Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA
SCUOLA DI SCIENZE
Corso di Laurea in Scienze dell'Informazione

Applicazione Mobile per Supporto alla Didattica

Relazione Finale in Programmazione

Relatore:
Chiar.ma Prof.ssa
Antonella Carbonaro

Presentata da:
Tomas Tassinari

Sessione III
Anno Accademico 2012/2013

The brick walls are there for a reason. Remember brick walls let us show our dedication. They are there to separate us from the people who don't really want to achieve their childhood dreams.

Randy Pausch - The Last Lecture [1]

Un importante ringraziamento va:

agli amici più cari, che mi hanno sostenuto in modo incondizionato, condividendo le gioie e i momenti difficili e restando sempre al mio fianco.

a coloro che durante questo primo percorso universitario hanno avuto occasione di trasmettermi piccoli e grandi insegnamenti di vita e spunti di riflessione personale, dei quali farò tesoro lungo la strada.

a colui che mi ha mostrato il traguardo, facendomi capire quanto sia importante e soddisfacente raggiungerlo.

alla mia famiglia, che mi ha permesso di cadere sul morbido e ha continuato a credere con fiducia che mi sarei rialzato.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Il settore tecnologico del *mobile* ha avuto, negli ultimi anni, un ruolo particolarmente centrale nelle strategie di tutte le grandi aziende del settore.

Tutto deve diventare *portatile* e *"smart"*.

Se fino a qualche anno fa la *regola non scritta* per ogni attività commerciale o fornitrice di servizi era avere un sito web, oggi, questa priorità è diventata il poter fornire al cliente un servizio consultabile da uno smartphone, che sia in forma di sito web ottimizzato per i dispositivi mobili, o piuttosto una vera e propria applicazione scritta in linguaggio nativo del sistema operativo mobile.

Questa tesi basa la propria esistenza e la propria utilità su questa nuova esigenza dell'utente comune: poter accedere a ogni risorsa tramite il proprio smartphone.

Si è quindi sentita la necessità di poter fornire agli studenti del corso di Programmazione di questo corso di laurea una versione compatibile con tutti gli smartphones Android del portale *Programmazione.Info*, attualmente utilizzato sia come repository dei materiali per il supporto alla didattica, sia per la gestione degli esercizi propedeutici all'esame che ogni studente deve svolgere di settimana in settimana durante la frequentazione delle lezioni.

Grazie a questa applicazione lo studente non solo avrà un accesso facilitato ai materiali didattici presenti attualmente sul sito ma potrà anche gestire dal proprio smartphone la consegna degli esercizi e controllarne lo stato di correzione, il tutto in mobilità e senza la necessità di utilizzare un computer, potendo anche sfruttare la comodità di servizi *cloud* esterni per salvare i

propri documenti e consegnarli poi tramite l'applicazione.

Questo progetto, inoltre, non è rivolto solo agli studenti ma anche agli operatori addetti alla correzione degli esercizi sottomessi dagli studenti, si vuole infatti fornire a questi ultimi la possibilità di consultare i materiali in attesa di correzione, accedere ai codici sorgenti caricati dagli studenti e effettuare il processo di convalida e definizione dell'esito della consegna.

Questa tesi illustra il processo di creazione di questa applicazione ed è suddivisa nei seguenti capitoli:

Capitolo 1 - Stato dell'arte delle tecnologie mobili

In questo primo capitolo viene fatto un breve cenno allo stato di diffusione dei dispositivi mobili, per poi illustrare più nel dettaglio i tre principali sistemi operativi che dominano il mercato degli smartphones: Google Android, Apple iOS, Microsoft Windows Phone 8.

Capitolo 2 - Ambiente di sviluppo del progetto

Vengono illustrate le principali tecnologie utilizzate nell'ambito del progetto, sia inerenti alla programmazione dell'applicazione in sé, sia riguardanti la creazione dei moduli esterni necessari al suo funzionamento e alle operazioni di test effettuate durante lo sviluppo.

Capitolo 3 - Fase di analisi del progetto

In questa sezione viene descritta la fase di analisi del problema e delle tecnologie esistenti a cui l'applicazione andrà ad affiancarsi, la definizione degli obiettivi da raggiungere e quali scelte sono state fatte riguardo gli strumenti da utilizzare per l'implementazione delle diverse funzionalità.

Capitolo 4 - Implementazione del progetto

In questo ultimo capitolo viene illustrato il progetto entrando nel dettaglio delle scelte implementative, fornendo una descrizione dei principali utilizzi dell'applicazione e delle sue caratteristiche più importanti.

Capitolo 1

Stato dell'arte delle tecnologie mobile

In questo capitolo si analizza la situazione attuale delle *tecnologie mobile* presenti sul mercato, fornendo una panoramica generale sui principali sistemi operativi per smartphones, sulle caratteristiche principali di ognuno e sulla situazione e lo sviluppo del mercato delle applicazioni.

1.1 Diffusione dei dispositivi mobili

Oggi, nel mondo, vengono usati quotidianamente più di un miliardo di smartphone. Una persona ogni sette possiede e utilizza abitualmente un telefono cellulare di ultima generazione.

Nell'ultimo anno vi è stata una crescita del 47%, passando da circa 700 milioni agli attuali 1,03 miliardi di smartphone in uso sul pianeta. Si stima inoltre che nei prossimi 3 anni vi sarà un ulteriore raddoppio dell'attuale numero, arrivando a quota 2 miliardi di dispositivi in uso per un giro d'affari di circa 219 miliardi di dollari.[2]

Dopo la rivoluzione mondiale portata dai telefoni cellulari, è ora il momento dell'ascesa dei telefoni multimediali e plurifunzionali.

É inoltre importante considerare che la diffusione di questi dispositivi non riguarda solo le zone più industrializzate del pianeta, ma vede tassi di crescita straordinari anche e soprattutto nelle zone in via di sviluppo, prime fra tutte l'Africa, l'India e la Cina. Attualmente, il 30% degli utenti mondiali vive in India e in Cina e questo dato è destinato a aumentare vertiginosamente nei prossimi anni, con un mercato che è solo all'inizio della sua espansione e già da adesso vede la Cina al primo posto con 160 milioni di utilizzatori di internet mobile. [3, 4]

Questa tendenza è evidenziata non solo dal numero di dispositivi venduti ma anche dal reale utilizzo che ne viene fatto: il 13% del traffico internet nel mondo parte da un dispositivo mobile, rispetto al 4% di soli 2 anni fa, un aumento superiore al 200% in 2 anni.[4]

Analizzando il solo dato della visualizzazione di pagine web, si scopre che nel 2012 il 10,1% del totale viene effettuato da un dispositivo mobile, con picchi del 12,9% in Africa e del 18% in Asia, dove la diffusione dei personal computer è ancora inferiore rispetto al resto del mondo. Solo un anno fa, la percentuali erano meno che dimezzate, con un 5,8% mondiale e un 8,3% in Asia (un aumento del 125% in soli 12 mesi)[5, 6].

1.2 Sistemi operativi mobile

L'evoluzione da telefoni cellulari a smartphone ha introdotto quella che attualmente è la maggior discriminante per confrontare due dispositivi di ultima generazione: il sistema operativo.

Nei primi cellulari non era importante "cosa facesse" il telefono, perchè tutti svolgevano le medesime funzioni: invio e ricezione di telefonate e SMS, gestione di numeri in rubrica, orologio, e poc'altro. Ciò che determinava la scelta, a parte ovviamente il prezzo, erano l'affidabilità del marchio, le dimensioni, il peso del dispositivo e pochi altri fattori di minore importanza.

Gli smartphone hanno rivoluzionato il modo in cui il cliente sceglie quale telefono acquistare. Fattori come dimensioni e peso sono finiti in secondo

piano, ora ciò che fa pendere l'ago della bilancia in un senso piuttosto che nell'altro è il numero e la qualità delle funzionalità che il telefono fornisce. E alla base di queste differenze vi è, appunto, il sistema operativo.

Stando ai dati del terzo quadrimestre 2012, Android, il sistema operativo di Google e iOs, di Apple, insieme dominano l'85% del mercato degli smartphones. Nel dettaglio, Android risulta installato nel 68,1% dei dispositivi venduti nel mondo, con una crescita considerevole rispetto al 46,9% dell'anno precedente. iOs segue con il 16,9% del totale, in leggero calo rispetto al 18,8% del 2011.

Il resto del mercato viene diviso da BlackBerry OS, in continua discesa negli ultimi anni, con il 4,4% del totale (solo un anno prima era al 11,5%), Symbian, il sistema operativo supportato da Nokia Corp, anch'esso in caduta libera con un 4,4% e un calo di 12,5 punti percentuali in soli 12 mesi e infine la versione mobile di Windows, nata dopo la rivoluzione che ha visto abbandonare Windows Mobile in favore di Windows Phone 8 e che attualmente occupa il 3,5% del mercato.

1.2.1 Windows Phone 8

Il sistema operativo del colosso di Redmond risulta essere l'unico reale competitor della coppia Android-iOs che attualmente domina il mercato. Questo prodotto è forte della partnership nata fra Microsoft e Nokia dopo l'acquisizione della seconda in favore della prima.

L'azienda ex-leader del mercato dei cellulari ha visto negli ultimi anni un tracollo irreversibile delle vendite e dei profitti, ed è evidente che la causa principale di questo declino sia da imputare alla scelta di investire su Symbian Os, sistema operativo che, è ormai evidente, non ha avuto la capacità di restare al passo con l'evoluzione del settore e competere con avversari che si sono saputi evolvere, al contrario di un sistema che è rimasto bloccato nella prima embrionale fase del passaggio da cellulari a smartphones.

