
                msgToPFS.put("idOwner", current_user.getId_user());
                msgToPFS.put("emailRequestor", requestor.getEmail());
                msgToPFS.put("nameRequestor", requestor.getFirstname());
                msgToPFS.put("surnameRequestor", requestor.getLastname());
                msgToPFS.put("nameSearched", current_user.getFirstname());
                msgToPFS.put("surnameSearched", current_user.getLastname());
            } catch (JSONException e) {
                e.printStackTrace();
            }

            //RSA with PFS public key the JSON and obtain an encrypted message
            String encrypetedMsgToPFS = RSA_Util.encryptPublic(PFS_pk.getModulus(),
PFS_pk.getExponent(), msgToPFS.toString());

            //Ask to create a new friendship between me and the user that made the request
            FriendshipCreation FC = new FriendshipCreation();
            new HttpPostHandler().makeServiceCall(FC.getURL(), encrypetedMsgToPFS);
        }

        //Data is empty
        return data;
    }

    ...
}
```

6.4.7 Effettuare richieste di amicizia

Oltre ad ottenere e gestire le richieste di amicizia è possibile effettuarle. Per poter procedere in tale operazione basterà possedere l'ID dell'utente che si intende aggiungere, in modo tale da richiedere la creazione di una nuova richiesta di amicizia che rimarrà in sospeso sinché non sarà confermata o rifiutata. Tale operazione è solitamente preceduta da un controllo riguardante la presenza di una richiesta di amicizia (confermata o in sospeso) con l'utente richiesto [vedi prossimo protocollo].

La task che si occupa di tale operazione si chiama RequestFriendshipCreation_task.

1. Precedentemente alla richiesta di creazione di una nuova amicizia viene effettuato un controllo riguardante lo stato della sessione dell'utente. La risposta è il token di sessione stesso, ripetuto. (1)
2. Richiesta a OSN costituita dall'identificativo dell'utente loggato e dell'utente a cui si vuole inviare la richiesta, una volta che viene effettuata non si riceve nessuna conferma dell'accettazione/rifiuto. (2)

1. *APP → OSN : [IDuser]*
   OSN → APP : [session]
2. *APP → OSN : [IDuser, IDuser(requested)]*

6.4.8 Controllo dello stato di un'amicizia e recupero dei dati dell'utente

Prima di effettuare qualsiasi tipo di richiesta di amicizia è necessario controllare se l'utente a cui la invio sia già mio amico o meno, oppure sussista una richiesta di amicizia in attesa di risposta. E' proprio per questo motivo che, dopo aver ricercato un utente, per decidere se mostrare l'opzione di richiesta di amicizia, si è implementato un metodo che effettui questi due controlli fondamentali per evitare di duplicare un collegamento già esistente. Tale operazione effettua il primo controllo su una richiesta di amicizia che deve essere accettata o meno: nel caso in cui sia il secondo caso allora si dovrà controllare se la richiesta di amicizia è già esistente, procedimento più complesso del primo. Da qui è possibile ottenere l'album di un utente senza effettivamente passare per l'operazione per ottenere un album, cosa utile nel caso in cui i due si conoscano di già e siano amici.

La task che si occupa di tale operazione si chiama ShowAndCheck_task.

1. Inizialmente vi è una chiamata POST di controllo della sessione dell'utente, con ritorno del token di sessione che sarà identico a quello posseduto o nullo se sarà necessario riloggare. (1)
2. A questo punto vi è una richiesta riguardante l'ottenimento dei dati relativi all'utente cercato del quale si deve controllare il legame d'amicizia. La richiesta è effettuata a OSN tramite l'ID dell'utente cercato, e la risposta è il profilo completo di quest'ultimo. (2)

   Se esiste una richiesta di amicizia che non è stata ancora confermata o rifiutata, allora la procedura terminerà con la richiesta di disattivare la possibilità di diventare amico di tale utente. In alternativa:
3. Viene richiesta la chiave pubblica di PFS per effettuare ulteriori controlli ad OSN, che la possiede in locale. La risposta conterrà la coppia pubblica. (3)
4. Viene creato un messaggio contenente l'ID dell'utente cercato e di quello dell'utente corrente, che verrà criptato tramite RSA e la chiave pubblica di PFS. Questo messaggio, poi, verrà inviato a PFS stesso dove verrà decriptato e testata la relazione tra i due utenti. In caso di amicizia avverrà la disattivazione della richiesta, in quanto un collegamento esiste già, in caso contrario potrà essere abilitata la possibilità di richiedere la creazione di un'amicizia nuova. Tale risultato è il messaggio di ritorno dalla comunicazione POST con PFS. (4)

1. $APP \rightarrow OSN : [IDuser]$
   $OSN \rightarrow APP : [session]$

2. $APP \rightarrow OSN : [IDuser, Iduser(requested)]$
   $OSN \rightarrow APP : [IDuser(requested), name, surname, email, city, birthday, photo]$

3. $APP \rightarrow OSN : ['PFS']$
   $OSN \rightarrow APP : [K^{+}_{PFS}]$

4. $APP \rightarrow PFS : [IDuser, IDuser(requested)]_{K^{+}_{PFS}}$

5. $PFS \rightarrow APP : [0/1]$

6.4.9 Ottenere l'album di un utente

Ottenere l'album di un utente è un'operazione semplicissima: l'unico requisito è quello di conoscere l'identificativo dell'utente di cui si vuole ottenere l'album, e verrà ritornato un messaggio contenente l'album voluto. Per poterle vedere però sarà necessario effettuare un'altra operazione, specifica per

la visualizzazione delle immagini [vedi modulo successivo].

La task che si occupa di tale operazione si chiama GetAlbum_task.

1. Viene effettuata una richiesta a OSN di ottenere l'album dell'utente con ID specificato. Il risultato è un array di JSON contenente le varie immagini che compongono l'album dell'utente, criptate.

1. *APP → OSN : [IDuser]*
OSN → *APP : [{metaTag; user{name; surname; email; id; birthday; city; photo}; fileName; ID}]*

6.4.10 Mostrare l'album di un utente

Per poter mostrare un album di un utente si intende la capacità di ottenere una risorsa memorizzata nel database da RMS e decriptarla utilizzando la chiave generata nel caricamento dell'album da KMS. Questa operazione è attualmente effettuabile solamente dal possessore della risorsa, in quanto conosce la passphrase necessaria per descriptarla.

La task che si occupa di tale operazione si chiama ShowAlbum_task.

1. Inizialmente vi è un controllo di sessione dell'utente presso OSN, per confermare il fatto che sia loggato nella rete sociale. Nel caso in cui lo sia viene ricevuto in risposta l'identificativo della sessione corrente. (1)
2. Successivamente viene effettuata una richiesta riguardante l'ID corrispondente al nome della risorsa che voglio controllare, in quanto solo OSN possiede l'identificativo mentre dal lato dell'applicazione viene mostrato il nome. (2)
3. Successivamente viene effettuata una richiesta a OSN della risorsa di cui voglio ottenere la chiave pubblica, nel nostro caso RMS, che viene restituita nella risposta POST. (3)
4. Viene effettuata una richiesta delle chiavi private relative all'utente per effettuare la decriptazione della risorsa, specificando l'ID dell'utente ricercato. Tale risorsa è criptata con la passphrase in AES. (4)
5. A questo punto si effettua una richiesta a RMS per ottenere la risorsa inserendo l'ID della stessa, l'ID dell'utente e un nonce creato per l'operazione, il tutto criptato tramite RSA con chiave pubblica di RMS. La risposta conterrà sia l'URL della risorsa criptato che i parametri criptati per effettuare la decodifica tramite AES, decifrabili solo tramite l'utilizzo della chiave privata dell'utente. Una volta effettuata la decifratura è possibile ottenere un link alla risorsa desiderata. (5)
6. A questo punto si può effettuare una semplice richiesta GET alla risorsa per ottenere l'album criptato tramite secretOwner, token e secret (combinazione di secretOwner e token) in AES. (6)

1. *APP → OSN : [IDuser]*
OSN → APP : [session]

2. $APP \rightarrow OSN : [IDuser(requested), fileName]$
$OSN \rightarrow APP : [IDresource]$

3. $APP \rightarrow OSN : ['RMS']$
$OSN \rightarrow APP : [K^{+}_{RMS}]$

4. $APP \rightarrow OSN : [IDuser]$
$OSN \rightarrow APP : [K^{-}_{APP}]_{Passphrase}$

5. $APP \rightarrow RMS : [\ IDrequestor;\ IDResource;\ n1]_{K^{+}_{RMS}}$
$RMS \rightarrow APP : [\ SecretOwner;\ [\ token;\ [url\_photo]_{K^{+}_{S}}]]_{K^{+}_{APP}}$

6. $APP \leftarrow RMS : [photo]_{K^{+}_{S}}$