Basti pensare che nel 2007, anno in cui venne presentato il primo iPhone, Nokia copriva il 49,4% del mercato degli smartphones, nei successivi tre anni

la quota è scesa fino al 34,2%, fino ad arrivare al 3% all'inizio del 2013.[7, 8]

Nokia è quindi stata scelta per fornire l'hardware per i prodotti equipaggiati con il nuovo sistema operativo mobile di Microsoft.

Tramite una intensa campagna di acquisizione di sviluppatori per il proprio market, l'azienda di Redmond sta riuscendo ad aumentare la propria offerta di applicazioni, rendendo il proprio sistema operativo più appetibile sia per l'utenza che, oltre alla familiarità con il brand, si trova di fronte a prodotti dalle caratteristiche hardware superiori alla media a prezzi competitivi, sia per i developers che vedono in Windows Phone uno dei possibili futuri leader del mercato nonché l'onda da cavalcare per ottenere concreti guadagni sfruttando lo stato iniziale del progetto.

1.2.2 iOS

iOs, attualmente sul secondo gradito del podio nella classifica di diffusione, è il sistema operativo alla base dell'iPhone, lo smartphone prodotto da Apple Inc. Giunto attualmente alla versione 7, questo sistema operativo fa delle sue caratteristiche principali la facilità d'uso e l'immediatezza, per puntare a una fascia di utilizzatori più ampia possibile senza richiedere alcuna conoscenza o capacità tecnica per un utilizzo soddisfacente del dispositivo.

Interfaccia utente

L'interfaccia utente è rimasta sostanzialmente invariata dalle prime release fino ad oggi, riuscendo a imporre standard e convenzioni nell'uso intuitivo di un dispositivo che non si conosce e su questi punti di forza ha continuato a basare la propria evoluzione. Con la versione 7 del sistema operativo è stato apportato un importante restyling grafico, mantenendo però le principali caratteristiche di usabilità che hanno sempre caratterizzato la sua interfaccia.

Caratteristiche chiave

Omogeneità.

É questa una delle principali e fondamentali caratteristiche dell'ambiente iOs.

Ogni smartphone che utilizza questo sistema presenta la medesima interfaccia utente, lo stesso funzionamento di base, gli stessi riferimenti, le stesse applicazioni.

Ogni iPhone si usa allo stesso modo di tutti gli altri.

Variano le caratteristiche hardware e le funzionalità evolute, a seconda delle novità apportate a partire dai primi modelli (iPhone 2G) fino agli ultimi (iPhone 5s) sono state introdotte funzionalità avanzate per il reparto fotografico, per esempio, come le foto panoramiche o la modalità di scatto HDR, o novità hardware come la bussola per la geolocalizzazione o il giroscopio per poter utilizzare i movimenti del device nello spazio, ma chi ha familiarizzato con il primo iPhone, sarà immediatamente in grado di utilizzare in modo soddisfacente anche l'ultimo modello immesso nel mercato.

Tutto è orientato alla "user experience". L'iPhone, e di conseguenza iOs, ha il prioritario scopo di essere semplice e alla portata di chiunque. Non esistono manuali di istruzioni, la curva di apprendimento è estremamente ripida, lo scopo di questa accoppiata hardware&software è risultare usabile e soddisfacente nell'immediato.

Un esempio che rende evidente qual è lo scopo di Apple può essere fornito con la procedura per il primo utilizzo del device. Inizialmente, un iPhone uscito dalla confezione e acceso presentava un'icona che invitava l'utente a collegare il dispositivo a un qualsiasi PC o Mac tramite il cavo USB fornito in dotazione. Con questa operazione, si costringeva l'utente a utilizzare il software iTunes che si occupava dell'inizializzazione del dispositivo e delle operazioni per renderlo operativo e utilizzabile. Tutto in automatico, l'intervento dell'utente era minimo.

Nonostante il sistema puntasse quindi prima di tutto sulla semplicità, con le ultime versioni si è voluto eliminare anche questo primo scoglio dell'interfacciamento con un pc (e quindi della necessità di possederne uno). Il risultato è un iPhone che appena acquistato e acceso richiede al proprietario poche

semplici informazioni per poi autoconfigurarsi e risultare immediatamente operativo.

Metodi di aggiornamento

Anche le operazioni di aggiornamento sono state ulteriormente semplificate, eliminando di fatto la necessità di utilizzare un qualsiasi altro dispositivo e rendendo ogni iPhone totalmente indipendente. Gli aggiornamenti vengono segnalati all'utente, scaricati tramite internet, e installati in modo totalmente automatizzato e nascosto all'utente, che ha il solo compito di confermare la volontà di eseguire l'update di sistema. Ogni operazione di backup e ripristino dei dati è eseguito in automatico sfruttando una struttura cloud proprietaria, denominata iCloud, fornita di serie e gratuitamente ad ogni possessore di un device Apple.

Non è necessario che l'utente comprenda nessuna delle operazioni che vengono svolte in questi frangenti, tutto ciò che gli è richiesto è "Vuoi aggiornare? Queste sono le novità che otterrai al termine delle operazioni", tutto il resto è gestito in autonomia dal sistema.

Sistema chiuso

La caratteristica che viene spesso additata come "peggior difetto" del sistema operativo made-in-Cupertino è la "chiusura" a interventi diretti dell'utente sul sistema a basso livello. iOs è sviluppato per fare in modo che l'utente non debba avere neanche la minima conoscenza in ambito informatico o tecnologico per poter utilizzare il sistema operativo e le sue applicazioni. Non vi è accesso diretto al file system né alle directory che lo compongono, l'utente può interagire solo con il "livello utente" del sistema. Le modifiche che egli può apportare sono quindi limitate a semplici configurazioni e all'attivazione/disattivazione delle funzionalità dello stesso.

Caratteristiche del file system e dati utente

Non è presente una "home" del file system per i dati dell'utente. Il sistema operativo, dall'esterno, risulta essere un insieme di applicazioni (dette *App*). La configurazione stessa del sistema è gestita come un'applicazione. Ogni *App* gestisce i dati dell'utente ad essa collegati, non vi è una cartella "Documenti", se l'utente vuole utilizzare un file di testo dovrà aprirlo e salvarlo utilizzando un'*App*, e all'interno del suo spazio dati verrà conservato il documento.

I documenti vengono quindi "sparsi" nel sistema utilizzando come riferimento l'applicativo che li gestisce. L'unico modo per introdurre files all'interno del dispositivo è tramite una delle interfacce create con questo scopo, che possono andare da strumenti cloud come il già citato iCloud, a interfacce di scambio dati specifiche di determinate *App*, gestite tramite il programma iTunes.

Cloud e backup

Risulta comunque evidente che la direzione intrapresa in ambiente iOs è quella del salvataggio dati non sul dispositivo ma su sistemi cloud, e anche questa operazione vuole essere progettata per escludere l'utente dalla gestione diretta in favore di un'automatizzazione che fornisca un prodotto "autonomo" che non richiede manutenzione.

Le operazioni di backup del dispositivo sono infatti idealmente divise in tre fasi indipendenti:

- Salvataggio dei dati multimediali (foto, video e musica), che inizialmente utilizzano la memoria del dispositivo, per poi venire salvati tramite internet su appositi e proprietari servizi che gestiscono la memorizzazione permanente dei dati e lo scambio fra dispositivi diversi e utenti diversi

- "Streaming Foto" per quanto riguarda foto e video

- "iTunes Match" per il reparto musicale

Il fine di questi servizi è fornire all'utente la disponibilità di tutta la propria *libreria multimediale* online, con il salvataggio e memorizzazione diretta sul dispositivo solo dei documenti utilizzati di volta in volta, così da limitare la necessità di spazio fisico sul dispositivo.

Per quanto riguarda le App, vengono resi indipendenti la parte di dati utente e la parte dell'applicativo in sé. Il file di installazione risulta disponibile in cloud su iTunes Store (negozio virtuale di Apple fornitore unico di applicativi per iOs) tramite l'account personale dell'utente, a cui vengono quindi associati gli applicativi acquistati. Una volta scaricato per la prima volta un programma, questo rimane per sempre proprietà dell'utente e può quindi essere riottenuto in qualsiasi momento fosse necessario una nuova installazione. Risulta quindi evidente che l'utente non ha mai la necessità di salvare o ripristinare manualmente i file che compongono gli applicativi.

Per quanto riguarda la configurazione e i dati relativi a un'App, questo pacchetto di informazioni viene salvato in automatico in rete tramite iCloud, con al suo interno tutte le informazioni per restituire all'utente, dopo una ipotetica cancellazione totale dei dati del dispositivo, un'App esattamente nello stesso stato precedente al reset, compresi tutti i documenti esterni che erano stati caricati e utilizzati al suo interno.

Multitasking

A partire dalla versione 4, iOs permette un utilizzo multitasking *parziale*[9]. All'utente è infatti permesso di utilizzare un'App alla volta, visualizzata a tutto schermo, ma risulta possibile passare da un applicativo ad un altro in modo diretto, con il sistema che si occupa in automatico di salvare e mantenere disponibile lo stato dell'applicazione nel momento dello "switch". Ogni applicazione aperta rimane quindi sempre disponibile in memoria mantenendo lo stato del momento in cui viene messa in pausa.

Tramite l'utilizzo di dieci APIs per le operazioni in background (sette introdotte con la versione 4 e tre con la versione 5) risulta inoltre possibile:

- Continuare ad ascoltare musica o l'audio di un video anche se l'applicazione non è più attiva in primo piano.
- Effettuare o ricevere una chiamata e tornare, al termine di essa, esattamente allo stato in cui si trovava il cellulare prima dell'evento.
- Gestire i cambiamenti di ubicazione geografica a prescindere dall'applicativo in uso.
- Ricevere notifiche da applicativi non attivi.
- Ricevere notifiche basate sulla posizione geografica.
- Scaricare contenuti con applicativi attivi in background.
- Comunicare con dispositivi esterni in modo indipendente dall'App in uso.