```
/**
 *  Asynchronous Task to pick the resource and decrypt it from the system, but you have to know
who owns it, it's name and the passphrase used to encode
 */
public class ShowAlbum_task extends AsyncTask<HashMap<String, Object>, Void, HashMap<String,
Object>> {

    ...
    /**
     * Asynchronous operation that calls the handler to create a pending friendship
     * @param hashmap Hashmap<String, Object> where you can get the params from. It's always
one, so it can be found at index 0.
     *                Params contained:
     *                id: "id_user_album" -> object: an Integer describing the user ID that i
want to get the Album of
     *                id: "passphrase" -> object: an String containing the input passphrase
     *                id: "filename" -> object: an String that is the name of the file
     * @return        Hashmap<String, Object> result of query.
     *                Params contained:
     *                id: "photo" -> object: byte array representing the photo
     */
    @Override
    protected HashMap<String, Object> doInBackground(HashMap<String, Object>... hashmap) {

        HashMap<String, Object> data = new HashMap<>();

        User user = SessionUser.getUser();
        Integer id_user_album = (Integer)hashmap[0].get("id_user_album");
        String passphrase  = (String)hashmap[0].get("passphrase");
        String file_name = (String)hashmap[0].get("filename");

        //Check the session
        JSONObject json = null;
        CheckSession CS = new CheckSession();
        String session = "";
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(CS.getURL(),
user.getId_user().toString()));
            session = (String)(json.get("session"));
        } catch (JSONException e) {
            e.printStackTrace();
        }
```

```
        if(session.length()==0){
            try {
                throw new EmptySessionException();
            } catch (EmptySessionException e) {
                e.printStackTrace();
            }
            return null;
        }

        //Get the Photo from the user with that name, by sending a request in JSON format like
this:
        //var album={"tag":tag,"id":Lockr.get("id"),"fileName":Lockr.get("fileName")};
        GetUserViewPhoto SA = new GetUserViewPhoto(id_user_album, file_name);
        Integer idPhoto = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(SA.getURL(),
SA.getJSON()));
            idPhoto = (Integer)(json.get("idPhoto"));
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Get RMS Keys
        GetPK GPKrms = new GetPK();
        String service = null;
        String modulus = null;
        String exponent = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(GPKrms.getURL(),
ServiceName.RMS.toString()));
            service = (String) json.get("service");
            modulus = (String) json.get("modulus");
            exponent = (String) json.get("exponent");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        PublicKey RMS_pk = new PublicKey(service, modulus, exponent);

        //Make a request to RMS by sending what resource you want to get, who wants it and
generating a random number for the request, JSON format:
        //var msgRMS={"idRequestor":Lockr.get('id'),'idResource':idPhoto,'N1':n1};
        JSONObject jsonmsgRMS = new JSONObject();
        try {
            jsonmsgRMS.put("idRequestor", user.getId_user());
            jsonmsgRMS.put("idResource", idPhoto);
            jsonmsgRMS.put("N1", Math.floor(Math.random()*100+1));
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Encrypt in RSA with RMS keys
        String encryptedMsgRMS = RSA_Util.encryptPublic(RMS_pk.getModulus(),
RMS_pk.getExponent(),jsonmsgRMS.toString());

        //Start the download of the resource: you obtain AES parameters for decryption and the
message encrypted
        GetDownload GD = new GetDownload();
        String AESParams = null;
        String encrypted_msg_client = null;
        try {
```

```
            json = new JSONObject(new HttpPostHandler().makeServiceCall(GD.getURL(),
encryptedMsgRMS));
            AESParams = (String)(json.get("AESParams"));
            encrypted_msg_client = (String)(json.get("encrypted_msg_client"));
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Generate the IV, Salt and variables to store the result of the service call to getting
back the private key exponent
        GetPKClient GPKC = new GetPKClient();
        String clientSalt = null;
        String clientIV = null;
        String clientEncryptedPK = null;
        PrivateKey clientPK = new PrivateKey();
        String modulus_public = null;
        String exponent_public = null;
        String private_key = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(GPKC.getURL(),
id_user_album.toString()));
            //Response is {"salt":salt, "iv":iv, "private_key":private_key}
            clientSalt = (String)json.get("salt");
            clientIV = (String)json.get("iv");
            clientEncryptedPK  = (String)json.get("private_key");
            AesUtil AES_client = new AesUtil();
            private_key = AES_client.decrypt(clientSalt, clientIV, passphrase,
clientEncryptedPK);
            clientPK.setModulus((String)json.get("modulus_public"));
            clientPK.setPublicExponent((String)json.get("exponent_public"));
            //Private or Public exponent? Not clear on the code! Needs to be checked.
            //Should be exponent public.
            clientPK.setPrivateExponent(private_key);

            //Javascript reference code
            //var
keyPrivate=aesUtil.decrypt(data.salt,data.iv,Lockr.get('passPhrase'),data.private_key);
            //rsa.setPrivate(data.modulus_public,data.exponent_public, keyPrivate);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Here AES parameters are getting decrypted using RSA and our modulus and private
exponent
        String AESsalt = null;
        String AESIV = null;
        String AESpassphrase = null;
        try {
            json = new JSONObject(RSA_Util.decrypt(clientPK.getModulus(),
clientPK.getPrivateExponent(), AESParams));
            AESsalt = (String) json.get("salt");
            AESIV = (String) json.get("iv");
            AESpassphrase = (String) json.get("passphrase");
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Here i decrypt the message to the client using the new parameters of AES that i got
just before
        AesUtil AES = new AesUtil();
        String secret_owner = null;
```

```
        String msgFromKMS = null;
        try {
            json = new JSONObject(AES.decrypt(AESsalt, AESIV, AESpassphrase,
encrypted_msg_client));
            //Reference javascript for the JSON result:
            //var secretOwner=decryptedMessageFromRMS.secret_owner;
            //var msgFromKMS=decryptedMessageFromRMS.msgFromKMS;
            secret_owner = (String)json.get("secret_owner");
            msgFromKMS = (String)json.get("msgFromKMS");
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //In the end the message from KMS that we just decrypted gets us the token and the url
of the encrypted resource
        String token = null;
        String url_encryptedrsc = null;
        String secret = null;
        try {
            //json = new JSONObject(RSA_Util.decrypt(clientPK.getModulus(),
clientPK.getPrivateExponent(), msgFromKMS));
            json = new JSONObject(msgFromKMS);
            token = (String)json.get("token");
            url_encryptedrsc = (String)json.get("url_encryptedrsc");
            secret = secret_owner.concat(token);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Here we ask for the photo at the URL
        String encryptedPhoto = new HttpHandler().makeServiceCall(url_encryptedrsc);
        AesUtil AESPhoto = new AesUtil();