1.2.3 Android

Android è il sistema operativo per dispositivi mobili creato e sviluppato da Google Inc. Detiene con largo margine il titolo di sistema operativo per smartphone più diffuso nel mondo coprendo quasi il 70% del totale, con percentuali in ascesa rispetto al 2011.

Caratteristiche chiave

Le caratteristiche che distinguono con forza il SO made-in-Mountain View dai competitor sono sostanzialmente 2:

- Si tratta di un sistema operativo Open Source
- E' sviluppato in modo indipendente dalle piattaforme su cui è installato

Indipendenza dalla piattaforma hardware

L'azienda di Mountain View si occupa unicamente di sviluppare il sistema operativo, lasciando inoltre libertà alle aziende produttrici dei device di modificarlo e personalizzarlo a piacimento.

Esiste infatti una versione di Android denominata *AOSP (Android Open Source Project)* che rappresenta l'"originale", ciò che viene fornito e sviluppato da Google.

Da questa base ogni produttore ottiene poi una propria versione che monta sui dispositivi che produce, creando di fatto una moltitudine di versioni dello stesso sistema operativo che condividono lo stesso funzionamento di base, lo stesso kernel, la compatibilità con le stesse applicazioni, lo stesso filesystem, ma con applicativi base di sistema diversi e sopratutto con un'interfaccia utente completamente diversa.

Da questo derivano tutti i principali pregi e i principali difetti di Android.

Grazie alla sua natura opensource, di fatto, Android è il sistema operativo mobile più utilizzato sul mercato [10], coprendo una fascia di mercato totale, dal prodotto economico entry level al telefono top di gamma di ultima generazione e dalle caratteristiche ai massimi livelli. Per utilizzare iOs serve necessariamente un iPhone, un prodotto di fascia alta, non c'è alternativa. Per utilizzare Android si può scegliere lo smartphone della categoria che si predilige, a seconda delle proprie necessità e disponibilità economiche.

Grazie al concetto stesso di Open Source, il fatto che ogni azienda possa elaborare il sistema operativo porta a una maggiore possibilità di innovazione che, a seconda dei casi, può entrare a far parte dello sviluppo del sistema operativo nativo.

Dal punto di vista del consumatore, si ha il vantaggio che cambiando, per esempio, marca dello smartphone, ci si trova davanti a un device apparentemente diverso dal precedente, fornendo quindi novità che possono risultare un miglioramento rispetto a precedenti esperienze.

Questa impostazione porta però, com'è ovvio, dei lati negativi. Una fra tutte: *la frammentazione*[11].

La frammentazione di Android

Android è presente su 11,868 dispositivi diversi (solo un anno fa erano 3,997). Ognuno di questi dispositivi presenta risoluzione dello schermo, personalizzazioni del sistema operativo, caratteristiche hardware (memoria esterna, fotocamera, gps, touchscreen) e versione del sistema operativo, potenzialmente (e realisticamente) diverse dagli altri. Questo fatto risulta estremamente problematico sia dal lato utente che dal lato sviluppatore. Quest'ultimo, progettando un'applicazione, dovrà tenere in considerazione il maggior numero di variabili possibili, perchè ovviamente è suo interesse rendere fruibile il proprio lavoro al maggior numero di utenti possibili. Si deve quindi tenere in considerazione ogni possibile risoluzione dello schermo per progettare l'interfaccia grafica dell'App, si deve considerare la possibilità che un device non abbia caratteristiche hardware "popolari" come una memoria esterna (SD Card) o una fotocamera. Oppure che, pur essendo presente l'hardware che permette di catturare foto e video, quest'ultimo possa avere una risoluzione ottica totalmente diversa da un device all'altro o la possibilità o meno di filmare video in risoluzione full HD con un framerate di 30fps.

Queste diversità portano quindi a fare delle scelte e a accettare compromessi. Il risultato è che non tutte le applicazioni create per Android sono compatibili con tutti i dispositivi che utilizzano questo sistema operativo, e non tutte le funzionalità di una applicazione sono disponibili in tutti i device in cui essa può essere installata.

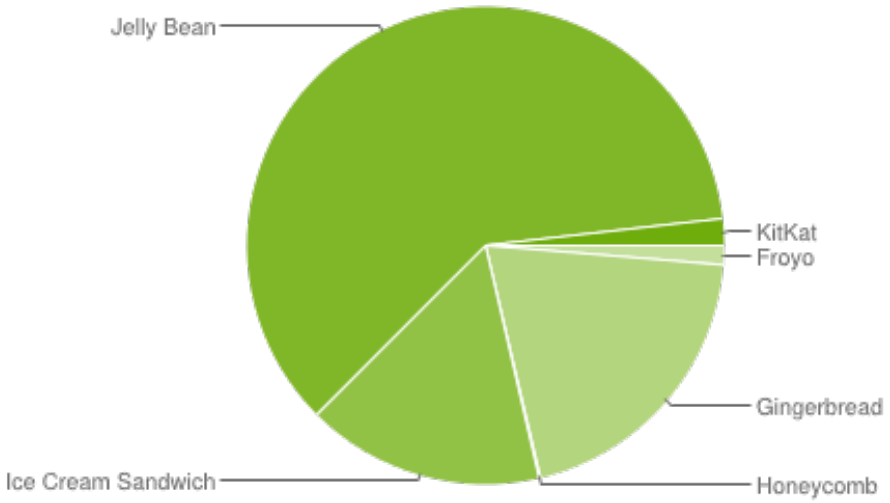
Oltre alle differenze hardware, un altro fondamentale scoglio portato dalla frammentazione è rappresentato dalle diverse versioni del sistema operativo e di conseguenza delle API utilizzabili per lo sviluppo.

Tabella 1.1: Distribuzione API di Android - 4 Febbraio 2014 [12]

Versione	**Nome in Codice**	**API**	**% di Distribuzione**
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	20.0%

3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.1%
4.1.x	Jelly Bean	16	35.5%
4.2.x		17	16.3%
4.3		18	8.9%
4.4	KitKat	19	1.8%

Figura 1.1: Analisi effettuata su un periodo di 7 giorni terminato il 4 Febbraio 2014. Le versioni con meno dello 0.1% di diffusione non sono elencate [12]



Dai dati risulta evidente la grossa fetta di utenza ancora ferma a Gingerbread, questo costringe gli sviluppatori a continuare a utilizzare APIs ormai antiquate e superate per garantire la diffusione del software, frenando l'evoluzione delle applicazioni che rimangono quindi indietro rispetto allo sviluppo del sistema operativo.

Il problema è inoltre accentuato dalle politiche commerciali dei vendor, che sempre più spesso tendono ad abbandonare i device più vecchi non fornendo agli utenti gli aggiornamenti alle nuove versioni di Android, per incentivare la sostituzione degli smartphones in favore dei nuovi prodotti venduti

dall'azienda stessa. Il risultato è ovviamente un numero sempre crescente di dispositivi che rimangono bloccati a vecchie versioni del sistema operativo, accentuando il problema della frammentazione e del freno allo sviluppo di APIs e App.

Per quanto riguarda il "lato utente", una delle principali modifiche che le aziende apportano al sistema operativo montato sui propri devices è quello dell'interfaccia grafica, detta *launcher*, ovvero la parte del sistema con cui l'utente interagisce costantemente. Alcuni esempi di interfacce grafiche proprietarie sono *HTC Sense*, dell'omonima azienda taiwanese, o *TouchWiz*, sviluppata da Samsung Electronics. Entrambe hanno la stessa funzione, ma la svolgono in modi completamente diversi, portando quindi un utente che ha preso dimestichezza con uno smartphone Android prodotto da un'azienda, a trovarsi completamente spaesato e inesperto davanti a un device prodotto da un'azienda concorrente, seppur equipaggiato con la stessa versione di Android.

Non vi è quindi una *user experience* omogenea, ogni azienda fornisce devices simili fra loro ma spesso estremamente diversi da smartphones di produttori concorrenti che pur condividono lo stesso sistema operativo.

Questo fa sì che la curva di apprendimento di un nuovo telefono sia indubbiamente più orizzontale nel caso di Android, nonostante questa peculiarità possa risultare un pregio se si considera la varietà come una risorsa e non una problematica.

Facilità di utilizzo

Analizzando le carattistiche generali del sistema operativo, si parte incontrando uno dei pregi di quest'ultimo in materia di facilità di utilizzo.

Uno smartphone Android appena uscito dalla confezione, se acceso, accoglierà l'utente con una sequenza di setup automatizzato che richiederà di inserire alcuni dati personali e l'account Google. Questo è infatti uno dei punti di forza dei sistemi Android: l'integrazione con tutto l'ecosistema di Google.

Inserendo i dati del proprio account non solo ci si trova a poter utilizzare tutta una serie di applicativi di base dei vari servizi Google (Gmail, Google Calendar, Google Music, Google Maps, Google+, ecc.) ma si ottiene una sincronizzazione completa e costante di tutti questi servizi su qualsiasi piattaforma o dispositivo. La rubrica del cellulare viene popolata automaticamente con i contatti memorizzati nell'account Gmail e con essi viene costantemente tenuta in sincronia. Non vi è più la necessità di tenere copia di sicurezza dei propri contatti telefonici perché essi sono sempre disponibili accedendo da un qualsiasi dispositivo al proprio account email. Stesso discorso per l'agenda e gli impegni presenti nel calendario, tutto viene costantemente sincronizzato, salvato, aggiornato. Tutto è sempre disponibile da idealmente qualsiasi dispositivo che abbia la capacità di aprire pagine web. Il tutto, automaticamente, solo inserendo in fase di primo avvio i dati del proprio account.