        //The photo is encrypted, i decrypt it using the secret_owner (salt), token (IV) and
secret (passphrase)
        byte[] photo = Base64.decode(AESPhoto.decrypt(secret_owner, token, secret,
encryptedPhoto).getBytes(), Base64.DEFAULT | Base64.NO_WRAP);

        data.put("photo", photo);

        return data;
    }

    ...
}
```

6.4.11 Caricare un nuovo album

Per poter caricare una nuova risorsa nella rete sociale è necessario effettuare una serie di passaggi di criptazione a KMS passando sempre per RMS. Queste operazioni diventano ben presto complesse e l'innesto delle richieste complica in modo graduale la richiesta finale che verrà inviata, poiché ogni richiesta RSA che viene effettuata contiene anche la generazione della chiave simmetrica che verrà passata e del messaggio codificato con la chiave simmetrica che viene inviato sia a KMS che a RMS. E' necessario inoltre ottenere la chiave privata dell'utente da KMS per effettuare la criptazione dell'immagine, e questi passaggi devono arrivare sino a KMS attraverso RMS.

La task che si occupa di tale operazione si chiama UploadPhoto_task.

1. Inizialmente vi è un controllo per confermare che l'utente sia in una sessione: se lo è allora verrà ricevuto un messaggio contenente l'eco di tale numero di sessione. (1)
2. Viene effettuata una prima richiesta a OSN per l'inserimento di una nuova risorsa nel sistema, fornendo l'ID dell'utente, il metatag associato all'immagine e al nome del file corrispondente all'immagine. Viene ricevuto in risposta l'id assegnato alla risorsa che la identificherà. (2)
3. A questo punto vengono effettuate due chiamate a OSN per ottenere le chiavi pubbliche dei servizi RMS e KMS, per effettuare in futuro una criptazione RSA. (3 – 4)
4. Da questo punto inizia la vera criptazione della risorsa: viene inviato un messaggio a RMS contenente i dati per la richiesta della propria chiave privata al sistema, passando oltre ai parametri della criptazione AES che verrà utilizzata in futuro per RMS, anche un altro sottomessaggio contenente la risorsa, l'ID e il token di sessione, il tutto incapsulato in un messaggio per KMS contenente l'ID della risorsa, quello dell'utente e un altro token di sessione. In entrambi i messaggi vengono inoltre generati due identificativi casuali unici per una comunicazione temporanea. Questo sistema è molto pericoloso in quanto, se due richieste che vengono effettuate nello stesso istante possiedono lo stesso nonce, si può venire a creare un conflitto temporaneo sulla risorsa da ottenere. A questo punto, dopo le operazioni che vengono svolte in background tra RMS e KMS, si riceve in risposta la chiave privata dell'utente per la decodifica della risorsa. (5)
5. E' necessario infine inviare la risorsa criptata: effettuando un ulteriore messaggio che segue la stessa logica del precedente (messaggio per KMS incapsuato in quello per RMS comprensivo di parametri AES e nonce) si cripterà la foto utilizzando la propria chiave privata, incapsulando tale risorsa nel messaggio per KMS, a sua volta incapsulato in quello per RMS, che otterrà la risorsa criptata e potrà salvarla nel database. (6)

1. $APP \rightarrow OSN : [\ IDuser]$
   $OSN \rightarrow APP : [\ session]$

2. $APP \rightarrow OSN : [\ IDuser;\ tag;\ fileName]$
   $OSN \rightarrow APP : [idResource]$

3. $APP \rightarrow OSN : ['RMS']$
   $OSN \rightarrow APP : [K^{+}_{RMS}]$

4. $APP \rightarrow OSN : ['KMS']$
   $OSN \rightarrow APP : [K^{+}_{KMS}]$

5. $APP \rightarrow RMS : [[salt;\ iv;\ passPrhase]_{K^{+}_{RMS}};\ [\ IDuser;\ n_1;\ idResource,\ [IDuser;\ idResource;\ n_2;\ sessionToken]_{K^{+}_{KMS}};\ sessionToken]]_{K^{+}_{S}}$
   $RMS \rightarrow APP : [\ secretUser;\ N_{11};\ [\ secretResource]_{K^{+}_{APP}}]_{K^{+}_{APP}}$

6. $APP \rightarrow RMS : [[salt;\ iv;\ passPrhase]_{K^{+}_{RMS}};\ [\ IDuser;\ n_{12};\ idResource;\ rule,\ [[iv,\ salt,\ passPhrase]_{K^{+}_{KMS}};\ [\ IDuser;\ idResource;\ n_{22};\ [\ photo]_{K^{+}_{S}}]_{K^{+}_{S}}]]$

```
/**
```

```
 *  Asynchronous Task to upload the photo
 */
public class UploadPhoto_task extends AsyncTask<HashMap<String, Object>, Void, HashMap<String,
Object>> {

    ...

    /**
     * Asynchronous operation that calls the handler to create a pending friendship
     * @param hashmap Hashmap<String, Object> where you can get the params from. It's always
one, so it can be found at index 0.
     *                Params contained:
     *                id: "rule" -> object: an Integer describing how far i want this resource
be seen
     *                id: "passphrase" -> object: an String containing the input passphrase
     *                id: "image" -> object: a byte array that is the image itself
     *                id: "image_name" -> object: a String that is the name of the file, with
format
     *                id: "tag" -> object: a String of the tag associated to the file
     * @return        Hashmap<String, Object> result of query.
     *                Params contained:
     *                id: "photo" -> object: byte array representing the photo
     */
    @Override
    protected HashMap<String, Object> doInBackground(HashMap<String, Object>... hashmap) {

        HashMap<String, Object> data = new HashMap<>();

        User user = SessionUser.getUser();
        String passphrase = (String)hashmap[0].get("passphrase");
        Integer rule = (Integer)hashmap[0].get("rule");
        byte[] image = (byte[])hashmap[0].get("image");
        String tag = (String)hashmap[0].get("tag");
        String image_name = (String)hashmap[0].get("image_name");

        //Check for the session
        JSONObject json = null;
        CheckSession CS = new CheckSession();
        String session = "";
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(CS.getURL(),
user.getId_user().toString()));
            session = (String)(json.get("session"));
        } catch (JSONException e) {
            e.printStackTrace();
        }

        if(session.length()==0){
            try {
                throw new EmptySessionException();
            } catch (EmptySessionException e) {
                e.printStackTrace();
            }
            return null;
        }

        //I post the request to save the file, i get back an ID for future operations
        SaveAlbum SA = new SaveAlbum(tag, user.getId_user(), image_name);
        Integer idResource = -1;
        try {
            //TODO: Check if it's an integer (single data) or JSON
```

```
            json = new JSONObject(new HttpPostHandler().makeServiceCall(SA.getURL(),
SA.getJSON()));
            if((Boolean)json.get("successful")) {
                idResource = Integer.parseInt((String) json.get("idAlbum"));
            }else{
                //exception
                return null;
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Get the Public keys of KMS
        GetPK GPKkms = new GetPK();
        String service = null;
        String modulus = null;
        String exponent = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(GPKkms.getURL(),
ServiceName.KMS.toString()));
            service = (String) json.get("service");
            modulus = (String) json.get("modulus");
            exponent = (String) json.get("exponent");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        PublicKey KMS_pk = new PublicKey(service, modulus, exponent);

        //Get the Public keys of RMS
        GetPK GPKrms = new GetPK();
        service = null;
        modulus = null;
        exponent = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(GPKrms.getURL(),
ServiceName.RMS.toString()));
            service = (String) json.get("service");
            modulus = (String) json.get("modulus");
            exponent = (String) json.get("exponent");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        PublicKey RMS_pk = new PublicKey(service, modulus, exponent);

        //Generate a new JSON for the request to chain together the user and the resource,
adding a random identifier and a session token
        //var
msgKms={"idu":Lockr.get("id"),"idR":Lockr.get("idResource"),"n2":n2,'session_token':self.user.ses
sion};
        JSONObject JSONmsgToKMS = new JSONObject();
        try {
            JSONmsgToKMS.put("idu", user.getId_user());
            JSONmsgToKMS.put("idR", idResource);
            JSONmsgToKMS.put("n2", Math.floor(Math.random()*100+1));
            JSONmsgToKMS.put("session_token", SessionUser.getSessionId());
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //encrypting the generated JSON via RSA using KMS keys
```

```
        String encryptedMsgToKMS = RSA_Util.encryptPublic(KMS_pk.getModulus(),
KMS_pk.getExponent(), JSONmsgToKMS.toString());