Sincronizzazione e interfacciamento PC

Android in questo settore presenta indubbiamente dei grossi vantaggi rispetto ai propri competitors. Tutti i dati sincronizzati tramite l'account Google sono accedibili tramite protocolli standard come Exchange, dai contatti agli eventi del calendario (che oltretutto supportano protocolli specifici come CalDav e CardDav per i contatti) tutti i dati salvati nel *cloud* di Google sono usufruibili da qualsiasi device o personal computer, indipendentemente dal sistema operativo, tutto questo grazie all'utilizzo di standard riconosciuti.

Superata quindi la fase di primo avvio e sincronizzazione con i propri dati, alla prima accensione ci si trova immediatamente nelle condizioni di utilizzare in tutto e per tutto il proprio dispositivo senza necessità di ulteriori operazioni di setup, traguardo a cui iOs è giunto solo recentemente.

Le operazioni di aggiornamento del sistema risultano estremamente diversificate a seconda del produttore dello smartphone. Nella maggior parte dei casi è prevista una procedura automatizzata, con notifica della presenza di nuovi aggiornamenti, download automatico e installazione. Il tutto è però

totalmente dipendente dalla casa produttrice, che determina se e quando un determinato modello riceverà un aggiornamento di sistema, con il risultato di tempi di attesa generalmente molto superiori alla norma e buone probabilità di vedere interrotti gli aggiornamenti dopo i primi mesi di vita del dispositivo.

Le operazioni di backup incluse nativamente nel sistema si limitano alla sincronizzazione dei dati con i servizi google, quindi rubrica, calendario, email, configurazioni di sistema, vengono in automatico salvati durante il normale utilizzo. Per tutto il resto dei dati, da quelli personali dell'utente alle applicazioni, è necessario utilizzare applicativi di terze parti. L'unica eccezione sta nel salvataggio delle applicazioni, che come nel caso dell'App Store di Apple, dopo l'acquisto vengono registrate nell'account Google e quindi rese disponibili in ogni momento per il download dallo store di Google, denominato Play Store. In caso di ripristino del dispositivo sarà quindi possibile recuperare tutti gli applicativi precedentemente utilizzati, ma saranno perse le impostazioni, i dati di utilizzo e i file dell'utente memorizzati all'interno dell'applicazione stessa.

File system

Il file system alla base di Android risulta essere un'altra delle fondamentali differenze rispetto agli altri sistemi mobile.

Unix è alla base del progetto, e ne determina le caratteristiche base come la struttura ad albero e i principali componenti. Inoltre, Android risulta anche completamente aperto all'intervento dell'utente, con alcune minime limitazioni. É infatti possibile accedere alle cartelle del file system, con l'unico limite di non poter eseguire operazioni che richiedano privilegi di amministratore.

L'utente ha quindi la propria cartella *home*, dove può liberamente salvare file di qualsiasi tipo in modo totalmente indipendente dagli applicativi utilizzati.

Collegando lo smartphone tramite USB a un qualsiasi personal computer è possibile navigare direttamente l'intero file system come se si stesse utiliz-

zando un dispositivo di memorizzazione esterno. Tramite semplici drag&drop è possibile quindi immagazzinare sul dispositivo qualsiasi tipo di documento, da file video a file di installazione di applicativi per lo stesso sistema operativo.

Questa libertà di utilizzo porta ovviamente numerosi vantaggi in termini di usabilità, che vengono però sfruttati principalmente da utenti esperti e capaci di usufruire di tali opportunità, e al contrario può risultare problematica per chi, non in grado di comprendere la tecnologia in esame, risulterebbe facilitato da una minor possibilità di scelta per evitare il pericolo di operazioni dannose da parte dell'utente.

1.3 Il mercato delle applicazioni mobile

Secondo i dati del rapporto di Gartner Group del 19 settembre 2013 [13] il totale dei download di applicazioni mobili nel 2013 si attesta intorno ai 102 miliardi, un aumento di circa il 60% rispetto all'anno precedente ed è previsto un continuo aumento nei prossimi anni, anche se con tassi di crescita lievemente in ribasso a causa della saturazione degli utenti che, superata una prima fase di evoluzione verticale, tenderanno sempre più a stabilizzare le proprie abitudini sulle applicazioni che conoscono, diminuendo progressivamente l'abitudine di scaricare nuove applicazioni.

Apple Store e *Google Play Store*, insieme, raggiungeranno circa il 90% del totale dei download entro il 2017. Queste 2 aziende risultano essere indiscutibilmente le dominatrici del mercato, grazie sopratutto a una diffusione dei dispositivi nettamente più ampia dei concorrenti e a community di sviluppatori ormai stabili e affermate.

Fonti di Guadagno per gli Sviluppatori Circa il 90% del totale delle applicazioni è distribuita in modo gratuito. Il Play Store di Google risulta il più conveniente con ben l'80% del totale mentre l'App Store Apple si ferma a circa il 60% del totale.

La tendenza riscontrata dal rapporto evidenzia però che le fonti di guadagno per gli sviluppatori si stanno velocemente spostando dal prezzo di vendita dell'applicazione verso il business degli *Acquisti In-App (IAP)*.

Se nel 2014 il totale dei guadagni degli sviluppatori derivanti dalle IAP si limitava al 17%, è prevista una crescita entro il 2017 che raggiungerà quasi il 50% del totale degli introiti.

Nella stessa direzione sta andando anche il settore del *mobile gaming* che sta vedendo una larghissima diffusione dei giochi così detti *freemium*, caratterizzati dalla gratuità dell'applicativo che spinge gli utenti a testala, per poi invogliare questi ultimi al pagamento di denaro per ottenere funzioni aggiuntive o agevolazioni sull'evoluzione del gioco.

Questi dati mostrano chiaramente quanto il settore sia continuamente in crescita (seppur in modo meno verticale rispetto agli ultimi anni) e in evoluzione.

Gli IAP risultano in modo evidente una soluzione alla problematica di saturazione del mercato delle App, in quanto se prima le possibilità di guadagno consistevano nel costo dell'applicazione, perché i nuovi utenti tendevano ad acquistarne molte per via della *novità* del settore, ora si punta sul fatto che l'utente medio continui ad usare le applicazioni che conosce e a cui si è abituato, portandolo a decidere di pagare per poter continuare ad utilizzare la stessa applicazione in modo facilitato rispetto allo standard.

Capitolo 2

Ambiente di sviluppo del progetto

In questo capitolo vengono illustrati i principali strumenti e le tecnologie utilizzate nel corso del progetto, iniziando dal sistema operativo Android, passando poi alle tecnologie relative ai database, locali all'applicazione e remoti, e concludendo con informazioni riguardanti i linguaggi di programmazione lato server e quelli utilizzati per le comunicazioni di rete.

2.1 Sviluppo dell'applicazione Android

2.1.1 Android

Android è un sistema operativo mobile, sviluppato inizialmente da Android Inc, società successivamente acquistata da Google. Attualmente Android è detenuta dalla Open Handset Alliance ed è interamente open source. Google rilascia la maggior parte del codice di Android sotto licenza Apache. Con questa licenza le aziende che rivendono device equipaggiati con questo sistema operativo possono aggiungere estensioni proprietarie senza doverli rilasciare alla community open source.

Ad ogni nuova versione di Android Google rilascia la versione AOSP (Android Open Source Project) e ogni azienda venditrice di smartphone rilascia

successivamente la stessa versione dopo aver eseguito modifiche e aggiunte al codice originale.

Android utilizza un kernel linux modificato e utilizza applicazioni sviluppate in java, sfruttando le librerie già esistenti con, in aggiunta, nuove librerie java sviluppate direttamente da Google per Android. Anche se le applicazioni sono scritte in Java, il sistema operativo non usa una Virtual Machine e non viene eseguito il byte code Java. Le classi Java sono ricompilate dentro eseguibili Dalvik e eseguite in una Virtual Machine Dalvik (vedi: 2.1.1).

Le applicazioni Android sono scritte in Java. L'SDK Android compila il codice, insieme a ogni dato e file risorsa, creando un Android package, un archivio con estensione .apk (vedi: 2.1.2). Tutto il codice in un singolo .apk fa parte di una singola applicazione.

Una volta installata in un dispositivo, ogni applicazione Android "vive" in una sandbox di sicurezza. Il sistema operativo Android è un sistema Linux multi utente in cui ogni applicazione è un utente diverso. Per default, il sistema assegna a ogni applicazione un user ID linux unico (l'ID è utilizzato solo dal sistema ed la singola applicazione non vi ha accesso e non lo "conosce") Il sistema definisce i permessi ai file di una applicazione utilizzando l'userID, in tal modo solo quella specifica applicazione può accedervi. Ogni processo ha una sua VM, in questo modo il codice di ogni singola applicazione viene eseguito in modo isolato rispetto alle altre applicazioni. Ogni appliczione viene eseguita in singolo processo, Android fa partire il processo quando un qualsiasi elemento dell'applicazione necessita di essere seguito, per poi eliminare il processo quando non è più richiesto o quando il sistema deve recuperare memoria per altre applicazioni

In questo modo, il sistema Android implementa il *principio del privilegio minimo*, ogni applicazione ha quindi accesso solo ai componenti richiesti per eseguire il proprio lavoro [14]. Questo crea un ambiente sicuro in cui una applicazione non può accedere a parti del sistema che non ha il permesso di utilizzare.