        //Encapsulating the message into a message for RMS with the same data
        //var
msgRMS={"idu":Lockr.get("id"),"n1":n1,"idR":Lockr.get("idResource"),"msgKMS":msgKMSEncrypted,'ses
sion_token':self.user.session};
        JSONObject JSONmsgToRMS = new JSONObject();
        try {
            JSONmsgToRMS.put("idu", user.getId_user());
            JSONmsgToRMS.put("n1", Math.floor(Math.random()*100+1));
            JSONmsgToRMS.put("idR", idResource);
            JSONmsgToRMS.put("msgKMS", encryptedMsgToKMS);
            JSONmsgToRMS.put("session_token", SessionUser.getSessionId());
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //This time the encryption is done via AES by generating salt, IV and using the original
passphrase
        String salt = AesUtil.random();
        String IV = AesUtil.random();
        AesUtil AES = new AesUtil();
        String encrypted_JSONmsgToRMS = AES.encrypt(salt, IV, passphrase,
JSONmsgToRMS.toString());

        //Generating the params that only RMS will read
        //var paramRMS={"salt":salt,"iv":iv,"passPhrase":passPhrase};
        JSONObject paramsForRMS = new JSONObject();
        try {
            paramsForRMS.put("salt", salt);
            paramsForRMS.put("iv", IV);
            paramsForRMS.put("passPhrase", passphrase);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Encrypting the params using RSA (secure) ans RMS keys
        String encryptedParamsRMS = RSA_Util.encryptPublic(RMS_pk.getModulus(),
RMS_pk.getExponent(), paramsForRMS.toString());


        //First upload of the request done to RMS
        //JSONmsgToRMS is not encoded
        UploadReq1 UR1 = new UploadReq1(encryptedParamsRMS, encrypted_JSONmsgToRMS);

        String key = null;
        String encrypted_msg_client = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(UR1.getURL(),
UR1.getJSON()));
            key = (String)json.get("AESParams");
            encrypted_msg_client = (String)json.get("encrypted_msg_client");
            //Answer will follow (most likely) the sequent format:
            //var key=data.AESParams;
            //var encrypted_msg_client=data.encrypted_msg_client;
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Here I get back my private key to decrypt the messages from other sources
```

```
        GetPKClient GPKC = new GetPKClient();
        String clientSalt = null;
        String clientIV = null;
        String clientEncryptedPK = null;
        PrivateKey clientPK = new PrivateKey();
        String modulus_public = null;
        String exponent_public = null;
        String private_key = null;
        try {
            json = new JSONObject(new HttpPostHandler().makeServiceCall(GPKC.getURL(),
user.getId_user().toString()));
            clientSalt = (String)json.get("salt");
            clientIV = (String)json.get("iv");
            clientSalt = (String)json.get("salt");
            modulus_public = (String)json.get("modulus_public");
            exponent_public = (String)json.get("exponent_public");
            clientEncryptedPK  = (String)json.get("private_key");
            //Toget the private key i have to decrypt using salt and IV received from the
request, plus the passphrase that only i know
            AesUtil AES_client = new AesUtil();
            private_key = AES_client.decrypt(clientSalt, clientIV, passphrase,
clientEncryptedPK);
            clientPK.setPrivateExponent(private_key);
            clientPK.setModulus(modulus_public);
            clientPK.setPublicExponent(exponent_public);
            //Example of the passage but in javascript
            //var
keyPrivate=aesUtil.decrypt(data.salt,data.iv,Lockr.get('passPhrase'),data.private_key);
            //rsa.setPrivate(data.modulus_public,data.exponent_public, keyPrivate);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //To obtain now the new key that i got before but that is crypted, i need to use the
private exponent that i just obtained
        //key=rsa.decrypt(key);
        JSONObject decryptedKey;
        String newIV = null;
        String newSalt = null;
        String newPassphrase = null;
        /*var iv=key.iv;
        var salt=key.salt;
        var passphrase=key.passphrase;*/
        try {
            //decryption using RSA and Client's private key
            decryptedKey = new JSONObject(RSA_Util.decrypt(clientPK.getModulus(),
clientPK.getPrivateExponent(), key));
            //TODO: Controllare che il metodo e' giusto
            newIV = (String)decryptedKey.get("iv");
            newSalt = (String)decryptedKey.get("salt");
            newPassphrase = (String)decryptedKey.get("passphrase");
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //With the new obtained AES data i can now decrypt the data from RMS
        String secret_user = null;
        Integer nonce_plus_one = null;
        String kms_msg = null;
        /*var secret_user=decrypted_msg.secretUser;
        var nonce_plus_one=decrypted_msg.nonce_one_plus_one;
```

```
        var kms_msg=decrypted_msg.KMSmsg;*/
        AesUtil AES2 = new AesUtil();
        try {
            JSONObject jsonMsgClient = new JSONObject(AES2.decrypt(newSalt, newIV,
newPassphrase, encrypted_msg_client));
            secret_user = (String)jsonMsgClient.get("secretUser");
            nonce_plus_one = (Integer)jsonMsgClient.get("nonce_one_plus_one");
            kms_msg = (String)jsonMsgClient.get("KMSmsg");
            kms_msg = kms_msg.replaceAll("(\\r|\\n)", ""); //At the end of KMSmsg there is a /r
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //With the new obtained AES data i can now decrypt the data from KMS
        JSONObject decryptedKMSmsg;
        String passsecret = null;
        String secret_resource = null;
        Integer nonce_two_plus_one = null;
        try {
            decryptedKMSmsg = new JSONObject(RSA_Util.decrypt(clientPK.getModulus(),
clientPK.getPrivateExponent(), kms_msg));
            secret_resource = (String)decryptedKMSmsg.get("secretRsc");
            nonce_two_plus_one = (Integer)decryptedKMSmsg.get("nonce_two_plus_one");
            passsecret = secret_user.concat(secret_resource);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Encryption for the photo
        AesUtil AES_encryptphoto = new AesUtil();
        String encryptedPhoto = AES_encryptphoto.encrypt(secret_user, secret_resource,
passsecret, new String(ImageConverter.bytesToBase64(image)));

        //Msg to KMS
        //var msgKMS={"id":Lockr.get("id"),"idResource":Lockr.get("idResource"),"n2_2":
(kms_msg.nonce_two_plus_one+1),"encryptedPhoto":encryptedPhoto};
        JSONObject jsonphoto = new JSONObject();
        try {
            jsonphoto.put("id", user.getId_user());
            jsonphoto.put("idResource", idResource);
            jsonphoto.put("n2_2", (nonce_two_plus_one+1));
            jsonphoto.put("encryptedPhoto", encryptedPhoto);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //JSON of the Photo and Resource plus nonce for KMS
        AesUtil AES_msgKMS = new AesUtil();
        String salt_msgKMS = AesUtil.random();
        String IV_msgKMS = AesUtil.random();
        String encryptedPhotoWithResource = AES_msgKMS.encrypt(salt_msgKMS, IV_msgKMS,
passphrase, jsonphoto.toString());

        //Needs to follow the format:
        //var keyKMS={"iv":iv,"salt":salt,"passPhrase":Lockr.get("passPhrase")};
        JSONObject jsonkeyforEPWR = new JSONObject();
        try {
            jsonkeyforEPWR.put("iv", IV_msgKMS);
            jsonkeyforEPWR.put("salt", salt_msgKMS);
            jsonkeyforEPWR.put("passPhrase", passphrase);
        } catch (JSONException e) {
```

```
            e.printStackTrace();
        }
        //Encryption of the Params for KMS
        String encryptedkeyforEPWR = RSA_Util.encryptPublic(KMS_pk.getModulus(),
KMS_pk.getExponent(), jsonkeyforEPWR.toString());

        //Adding both params and photoresource to a JSON
        JSONObject jsonkeysandphoto = new JSONObject();
        //var messageToKMS={"keyKMSencrypt":keyKMSencrypt,"msgKMSEncrypt":msgKMSEncrypt};
        try {
            jsonkeysandphoto.put("keyKMSencrypt", encryptedkeyforEPWR);
            jsonkeysandphoto.put("msgKMSEncrypt", encryptedPhotoWithResource);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Encapsulation of said message for RMS
        //var msgRMS={"id":Lockr.get("id"),"N1_2":
(nonce_plus_one+1),"idResource":Lockr.get("idResource"),"rule":Lockr.get("rule"),"messageToKMS":J
SON.stringify(messageToKMS)};
        JSONObject jsonUpperlayermsgtoKMS = new JSONObject();
        try {
            jsonUpperlayermsgtoKMS.put("id", user.getId_user());
            jsonUpperlayermsgtoKMS.put("N1_2", (nonce_plus_one+1));
            jsonUpperlayermsgtoKMS.put("idResource", idResource);
            jsonUpperlayermsgtoKMS.put("rule", rule);
            jsonUpperlayermsgtoKMS.put("messageToKMS", jsonkeysandphoto.toString());
        } catch (JSONException e) {
            e.printStackTrace();
        }

        //Encryption of the message for RMS
        AesUtil AES_msgKMSUpperLayer = new AesUtil();
        String upperLayerSalt = AesUtil.random();
        String upperLayerIV = AesUtil.random();
        String encryptedMsgKMS = AES_msgKMSUpperLayer.encrypt(upperLayerSalt, upperLayerIV,
passphrase, jsonUpperlayermsgtoKMS.toString());

        //JSON containing the params for RMS
        //var keyRMS={"iv":iv,"salt":salt,"passPhrase":Lockr.get("passPhrase")};
        JSONObject jsonKeyRMS = new JSONObject();
        try {
            jsonKeyRMS.put("iv", upperLayerIV);
            jsonKeyRMS.put("salt", upperLayerSalt);
            jsonKeyRMS.put("passPhrase", passphrase);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        //Encryption of the params
        String encryptedKeyRMS = RSA_Util.encryptPublic(RMS_pk.getModulus(),
RMS_pk.getExponent(), jsonKeyRMS.toString());

        //Send of the message to RMS
        UploadReq2 UR2 = new UploadReq2(encryptedKeyRMS, encryptedMsgKMS);
        new HttpPostHandler().makeServiceCall(UR2.getURL(), UR2.getJSON());

        return data;
    }

    ...
}
```

6.4.12 Logout

Il logout è l'operazione inversa per il login, senza controllo di email e password per l'accesso: in poche parole è una singola operazione che consiste nell'eliminare la sessione corrente dal sistema.

La task che si occupa di tale operazione si chiama Logout_task.

1. Vi è un'unica chiamata POST a OSN, che non ha risposta e accetta in ingresso solamente l'ID dell'utente che vuole effettuare il logout.

    1. APP → *OSN : [IDuser]*

6.5 Eccezioni

Le eccezioni implementate nel progetto sono abbastanza auto esplicative: tutte le classi estendono la classe astratta Exception da cui ereditano anche il costruttore tramite la chiamata super(); all'interno di ognuna di esse, in cui viene impostato un messaggio da mostrare nel caso in cui si verifichi l'errore. Il nome della classe identifica univocamente le condizioni dell'errore.

Purtroppo gran parte delle eccezioni non è stata gestita tramite comunicazioni RESTful web service, rendendo impossibile ottenere i codici d'errore relativi al motivo dell'errore in quanto precedentemente implementati dal passaggio dell'errore puro di Java.

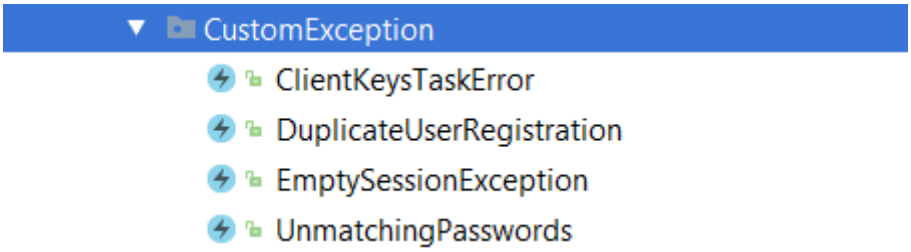

Figura 6.5.1 Package CustomException

6.6 Interfaccia grafica

L'interfaccia grafica di Android è suddivisa in due principali componenti: da un lato troviamo l'effettiva **interfaccia grafica** e le componenti con relativi attributi di posizione, dimensione, etc. che la compongono descritti tramite formato .**xml**, che creano un'attività statica che, da sola, non è in grado di svolgere nessuna funzione. Dall'altro lato si può trovare, nelle **classi Java** corrispondenti alle attività xml, la **gestione** di qualsiasi **funzione** logica e grafica dinamica che è possibile effettuare, con la possibilità di svolgere anche normali operazioni logiche. Naturalmente, in modo più limitato possibile, queste operazioni sono state spostate in altre classi (già descritte precedentemente) in modo da ottenere un codice il più possibile separato tra estetica e logica.

Un classico esempio di **UI** (**user interface)** xml è costituita da un semplice file xml dell'attività

desiderata come, per esempio, il login. All'interno del file è possibile trovare il codice di riferimento a layout e view, ovvero oggetti per la disposizione grafica dei widget (ovvero gli oggetti veri e propri forniti di base da Android, quali bottoni, text input, etc.) contenuti all'interno e gli stessi widget. La disposizione di questi è effettuabile anche da un editor grafico preinstallato su Android, che semplifica le operazioni di implementazione o di modifica della struttura grafica dell'applicazione.

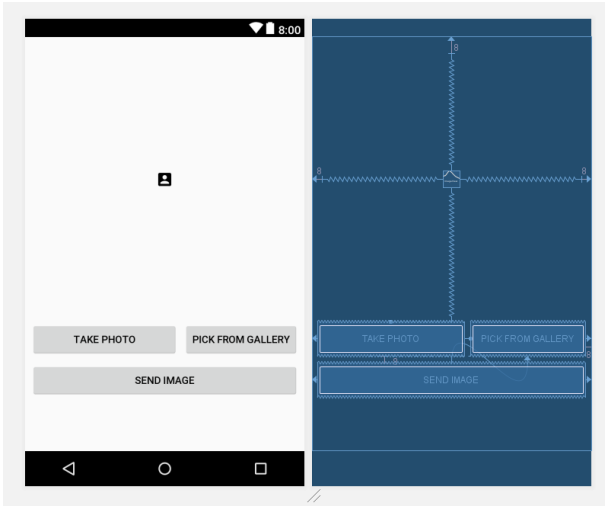

Figura 6.6.1 Interfaccia grafica di creazione della UI di Android

Effettivamente non è sempre questo il caso: infatti alcune delle attività sono costituite da più file, come per le operazioni della rete sociale che possiedono nella normale attività delle richieste di creazione di barre laterali di navigazione oppure di una barra superiore di opzioni (**navigation view**) che fanno riferimento ai file **content** per cambiare di contenuto. In base anche a queste differenze è necessario effettuare dei cambiamenti alla struttura dei file xml.

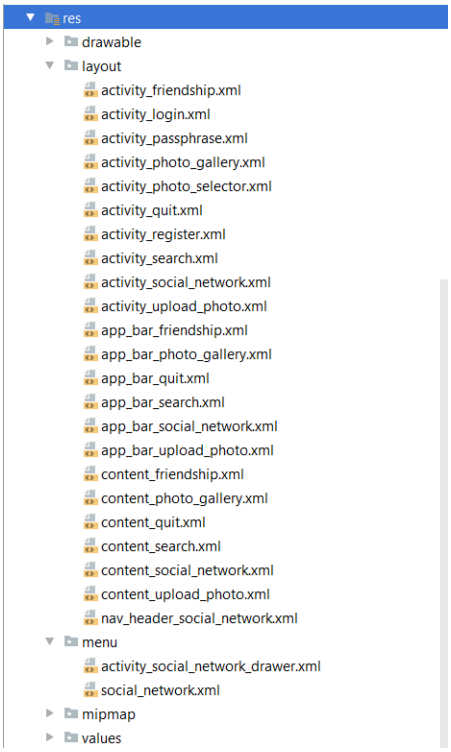

Figura 6.6.2 Folder res

I file Java, in modo opposto a quelli xml, non rappresentano semplicemente il contenuto ma le operazioni sugli stessi: cambiare il testo, visibilità oppure assegnare addirittura una specifica funzione in base ad un evento sono prerogative di queste classi che estendono una delle sottoclassi della superclasse **Activity**.

Oltre a quelle citate, operazioni quali gli intenti saranno considerati parte della gestione della grafica: la creazione di un intento è la volontà di voler passare dalla schermata di attività ad un'altra con la possibilità di includere all'interno di quest'ultimo degli **extra** ovvero delle informazioni supplementari che l'attività successiva necessita di conoscere per il suo corretto funzionamento.

Se inoltre, come detto precedentemente, vi è la necessità di gestire eventi relativi alla barra di navigazione laterale saranno necessari alcuni metodi ereditati dalla classe superiore, in modo tale da poter rimandare alle varie operazioni tramite gli intenti quando richiesto.

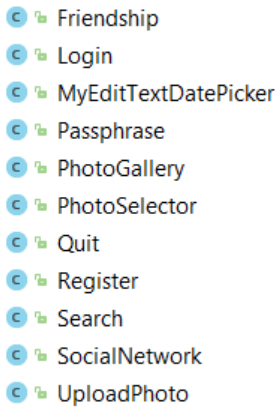

Figura 6.6.3 Classi di modellazione grafica e logica

Poiché tutte le classi eseguono ruoli diversi e operazioni differenti pur essendo molto simili sarebbe inutile riportare ogni singolo caso: l'ideale è dimostrare il funzionamento dell'interfaccia con alcuni esempi di codice ideale di un file d'attività .xml e della classe Java che lo gestisce, insieme a un'immagine che rappresenta i vari passaggi disponibili nell'interfaccia grafica.