Ci sono comunque modi in cui una applicazione può condividere i dati

con altre applicazioni o accedere a servizi di sistema:

La prima problematica è risolta permettendo a due applicazioni di condividere lo stesso UserID, in questo caso queste hanno possibilità di accedere ai file l'una dell'altra. Per risparmiare risorse di sistema, applicazioni con lo stesso UserID possono essere eseguite nello stesso processo e condividere la stessa VM, ma per farlo devono essere firmate con lo stesso certificato

Infine, una applicazione può richiedere il permesso di accedere a dati del dispositivo (come i contatti dell'utente, gli sms e la scheda di memoria esterna) o a dispositivi come fotocamera, bluetooth, ecc. Questi permessi vengono richiesti all'utente in fase di installazione e possono essere in seguito ritirati.

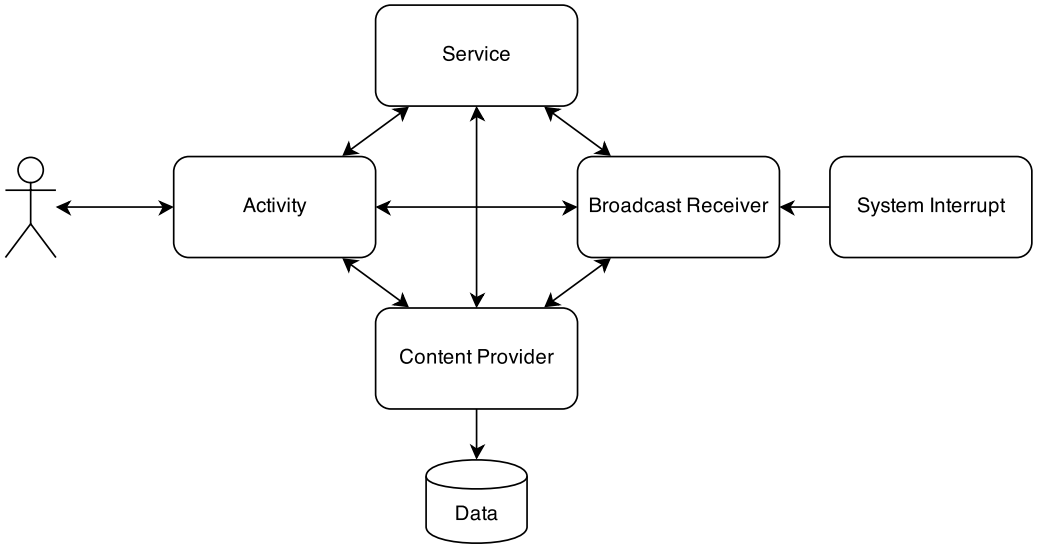
Componenti base di sistema

I componenti base di sistema sono i blocchi di cui è composta ogni applicazione Android. Ogni componente è un punto tramite il quale il sistema può accedere a una applicazione. Non tutti i componenti sono effettivamente dei punti di ingresso per l'utente e alcuni dipendono da altri, ma ognuno possiede un ruolo specifico, ognuno è un blocco unico che aiuta a definire il comportamento generale di una applicazione

Ci sono 4 tipi diversi di componenti, ogni tipo ha un diverso scopo e ha un diverso ciclo di vita che definisce come il componente viene creato e distrutto

Activities Una activity rappresenta una singola schermata con una interfaccia utente. Per esempio, una applicazione che gestisce le email avrà una activity che mostra una lista di nuove mail, un'altra activity per comporre una mail e una terza activity per leggere una mail. Ogni activity è indipendente dalle altre ma insieme creano tutta la user-experience di utilizzo dell'applicazione. L'utilizzo "classico" dell'applicazione vedrà un determinato susseguirsi delle activities, ma ognuna di esse può essere aperta direttamente all'avvio dell'applicazione, se permesso e definito dallo sviluppatore. Ad esempio, nel caso dell'applicazione per la gestione delle mail, potrebbe

Figura 2.1: Rappresentazione grafica dei componenti di base di Android [15]



essere possibile accedere direttamente all'activity di creazione di una nuova mail, nel caso in cui venga richiamata da una applicazione diversa che vuole inviare una foto o un qualsiasi altro tipo di dato tramite email.

Services Un servizio è un componente avviato in background per eseguire operazioni "di lunga durata" o per lavorare con processi remoti (server esterni o qualsiasi processo che non dia garanzie di risposte in tempi brevi). Un servizio non ha interfaccia grafica. Un classico esempio di processo può essere la musica riprodotta in background mentre l'utente utilizza un'applicazione diversa, oppure il download in background di dati senza bloccare l'interazione con una activity da parte del'utente. Come per le activities, i servizi possono essere avviati da altri componenti e utilizzati limitatamente al tempo in cui sono necessari, per poi essere interrotti.

Content provider Un Content Provider è un gestore di dati di applicazioni. La logica di questo componente è fornire a una applicazione (o a più di una nel caso in cui i dati siano condivisi) l'accesso a dati, indipendentemente da come questi dati siano immagazzinati e gestiti. Un content provider

fornirà la stessa interfaccia per la gestione dei dati sia che questi risiedano in un database SQLite, sia che siano salvati sul web, o su un qualsiasi altro tipo di supporto. Funziona quindi come una black box, ogni accesso ai dati passa attraverso il content provider. Questa caratteristica porta numerosi vantaggi, ad esempio nella gestione con database SQLite si vanno ad evitare tutti i problemi di concorrenza nell'accesso al DB e ai possibili bug relativi a duplice copia dell'istanza del database. Inoltre fungendo da layer che divide il livello di "utilizzo" dei dati dal livello di gestione e salvataggio degli stessi, permette di effettuare modifiche su quest'ultimo senza che la parte software che utilizza i dati ne risenta.

BroadCast receivers Un broadcast receiver è un componente che risponde a un segnale emesso dal sistema. L'esempio classico di broadcast è il segnale da parte del sistema che lo schemo è stato spento, la batteria è scarica o che è stata scattata una fotografia. Il segnale è creato in seguito a un certo evento o situazione e trasmesso a tutti i componenti di sistema senza distinzione, se è presente un broadcast receiver impostato per rispondere a quel segnale, verranno eseguite le operazioni previste. Un segnale broadcast può essere emesso anche da applicazioni quando queste vogliono segnalare agli altri componenti che si è verificato un determinato evento e permettere ad essi di interagire. Un broadcast receiver non ha una user interface, può al massimo creare una notifica nella status bar per avvertire l'utente del verificarsi di un evento. Generalmente questi componenti vengono utilizzati come gateaway, in pratica il loro lavoro è limitato all'accorgersi del verificarsi di un fatto e far partire un altro componente in risposta. Il lavoro fatto dal receiver è quindi minimo ma fondamentale per poter implementare un sistema composto da parti che interagistono tra loro in modo *event-driven*.

Dalvik

Dalvik è una VM modificata per Android e ottimizzata per dispositivi con cpu poco potenti e alimentati a batteria [16]. É il software che esegue

le applicazioni sui dispositivi Android. Le applicazioni sono scritte in java e compilate in bytecode, poi convertite da file .class compatibili con la JVM in file .dex compatibili con la Dalvik VM. Dalvik è un software open source, scritto da Dan Bornstein, dipendente di Google.

Una delle principali differenze rispetto alla JVM è che mentre quest'ultima ha una architettura basata sullo stack, la DVM usa una architettura basta su registri.

Esiste un tool chiamato "dx" che converte (non tutti) i file .class di java in file .dex equivalenti per DVM che spesso racchiudono un insieme di classi. Viene effettuata una ottimizzazione anche dal punto di vista del bytecode, per esempio andando a salvare in un singolo file costanti, stringhe e altre risorse che in java venivano replicate in classi diverse, il tutto con lo scopo di risparmiare spazio e ottimizzare le prestazioni. Viene inoltre convertito lo stesso bytecode che quindi fa risultare quello dei .dex diverso da quello dei .class di java. All'interno della MV Dalvik viene usato un compilatore JIT.

Il successore della Dalvik VM nei sistemi Android è la ART VM, introdotta con Android 4.4. Utilizza un processo *AOT (Ahead-of-time)* che pre-compila il bytecode nel codice macchina nel momento in cui l'applicazione viene installata. In questo modo il tempo di esecuzione viene ridotto della metà, comparato con la Davlik. Uno degli ovvi svantaggi è un tempo maggiore per l'installazione delle applicazioni e una maggiore occupazione di memoria (circa il 10-20%) a causa del bytecode precompilato. Va però evidenziato che seppur lo spazio disco è una risorsa che non abbonda nei dispositivi mobili, ciò su cui si punta maggiormente per migliorare l'esperienza utente è senza dubbio la velocità di esecuzione delle applicazioni, in questo senso i vantaggi portati dalla ART-VM seguono questa filosofia portando, a conti fatti, svantaggi trascurabili.

2.1.2 Android SDK

Eclipse con ADT

L'SDK di Android fornisce una ricca serie di tools: debugger, librerie, emulatori, documentazione, codice di esempio e tutorials. Le applicazioni Android possono essere sviluppate utilizzando Eclipse tramite l'aggiunta di un plu-in chiamato Android development Tools (ADT). L'appoggiarsi a Eclipse permette quindi di utilizzarne tutte le caratteristiche più utili, come content assit, ricerca, possibilità di utilizzare plug-in ulteriori, integrazione con JUnit, utilizzo di Javadoc, e diverse altre. Si vuole quindi sfruttare un IDE consolidato e potente come Eclipse come base a cui aggiungere le caratteristiche tipiche dello sviluppo mobile per Android.

2.2 Gestione dei database

2.2.1 SQLite

SQlite è una libreria che implementa un motore SQL transazionale serverless, autonomo e che necessita di poca configurazione. Il codice è di pubblico dominio ed utilizzabile gratuitamente sia a livello privato che commerciale [17].