```
<SomeSortofLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.socialcloud.Content">

    <!-- Here there are all the graphical objects in XML format -->
    <AnotherView
        android:id="@+id/myid"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
```

```
            <android.support.design.widget.Widget
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
            <Widget
                android:id="@+id/mywidget"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:otherOptions="options"/>
            </android.support.design.widget.Widget>
    </AnotherView>

    <!-- Here other objects can implemented too, like another layout (in another xml
file) -->

</SomeSortofLayout>
```

```
public class JavaActivity extends Activity
        implements Other {

    //variables UI

    /**
     * Method called when you create the activity
     * @param savedInstanceState previous saved state
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //default setup
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity);

        //setup for the navigation bar and the toolbar, if present

        //operations that needs to be done onLoad
    }

    /*
        Other functions that can be starting a new intent, select an object and
changing his format/data, operations for events, management for the drawer, standard
logic operations
    */
}
```

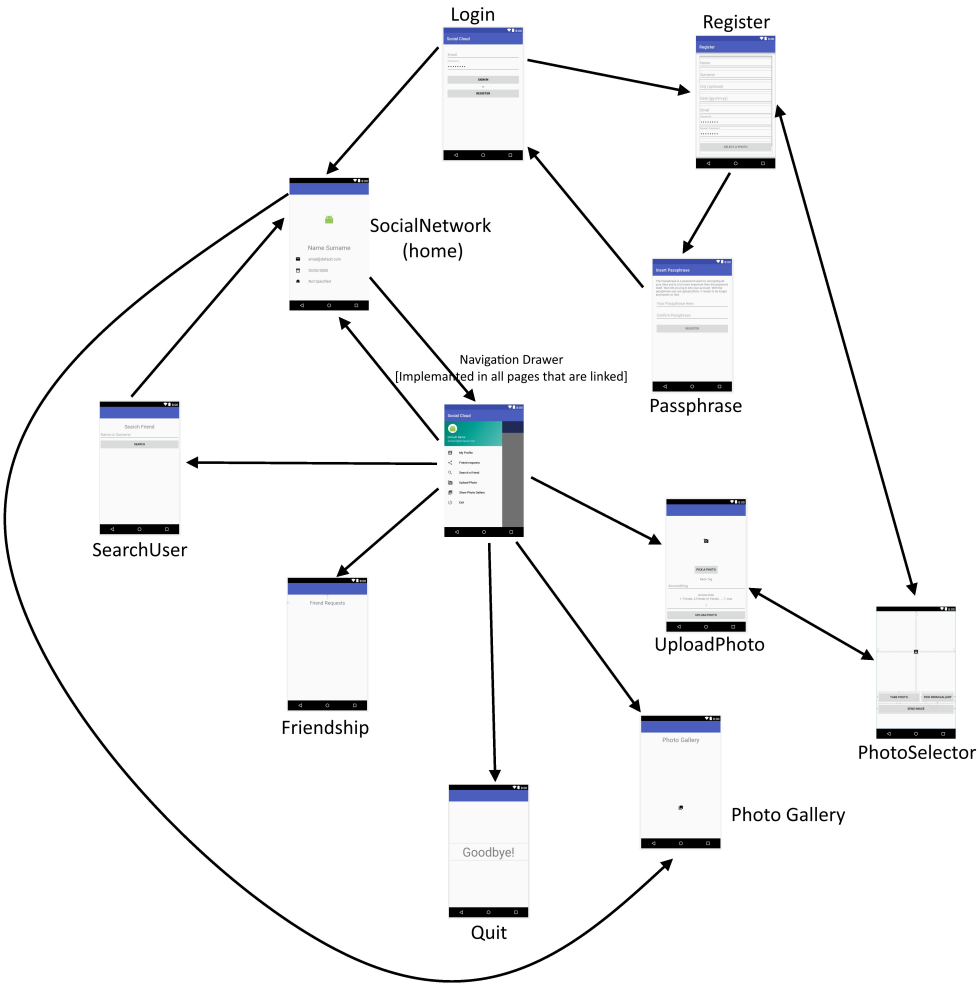

Figura 6.6.4 Activities e rappresentazione grafica degli Intent tra di esse

7

Conclusione

7.1 Modifiche di sviluppo

Seppure lo sviluppo dell'applicazione Android abbia seguito le direttive imposte dal lato server, sarebbe stato comunque possibile procedere in modo diverso e decisamente più affidabile.
Purtroppo, tenendo conto del tempo a disposizione, mi sono è ritrovato ad effettuare una scelta obbligante molto importante e limitativa: modificare solamente uno (o al massimo due) dei moduli del progetto.
Sarebbe stato inoltre il caso di creare un nuovo sistema per interfacciarsi direttamente ai vari moduli e non passare esclusivamente dall'OSN: gran parte delle richieste vengono "copiate" o comunque ricevute da OSN prima di essere spedite al requestor, anche se vengono rese indecifrabili anteriormente all'invio. Questo perché il sistema doveva precedentemente passare da AngularJS, che era comunque un modulo OSN, per essere mostrato. Ora, utilizzando un'applicazione esterna come per Android, ciò non è più necessario: nei restanti moduli non è più richiesto inviare all'indirizzo statico di OSN la risposta quando viene effettuata, ma si può passare direttamente al nuovo "CS" che è separato dal contesto del modulo "madre". Ciò significherebbe modificare le richieste ai moduli KMS, RMS e PFS, operazione che richiede lo studio attento e approfondito di ogni modulo per evitare di creare problemi a catena, senza contare la rimozione di alcune funzionalità da OSN.

Nel caso in cui non vi fosse stata questa limitazione, comunque, l'ideale starebbe stato modificare ogni modulo in modo da applicare la soluzione già precedentemente mostrata anche a livello di criptazione: *perché impiegare risorse e tempo all'utente per effettuare algoritmi dispendiosi quando è possibile effettuare una ben più sicura criptazione e autenticazione da esterno che non implica nessun aumento di costo sul progetto?*

Altro problema che si è venuto a creare ma che, ora, non è stato più risolto è quello della versione desktop della rete sociale. Infatti, come abbiamo sfatato nelle prime battute della tesi, Javascript non si può considerare un linguaggio sicuro. Questo problema, a sua volta, finisce a influire sullo sviluppo effettuato nel progetto iniziale che forniva una versione di accesso da client via proprio web browser usando come criptazione javascript nel lato front end.
Le soluzioni sono nell'effettivo due: o cambiare linguaggio su quel lato o fornire il servizio di criptazione lato server.
Purtroppo... non è possibile cambiare il linguaggio. Javascript è diventato, senza ombra di dubbio, l'unico linguaggio lato client possibile e disponibile in ogni web browser. Esistono solamente dei sottolinguaggi sviluppati in javascript, ed è forse l'unica possibilità disponibile per ottenere ciò che cerchiamo, ma sempre con l'ausilio di TLS/SSL.

Per quanto riguarda la gestione tramite lato server, la soluzione sarebbe decisamente più semplice e diffusa: infatti gran parte dei siti effettua questo tipo di criptazione di dati rilevanti (anche bancari, ad esempio). Unico vero e proprio problema è quello di separare ulteriormente il sistema di gestione di richieste, chiavi, etc. (OSN) da quello di login e richieste: in questo modo non vi potrà essere alcun tipo di tracciatura dei movimenti e dati dell'utente.

7.2 Soluzioni alternative: SSL e TLS

Seppure sia stato effettuato un approccio che risolve i problemi e la criptazione avvenga da parte sia del client che del server attraverso criptazioni a livello pubblico, con l'avanzare del tempo non è garantito che le tecniche, soprattutto i parametri delle dimensioni delle chiavi ad esempio, utilizzate nel progetto siano soggette a futuri sviluppi in campo crittografico e vengano dichiarate insicure o si rivelino soggette ad un nuovo tipo di attacco. Infatti, come spiegato per AES, la possibilità sussiste seppure sia solamente teorica. E' inoltre necessario dire che le comunicazioni, pur essendo buone, richiedono che l'utente svolga molteplici operazioni possibilmente inutili a un grande costo di tempo e risorse, rendendo più pesante l'applicazione specialmente su dispositivi mobili più arretrati dotati di capacità minori.

Dunque quali sono le soluzioni alternative?
Le risposte sono semplici: **SSL** e **TLS**.

SSL, acronimo di **Secure Sockets Layer**, e **TLS**, **Transport Layer Security**, sono dei protocolli in grado di fornire la criptazione e autenticazione operando sopra il **livello di trasporto** della **pila protocollare TCP/IP**. Il funzionamento dei due sistemi varia solamente per il metodo con cui viene iniziata la comunicazione.
Nel caso in cui sia eseguita in modo **esplicito** allora si parlerà di SSL: la richiesta di connessione avviene attraverso una specifica porta impostata dal lato server e viene utilizzata nello scambio di messaggi.
Nel caso in cui la comunicazione avvenga attraverso un **protocollo specifico**, allora si parlerà di TLS: la comunicazione inizia con un handshake tra client e server, per poi passare a una comunicazione sicura su altri canali, e non più fissi.

Lo svolgimento dello scambi di messaggi tramite protocollo TLS è il seguente, e viene condiviso (escludendo l'handshake iniziale, inutile per SSL che era preimpostato su una porta specifica):

- Inizialmente si **negozia** fra le due parti il protocollo di cifratura, gli **algoritmi** da **utilizzare** e l'algoritmo di **autenticazione** insieme al **MAC** (**Message Authentication Code**) e utilizza algoritmi asimmetrici che potrebbero avere **chiave precondivisa**.
- Successivamente si effettua il vero e proprio **scambio di chiavi** e l'**autenticazione** (effettuata tramite **Hash** o **funzioni pseudorandom**).
- **Cifratura simmetrica** e **autenticazione** dei **messaggi**, ottenuta tramite algoritmi simmetrici.

La struttura come possiamo notare è effettivamente simile a quella utilizzata nel progetto principale, con la grande differenza che questi processi non vengano eseguiti solamente tra due enti che devono eseguire tale procedura, infatti entra in atto un'**autorità di certificazione** (**CA**) esterna che **effettua**

le **operazioni** di **validazione** delle comunicazioni e del **certificato** all'admin del **sito web**. Queste verranno confermate semplicemente all'utente nel momento in cui, accedendo al sito web che mostrerà il proprio certificato, si potrà comparare la chiave ricevuta con quelle presenti nella CA, successivamente procedendo la comunicazione su un canale di comunicazione sicuro[16].

Purtroppo, seppure le versioni **SSL 3.0** e **TLS 1.2** erano disponibili all'uso sino a qualche tempo fa, è stato recentemente imposto dalla **IETF** (**Internet Engineering Task Force**) che questi protocolli sono **deprecati** e devono essere obbligatoriamente **sostituiti** con la versione **TLS 1.3**, giudicata l'unica in grado di **garantire** la **sicurezza** del **web**. Non solo dunque è stato sconsigliato l'ulteriore sviluppo in **SSL 3.0** ma è stato bensì definito come **proibito** e dunque da evitare assolutamente[17].

7.3 Studio e aggiornamento

Seppure il codice proposto possa apparire perfettamente funzionante e privo di grandi problemi, la realtà è che il rilascio effettivo di un'applicazione a questo livello si rivela ben lontana dalla realtà attuale. Escludendo la parte grafica del progetto, le prestazioni dell'applicazione devono essere migliorare con successivi studi approfonditi degli algoritmi utilizzati che variano anche in base al tipo di criptazione utilizzata. Spostando infatti la criptazione su TLS/SSL risulterebbe più leggera: la criptazione effettuata dal client sarebbe asimmetrica solamente inizialmente e utilizzerebbe criptazioni simmetriche per tutti i successivi messaggi in canali sicuri. Questa cosa può essere al massimo riprodotta dalle comunicazioni simmetriche già presenti tra i vari moduli, evitando la complessità di quelle asimmetriche in tutti gli altri casi in cui è invece usata, effettuando in grande risparmio di tempo e risorse.

Successivamente all'incremento delle prestazioni e al bugfix degli errori principali, è necessario procedere con lo sviluppo delle funzionalità della rete sociale.
Come è facile prevedere, per competere con i grandi colossi delle reti sociali, è necessario creare un sistema in grado di fornire dei servizi base e superiori in grado di soddisfare l'utenza e ciò che ricerca in una rete sociale: la possibilità non solo di condividere brevi messaggi o addirittura album di foto, ma anche video, la capacità di cercare e trovare contenuti e persone velocemente, senza contare funzionalità aggiuntive quali chat private o condivisione di file ad esempio.
Queste scelte devono essere adoperate in base alla direzione che si vuole dare alla rete sociale, senza finire ad essere un “Jack of all trades”: senza una direzione precisa si è destinati a fallire.

Problema finale sia di sviluppo che di marketing è quello di proporre una grafica accattivante e di facile utilizzo: poter ottenere il maggior bacino di utenti è non solamente necessario ma imperativo per potersi far conoscere e ottenere dunque l'utenza iniziale in grado di potare le persone a sfruttare il servizio creato. E' sulla base della teoria dei giochi e di cluster che si dovrebbe effettuare un approfondimento per l'espansione del servizio: è vero che facebook è il più potente delle social network, ma *è possibile che nessuno ha mai pensato di crearne uno che cerchi di esserne un rivale?*
La risposta è un semplice “Sì”, ma tutti hanno fallito precedentemente, poiché il prezzo da dover pagare per “abbandonare” una piattaforme così utilizzata e conosciuta è troppo alta rispetto allo sviluppo ed espansione di una nuova.
Che cosa si intende con questa frase? La teoria dei giochi è una scienza matematica che studia il rapporto tra probabilità di vincita e perdita di un determinato evento, in cui la scelta di una persona viene influenzata dal guadagno che questo ricava sulla decisione che effettua.

La teoria può essere vista e applicata anche per l'applicazione ed espansione di nuove tecnologie o sistemi, come nel nostro caso, abbinata all'espansione su cluster basati su una soglia, ovvero quante persone sono necessarie per fare in modo che, in una rete di relazioni su una persona decide di utilizzare un nuovo sistema, tutti i componenti finiscano ad adottarla.

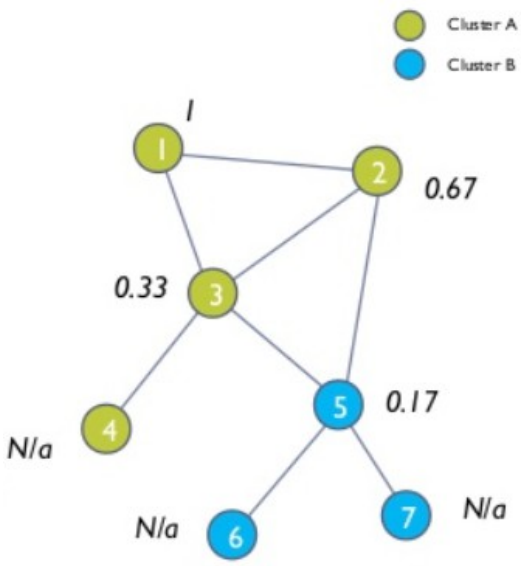