Differentemente dai classici database SQL, SQLite non ha un processo server separato, legge e scrive direttamente su normali file disco. Un intero Database con tabelle, indici, triggers e viste è contenuto in un singolo file che ha il vantaggio di essere cross-plaform e utilizzabile indipendentemente dall'archietttura. Queste caratteristiche fanno di SQLite non tanto una alternativa ai principali DBMS come Oracle ma bensì una valida soluzione come formato-file per il salvataggio dei dati da parte di applicazioni.

I vantaggi di questa scelta sono molteplici:

1. Vengono automatizzate le funzioni di scrittura e lettura dei dati, evitando al programmatore di dover scrivere funzioni di debug e controllo.

2. L'accesso ai dati viene fatto tramite query SQL che semplificano quindi la complessità del codice per eseguire queste operazioni.

3. Il sistema è estendibile e modificabile non dando quindi vincoli sulla manipolazione dei dati.

4. Si utilizza un solo file per racchiudere informazioni di diversa natura che altrimenti sarebbero state conservate ognuna in un file differente.

5. *I dati sono indipendenti dal livello software che li ha salvati e possono quindi essere letti, modificati e utilizzati da diversi applicativi indipendenti fra loro e senza necessità di conoscere le caratteristiche del codice che ha generato i dati.*

6. Vengono caricati in memoria solo i dati necessari, ottimizzando la trasmissione dei dati e i tempi di lettura e di scrittura.

7. Vengono risolti automaticamente problematiche di accesso concorrenziale ai dati, diversi applicativi possono accedere contemporaneamente alla stessa fonte dati che si occupera autonomamente di gestire la concorrenza e garantire le proprietà A.C.I.D. e la consistenza dei dati.

8. Si hanno a disposizione le classiche feature dei DBMS relazionali come processi di undo/redo cross-sesssion ottenuti utilizzando i triggers.

SQLite si definisce *Self-Contained*: richiede minimo supporto da librerie esterne e dal sistema operativo. Questo lo rende ottimizzato per dispositivi portatili e che non hanno i vantaggi delle infrastrutture dei computer desktop e per le applicazioni che necessitano di esser eseguite su diverse macchine con configurazioni diverse, senza che queste ultime debbano essere modificate appositamente.

SQLite è scritto in ANSI-C ed è quindi facilmente compilabile da ogni compilatore C standard. Utilizza appena 7 funzioni C (memset(), memcpy(), memcmp(), strcmp(), malloc(), free(), realloc()). L'utilizzo della malloc() è inoltre sostituibile con un buffer statico in-place tramite opportuna configurazione. Vi

è inoltre il supporto a funzioni di date-and-time fornite da librerie C supplementari ma queste possono essere omesse in fase di compilazione se non necessarie. SQLite può essere eseguito in multi-threading e sfrutta le librerie mutex per garantire il corretto funzionamento relazionale e transazionale del database.

La dimensione della libreria con tutte le funzionalità abilitate può arrivare ad essere inferiore a 0,5Mb (a seconda della piattaforma e delle ottimizzazioni del compilatore). Sono ridottissimi anche i requisiti di stack-space (4Kb) e di heap-space (100Kb), rendendo SQLite un ottimo motore relazionale per database per dispositivi con poca memoria come gli smartphone, ottenendo uno dei migliori rapporti prestazioni/risorse in questi scenari d'uso.

La comunicazione fra SQLite, il sistema operativo e il livello di memorizzazione dei dati avviene tramite un layer VFS (Virtual File System) che potrà quindi essere ottimizzato per dispositivi portatili. Risulta solido anche il rispetto delle proprietà A.C.I.D, che vengono rispettate in ogni transazione anche in caso di crash di sistema o interruzione dell'alimentazione del dispositivo.

2.2.2 PhpMyAdmin

phpMyAdmin è un tool scritto in PHP, rilasciato gratuitamente su licenza open source, che si occupa dell'amministrazione di un database MySQL tramite interfaccia web[18]. É in grado di eseguire tutti i tipi di operazioni necessarie alla creazione, gestione e amministrazione di una base di dati, partendo dalla definizione e gestione dello schema fino all'esecuzione di query SQL e gestione di utenti e permessi

Fra le principali funzionalità offerte troviamo:

- Gestione tramite interfaccia web, con tutti i vantaggi che ne derivano come indipendenza dal sistema operativo e dal software utilizzato.
- Gestione di database MySQL.
- Importazione di dati e schema tramite file .CSV e .SQL.

- Esportazione di dati in numerosi formati fra cui: .CSV, .SQL, .XML, .PDF, ISO/IEC26300 - OpenDocument, Microsoft Word e Excel, LaTeX.
- Creazione di grafici in pdf del layout del database.
- Wizard per la creazione di queries complesse utilizzando Query-by-Example (QBE).
- Ricerca all'interno di un database.
- Grafici in tempo reale per il monitoraggio delle attività di un server MySQL come connessioni, processi, utilizzo delle risorse.

Utilizzo nel progetto Il tool phpMyAdmin è stato utilizzato per la creazione e gestione del database del CMS Esercizi2013 su un server locale utilizzato a scopo di testing e sviluppo. É stato caricato un dump del server di produzione e utilizzato quest'ultimo per definire le query utilizzate nelle diverse chiamate http-post effettuate dalla applicazione Android verso il servizio PHP.

2.3 Comunicazione client-server

2.3.1 JSON

JSON (JavaScript Object Notation) è un linguaggio di scripting usato per trasmettere dati strutturati in coppie chiave-valore. Generalmente utilizzato per la trasmissione di dati fra client e webserver, ha nei suoi punti di forza il ridotto utilizzo di spazio per memorizzare i dati e la leggibilità del codice e si pone come valida alternativa allo standard XML [19]. Rispetto a quest'ultimo, risulta avere la stessa espressività e potenzialità di trasmissione dei dati ma generalmente viene ridotta la lunghezza del codice risparmiando, ad esempio, sui tag di chiusura.

Nonostante abbia origine dal JavaScript, JSON risulta indipendente dai linguaggi di programmazione che ne sfruttano le funzionalità, è infatti possi-

bile generare e effettuare il parsing di codice JSON da moltissimo linguaggi esistenti.

JSON viene usato frequentemente nella programmazione in AJAX in quanto il suo uso tramite JavaScript è particolarmente semplice: l'interprete è in grado di eseguirne il parsing tramite una semplice chiamata alla funzione eval() (inizialmente progettata per valutare espressioni JavaScript). Questa caratteristica è stata una delle chiavi della sua diffusione grazie al largo uso di JavaScript nel mondo web ma risulta essere anche una delle sue maggiori vulnerabilità. Al contrario del parsing nativo di JSON, l'interprete JavaScript esegue effettivametne il codice JSON per produrre oggetti JavaScript, la falla di sicurezza nasce dal fatto che JSON supporta un set di caratteri UNICODE superiore a quello gestito da JavaScript, che quindi può creare problemi durante l'esecuzione di caratteri non supportati.

Esistono ovviamente metodi più sicuri e più performanti per gestire il codice JSON, per esempio la funzione *JSON.parse()* che è stata sviluppata specificatamente per lavorare con codice JSON, ma vista la maggior diffusione di JavaScript è ancora molto frequente l'utilizzo della funzione eval() con tutte le problematiche di sicurezza che ne derivano.

Uno dei metodi per diminuire queste problematiche è la validazione tramite espressioni regolari prima dell'utilizzo della funzione eval().

Dal 2009 i browser web hanno introdotto funzioni native di encodingdecoding del codice JSON con l'effetto di limitare l'uso (e le relative problematiche) della funzione eval() e aumentare le performance in quanto le funzioni non devono può subire il processo di parsing.

2.3.2 PHP

PHP, che significa "PHP: Hypertext Preprocessor", è un linguaggio di scripting general-purpose Open Source molto utilizzato, è specialmente indicato per lo sviluppo Web e può essere integrato nell'HTML. La sua sintassi è basata su quella di C, Java e Perl, l'obiettivo principale del linguaggio è

quello di permettere agli sviluppatori web di scrivere velocemente pagine web dinamiche [20].

```
Example #1 Un esempio introduttivo

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Esempio</title>
    </head>
    <body>

        <?php
            echo "Ciao, sono uno script PHP!";
        ?>

    </body>
</html>
```

Invece di molti comandi per produrre HTML (come si è visto in C o Perl), le pagine PHP contengono HTML con codice incorporato che fa *qualcosa* (in questo caso, produce "Ciao, sono uno script PHP!"). Il codice PHP è delimitato da speciali istruzioni di elaborazione di inizio e fine `<?php` e `?>` che permettono di entrare e uscire dalla *modalità PHP*.

Ciò che distingue PHP da altri linguaggi di scripting del tipo client-side JavaScript è che il codice viene eseguito nel server, generando HTML che sarà dopo inviato al client. Il client dovrebbe ricevere i risultati dell'esecuzione dello script, ma non potrà conoscere qual'è il codice eseguito.

La cosa più interessante nell'uso di PHP è che si tratta di un linguaggio estremamente semplice per il neofita, ma che, tuttavia, offre molte prestazioni avanzate al programmatore di professione.

> Che cosa può fare PHP? Qualsiasi cosa [21]

PHP ha come obiettivo principale lo scripting server-side, per cui può fare tutto ciò che può fare un qualunque programma CGI, come raccogliere dati da un form, generare pagine dai contenuti dinamici, oppure mandare e ricevere cookies. Ma PHP può fare molto di più.

Esistono tre campi principali in cui vengono usati gli scripts PHP:

Lo scripting server-side Questo è il campo più tradizionale ed il maggiore obiettivo del PHP. Per fare questo lavoro occorrono tre cose

1. Il parser PHP (CGI o server module).
2. Un webserver ed un browser web.
3. Occorre avviare il server web con un'installazione di PHP attiva.

Si può accedere all'output del programma PHP con un browser web e vedere la pagina PHP tramite il server. Tutto ciò può essere attivato sul pc di casa se si desidera semplicemente provare la programmazione PHP.

Lo scripting di righe di comando Si può creare uno script PHP da usare senza alcun server o browser. Per usarlo in questo modo, l'unica cosa necessaria è un parser PHP. Questo tipo di utilizzo è ideale per gli scripts eseguiti con cron (sui sistemi unix o Linux) oppure il Task Scheduler (su Windows). Questi script possono essere utilizzati per semplici task di processamento testi.

Scrittura di applicazioni desktop Probabilmente PHP non è il linguaggio più adatto per scrivere applicazioni desktop, con interfaccia grafica, ma, se lo si conosce molto bene, e se si vogliono usare le sue caratteristiche avanzate in applicazioni client-side, si può anche utilizzare PHP-GTK per scrivere questo tipo di pogrammi. Allo stesso modo, c'è anche la possibilità di scrivere applicazioni cross-platform.

PHP può essere usato su tutti i principali sistemi operativi, inclusi Linux, molte varianti di Unix (compresi HP-UX, Solaris e OpenBSD), Microsoft Windows, MacOS X, MacOS Xserver, RISC OS, e probabilmente altri. Inoltre supporta anche la maggior parte dei server web esistenti: ciò comprende Apache, IIS, e molti altri, questo include qualsiasi server web che può utilizzare il binario PHP FastCGI, come lighttpd e nginx. PHP funziona sia come un modulo sia come un processore CGI.

Pertanto, con PHP si ha la libertà di scegliere praticamente qualsiasi sistema operativo e qualsiasi server web. Inoltre, si può anche scegliere se fare uso di una programmazione procedurale oppure orientata agli oggetti (OOP), o una combinazione di entrambe.

Con PHP non si è limitati soltanto ad un output in HTML. Le possibilità di PHP, infatti, includono l'abilità di generare immagini, files PDF e perfino filmati Flash al volo (utilizzando libswf e Ming). Si può generare facilmente qualsiasi testo, come XHTML e qualsiasi altro file XML. PHP può autogenerare questi file, e salvarli nel file system, piuttosto che eseguire un printing esterno, o creare server-side cache per contenuti dinamici.

Una delle caratteristiche più importanti e significative di PHP è la possibilità di supportare una completa gamma di database. Scrivere una pagina web collegata ad un database è incredibilmente semplice utilizzando una delle specifiche estensioni del database (mysql per esempio), o utilizzando un abstraction layer come PDO, o connettendosi a qualsiasi database che supporta lo standard Open Database Connection tramite l' estensione ODBC. Altri database possono utilizzare cURL o i socket, come CouchDB.

PHP fa anche da supporto per dialogare con altri servizi utilizzando i protocolli del tipo LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (in Windows) e innumerevoli altri. Si possono anche aprire network sockets ed interagire usando qualsiasi altro protocollo. Inoltre supporta l'interscambio di dati complessi WDDX tra, virtualmente, tutti i linguaggi di programmazione web. A proposito di interconessioni, PHP supporta l'istanziazione dei Java Objects e l'utilizzo di questi come oggetti PHP in modo trasparente.

PHP possiede utili caratteristiche per l'elaborazione testi, che includono la compatibilità alle espressioni regolari di Perl (PCRE), e molte estensioni e strumenti per analizzare e accedere ai documenti XML. PHP standardizza tutte le estensioni XML sulla solida base di libxml2, ed estende le caratteristiche aggiungendo il supporto SimpleXML, XMLReader e XMLWriter.

Capitolo 3

Fase di analisi del progetto

In questo capitolo si descrivono le fasi di analisi preliminari per l'organizzazione del progetto, cercando di descrivere e schematizzare le informazioni principali che erano disponibili all'inizio dei lavori. Si continua poi con una descrizione delle scelte implementative principali utilizzate per realizzare tutte le funzionalità richieste, descrivendo i motivi delle scelte e i benefici ottenuti. Nell'ultima sezione si analizzano più approfonditamente le basi di dati utilizzate la gestione dei dati, una locale all'applicazione con la propria controparte su server remoto, l'altra presente unicamente su server online e interrogata in tempo reale.

3.1 Analisi del progetto

3.1.1 Analisi dei requisiti

Scopo

Creare una applicazione per sistemi mobile che permetta agli studenti del corso di *Programmazione* di accedere ai servizi forniti dai portali web dei due corsi e agli operatori dei portali di effettuare le operazioni di gestione e correzione degli elaborati degli studenti.

Vincoli

Le funzionalità del portale sono distribuite su due CMS pre-esistenti:

Programmazione Viene fornito come "black-box": tramite pagina php pre-esistente è possibile ottenere i dati presenti nel database in formato JSON. I dati sono "sovrabbondanti" e solo parte di essi sono utili al fine di fornire l'accesso ai contenuti. La stringa JSON contiene inoltre il codice di versione dei dati presenti sul database, inviando questo valore come parametro POST in una chiamata al servizio, vengono forniti i dati cancellati, modificati o aggiunti dall'ultima modifica dei contenuti.

Non vi sono ulteriori informazioni su come i dati siano gestiti a livello server.

I file depositati sul server sono raggiungibili tramite URL pubblico tramite opportuna stringa definita dalle informazioni del singolo file. L'accesso ai file non è limitato in alcun modo.

Viene fornito l'accesso in modalità amministratore al tool di amministrazione del CMS ospitato su un dominio di testing, tramite il quale è possibile eseguire l'inserimento di dati nel CMS e testarne successivamente il caricamento sull'applicazione

Esercizi2013 Il committente fornisce il codice completo del CMS, scritto interamente in PHP. Non è fornita documentazione ma tramite il codice è possibile risalire ai metodi pre-esistenti con cui vengono gestiti i dati, oltre e sopratutto alle specifiche della base di dati su cui sono registrati. É quindi reso possibile un lavoro di sviluppo e testing su server locale con relativo database di test. É fornito un dump di alcuni dati esistenti sul server per le procedure di sviluppo e test.

Non è concessa la modifica di nulla di pre-esistente, viene però fornita la possibilità di aggiungere pagine php sul server che potranno gestire l'interfacciamento dell'applicazione.

Il sistema di autenticazione degli utenti dovrà essere duplicato ad-hoc per l'applicazione mobile, non è possibile utilizzare i metodi pre-esistenti.

L'accesso ai file consegnati dagli studenti è protetto, il webserver non permettere l'accesso libero ai documenti che quindi dovranno essere recuperati e restituiti tramite il servizio PHP creato per l'applicazione.

Requisiti

R1 L'applicazione deve permettere la visualizzazione delle voci (es. news) e dei materiali depositati sul CMS Programmazione e dare la possibilità di effettuare il download dei file allegati (es. dispense delle lezioni)

R1.1 I materiali si suddividono in:

1. Informazioni sull'organizzazione del corso, suddiviso in
   - Esame
   - Il Progetto
   - Esercizi Settimanali
   - Il Laboratorio
2. News
3. Dispense Lezioni, *divise per A.A.*
4. Dispense Laboratorio, *divise per A.A.*
5. Soluzioni Esercizi Laboratorio
6. Compiti d'Esame, *divisi per anno*
7. Esempi di Codice, suddivisi in:
   - Giochi
   - Istruzioni e Funzioni
   - Vari

R1.2 I punti 1, 3, 4, 6, 7 trattano dati con una suddivisione in sottocategorie. I punti 2 e 5 trattano dati omogenei.

É richiesto che tali suddivisioni vengano mantenute nella consultazione delle informazioni tramite dispositivo mobile.

R1.3 I punti 3, 4, 5, 6, 7 contengono singoli elementi associati a file che devono poter essere scaricati sul dispositivo dagli utenti.

R2 É richiesto che le informazioni contenute nel CMS Programmazione siano consultabili sul dispositivo in modalità offline, è quindi richiesta un salvataggio delle informazioni sul dispositivo in modo persistente.

R3 L'applicazione deve permettere il login di uno studente (i cui dati sono registrati sul CMS Esercizi2013).

R3.1 I dati richiesti per il login sono indirizzo e-mail e password.

R4 L'applicazione deve permettere il login di un operatore (i cui dati sono registrati sul CMS Esercizi 2013).

R4.1 I dati richiesti per il login sono username e password.

R5 L'applicazione deve permettere allo studente autenticato di accedere all'elenco degli esercizi settimanali e visualizzare lo stato delle proprie consegne per ogni esercizio.

R5.1 E' possibile effettuare più consegne per ogni esercizio, deve essere permessa la consultazione dei dettagli di tutte le consegne di ogni singolo esercizio.

R6 L'applicazione deve permettere a un operatore autenticato di accedere all'elenco di tutte le consegne non ancora corrette, accedere al dettaglio, scaricare il file contenente l'esercizio svolto consegnato dallo studente ed infine inviare la correzione.