Figura 7.3.1: Rete sociale clusterizzata

Purtroppo non è effettivamente possibile dare un valore effettivo, se non con studi molto più approfonditi su gruppi di persone attraverso questionari, ad esempio, a questi valori al momento, ma si può presupporre che la probabilità di introdurre e sostituire una rete sociale come Facebook o Twitter, al momento, sia quasi impossibile se non per gruppi molto piccoli o scollegati principalmente.

Bisogna dunque puntare sulla capacità di poter “mantenere” entrambe le reti sociali riducendo la quantità di “lavoro” che ogni utente spende su quest'ultima ma facendo affidamento invece ad un'espansione più rapida e localizzata, puntando a influenzare un paese, ad esempio.

Extra

Attacchi crittografici

Atk1: Bruteforce

Un attacco di tipo **bruteforce** non è altro che il meno metodico e possibilmente più semplice degli attacchi possibili nella crittografia: si basa sul provare ogni combinazione possibile di numeri, stringhe o altro necessarie per effettuare la criptazione andando a tentativi sequenziali sino a trovare il risultato ottenuto.

Atk2: Man-in-the-Middle

L'attacco **MITM** consiste in un attaccante che si pone come **terza parte intermedia** di una **conversazione** tra due parti, assicurandosi la capacità di intercettare e, a suo volere, modificare i messaggi provenienti da entrambi, impersonandosi in una delle due persone coinvolte. Nell'effettivo l'attaccante MITM è colui che possiede le sorti della conversazione stessa: tutti i messaggi passano attraverso lui e non vi è modo di intendere se l'attacco è in atto o meno.

Il problema di questi attacchi nasce se l'attaccante riesce a **impossessarsi** di **una delle** due **chiavi pubbliche** (**o** della chiave **simmetrica**) delle parti coinvolte: se il MITM succede in questa operazione, invece che inviare la chiave pubblica dell'utente a cui è stata rubata può inviare la sua, facendo dunque criptare ad una delle due vittime i messaggi con la sua chiave pubblica corrispondete, leggibile da lui soltanto. I successivi messaggi verranno dunque criptati con la chiave del MITM che potrà decifrarli, leggerli, e poi criptarli con la chiave pubblica che originariamente doveva essere inviata all'altro utente e che, invece, è stata presa dall'attaccante.

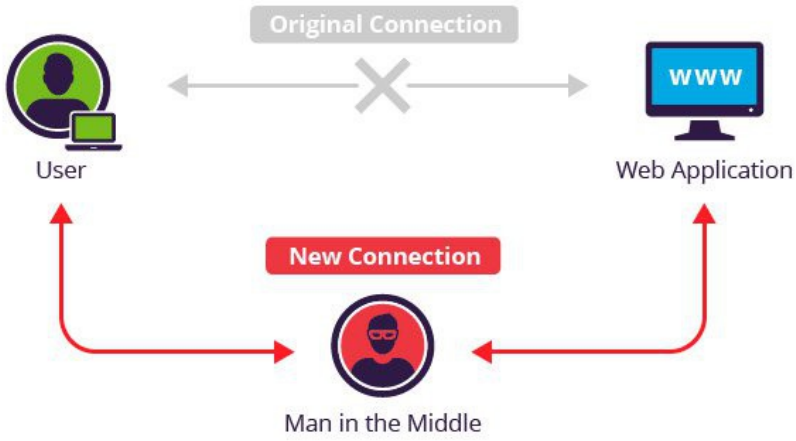

Figura Atk2: Man in the Middle

Atk3: Code Hijacking

Il termine **Hijacking** può essere tradotto in "**dirottare**" o "**prendere possesso**": nel caso di **code hijacking** si intente la capacità attaccate di **ottenere** il **codice** eseguito da parte dell'utente attaccato in modo tale da essere **modificato** per ottenere le informazioni o cambiarne il contenuto.

Atk4: Cross-site scripting

L'attacco **XSS** consiste nella raccolta, manipolazione e modifica dei dati presenti su server oppure delle pagine e di come vengono visualizzate dal client, come ad esempio negli **attacchi DOM-based**.
Questi attacchi sono una scoperta abbastanza recente (risalente al 2011) e consistono in una modifica di come il lato client vede riflesse le informazioni provenienti dal server, sfruttando delle "injection" di codice javascript non desiderate per modificare la dinamicità delle pagine.

Bibliografia

[1] Di **Luca Scarcella** - “Facebook raggiunge 2 miliardi di utenti attivi al mese”, disponibile presso: http://www.lastampa.it/2017/06/28/tecnologia/news/due-miliardi-di-utenti-usano-facebook-Qne6S3f8L1Pu6ebMRsB8RK/pagina.html
[1] Di **Chris Welch** - “Facebook crossed 2 billion monthly users”, disponibile presso: https://www.theverge.com/2017/6/27/15880494/facebook-2-billion-monthly-users-announced

[2] Di **Statista** - “Statistiche su smartphone”, disponibile presso: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[3] Di **Zakas, Nicholas C. -** "The evolution of web development for mobile devices.", disponibile presso: http://www.yildiz.edu.tr/~aktas/courses/CE-0112822/29-04-4-1.pdf

[4] Di W3C - “The Web and Mobile Devices”, disponibile presso: https://www.w3.org/Mobile/

[5] Di **Judy Brewer** e **Shawn Lawton Henry** - “Mobile accessibility at W3C”, disponibile presso: https://www.w3.org/WAI/mobile/
[5] Di **W3C** - “Mobile accessibility current status”, disponibile presso: https://www.w3.org/standards/techs/mobileaccessibility#w3c_all

[6] Di **Art Barstow, Anssi Kostiainen, Jo Rabin, J. Manrique López, Mounir Lamouri, Marcos Caceres, François Daoust** e **Ronan Cremin** - “Standards for Web Application on Mobile: current state and roadmap”, disponibile presso: https://www.w3.org/Mobile/mobile-web-app-state/

[7] Di **Google** - “Siti mobili”, disponibile presso: https://developers.google.com/search/mobile-sites/

[8] Di **Maurizio Casimirri** - “MobileAngularUI”, disponibile presso: http://mobileangularui.com/

[9] Di **Google code** - “CryptoJS”, disponibile presso: https://code.google.com/archive/p/crypto-js/

[10] Di **NCCGroup** - “Java Cryptography Considered Harmful”, disponibile presso: https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/

[11] Di **Stanford University** - “SJCL”, disponibile presso: https://github.com/bitwiseshiftleft/sjcl

[12] Di **StatCounter** - “Share OS mobile”, disponibile presso: http://gs.statcounter.com/os-market-share/mobile/worldwide

[13] Di **Oracle** - “Documentazione javax.Crypto”, disponibile presso: https://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html

[14] Di **Daniel J. Bernstein** - “Cache timing attacks on AES”, disponibile presso: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[15] Di **Wikipedia (en)** - “XSL attack”, disponibile presso: https://en.wikipedia.org/wiki/XSL_attack

[16] Di **Internet Security Research Group (ISRG)** - “Let's Encrypt, How It works”, disponibile presso: https://letsencrypt.org/how-it-works/

[17] Di **IETF** - “The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-23”, disponibile presso: https://tools.ietf.org/html/draft-ietf-tls-tls13-23