R6.1 Il file che contiene l'esercizio svolto non è accessibile tramite webserver, deve quindi essere sviluppato il componente che permette all'applicazione di accedervi

R6.2 Una correzione consiste nel determinare se l'esercizio è stato superato o meno (*corretto* o *non corretto*), di un eventuale commento fornito dall'operatore, e dalla data in cui è stata effettuata la correzione

3.1.2 Analisi dei casi d'uso

Casi d'uso analizzati:

- CU1 Consultazione di una "News"
- CU2 Consultazione della dispensa di una lezione
- CU3 Consultazione dell'esercizio 1 della settimana 1 e delle sue relative consegne
- CU4 Consultazione di tutte le consegne non ancora corrette
- CU5 Correzione di una consegna

CU1 Consultazione di una "News"

Tabella 3.1: CU1

CU1: Consultazione di una "News"
Attore: Utente generico

Precondizione: Operazione di download delle informazioni dal CMS Programmazione
Sequenza Eventi: 1. L'utente apre il menu con uno *swipe* da sinistra verso destra o tramite l'apposito pulsante della barra superiore 2. L'utente seleziona la voce del menu *News* 3. L'utente seleziona con un *tap* la news che desidera leggere consultando l'elenco dei titoli 4. L'utente accede a una videata che fornisce i dettagli della news
Invariante: A partire dal punto 2 il caso d'uso termina in qualunque momento venga riaperto il menù e selezionata un'altra voce dell'elenco
Sequenza Alternativa:
Postcondizione:

CU2 Consultazione della dispensa di una lezione

Tabella 3.2: CU2

CU2 Consultazione della dispensa di una lezione
Attore: Utente generico
Precondizione: Operazione di download delle informazioni dal CMS Programmazione. Connessione a internet attiva per il download dell'allegato.
Sequenza Eventi:

1. L'utente apre il menu con uno *swipe* da sinistra verso destra o tramite l'apposito pulsante della barra superiore 2. L'utente seleziona la voce del menu *Dispense delle Lezioni* 3. L'utente seleziona con un *tap* l'A.A. di cui vuole consultare le dispense 4. L'utente seleziona con un *tap* la dispensa che desidera leggere consultando l'elenco dei titoli 5. L'utente accede a una videata che fornisce i dettagli della news 6. L'utente esegue un *tap* sull'apposito pulsante per avviare il download del file allegato
Invariante: A partire dal punto 2 il caso d'uso termina in qualunque momento venga riaperto il menù e selezionata un'altra voce dell'elenco
Sequenza Alternativa:
Postcondizione:

CU3 Consultazione dell'esercizio 1 della settimana 1 e delle sue relative consegne

Tabella 3.3: CU3

CU3 Consultazione dell'esercizio 1 della settimana 1 e delle sue relative consegne
Attore: Studente
Login effettuato con successo con credenziali di tipo *studente*. Connessione a internet attiva.
Sequenza Eventi:

1. Lo studente apre il menu con uno *swipe* da sinistra verso destra o tramite l'apposito pulsante della barra superiore 2. Lo studente seleziona la voce del menu *Esercizi* 3. Lo studente seleziona con un *tap* la settimana 1 4. Lo studente visualizza l'elenco di tutti gli esercizi della settimana 1 con relativo stato dell'ultima consegna (*Corretto* o *Sbagliato*), se effettuata 5. Lo studente seleziona con un *tap* l'esercizio 1 6. Lo studente accede a una videata che fornisce i dettagli dell'esercizio 1
Invariante: A partire dal punto 2 il caso d'uso termina in qualunque momento venga riaperto il menù e selezionata un'altra voce dell'elenco
Sequenza Alternativa: Se lo studente ha precedentemente inviato almeno una consegna dell'esercizio 1 della settimana 1, durante la visualizzazione dei dettagli dell'esercizio visualizza anche i dettagli della correzione Può inoltre, tramite l'apposito pulsante, accedere all'elenco di tutte le consegne relative al singolo esercizio
Postcondizione:

CU4 Consultazione di tutte le consegne non ancora corrette

Tabella 3.4: CU4

CU4 Consultazione di tutte le consegne non ancora corrette
Attore: Operatore
Login effettuato con successo con credenziali di tipo *operatore*.

Connessione a internet attiva.
Sequenza Eventi: 1. L'operatore apre il menu con uno *swipe* da sinistra verso destra o tramite l'apposito pulsante della barra superiore 2. L'operatore seleziona la voce del menu *Elenco consegne* 3. L'operatore visualizza l'elenco di tutte le consegne di tutti gli studenti, che non sono state ancora corrette
Invariante: A partire dal punto 2 il caso d'uso termina in qualunque momento venga riaperto il menù e selezionata un'altra voce dell'elenco
Sequenza Alternativa:
Postcondizione:

CU5 Correzione di una consegna

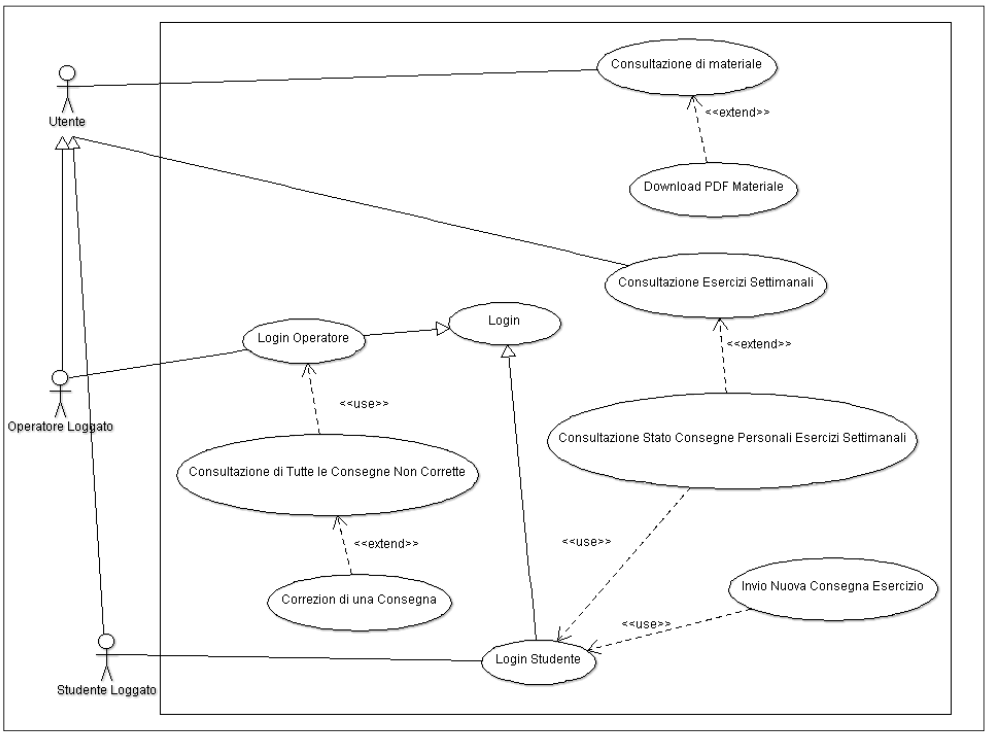
Tabella 3.5: CU5

CU5 Correzione di una consegna
Attore: Operatore
Login effettuato con successo con credenziali di tipo *operatore*. Connessione a internet attiva.
Sequenza Eventi:

1. L'operatore apre il menu con uno *swipe* da sinistra verso destra o tramite l'apposito pulsante della barra superiore
2. L'operatore seleziona la voce del menu *Elenco consegne*
3. L'operatore visualizza l'elenco di tutte le consegne di tutti gli studenti, che non sono state ancora corrette
4. L'operatore seleziona con un *tap* la consegna che desidera correggere
5. L'operatore visualizza i dettagli della consegna
6. L'operatore esegue un *tap* sul pulsante per eseguire il download del file contenente il codice della consegna
7. L'operatore visualizza tramite applicazione esterna il contenuto del file
8. L'operatore seleziona l'esito della consegna (*Corretto* o *Sbagliato*)
9. L'operatore esegue un *tap* sul pulsante di invio della correzione

Invariante: A partire dal punto 2 il caso d'uso termina in qualunque momento venga riaperto il menù e selezionata un'altra voce dell'elenco. Il caso d'uso termina nel caso in cui sul dispositivo non vi siano applicazioni in grado di aprire e visualizzare il file contenente il codice della consegna.
Sequenza Alternativa: L'operatore può inserire un commento prima di eseguire l'invio della correzione
Postcondizione:

Figura 3.1: Diagramma dei Casi d'Uso



3.1.3 Problematiche principali e obiettivi

Per concludere la fase di analisi si definiscono quelli che risultano essere le problematiche più critiche per lo sviluppo dell'applicazione e allo stesso tempo i punti su cui porre particolare attenzione perché essenziali per la buona riuscita del progetto.

Punti critici

Necessità di una procedura di login parallela a quella esistente L'esigenza di riconoscere un utente autenticato risulta essenziale per tre funzionalità fondamentali che si vogliono implementare

- Uno studente deve poter controllare lo stato delle proprie correzioni senza avere in alcun modo accesso a quelle dei propri colleghi.

- La consultazione di tutte le consegne e la modifica di alcune di esse per l'aggiunta della relativa correzione devono essere permesse unicamente agli operatori.

- Il download dei sorgenti consegnati dagli studenti deve essere permesso solo a un operatore o allo studente che ha inviato il file in origine.

É inoltre richiesto espressamente dal committente che la procedura di login sia totalmente separata da quella utilizzata tramite interfaccia web e attualmente a regime di funzionamento.

Si deve quindi implementare una procedura che permetta di autenticare gli utenti registrati tramite *username* e *password*. É inoltre molto importante fare in modo che un utente, una volta effettuato il login, rimanga autenticato anche dopo la chiusura e riapertura dell'applicazione o lo spegnimento del dispositivo, fino a esplicita richiesta di logout.

Si evidenzia che per l'accesso e la consultazione dei dati del CMS Programmazione non è richiesta alcun tipo di autenticazione.

Interfacciamento con una "Black-Box" per il reperimento dei dati di uno dei 2 CMS Risulta necessaria una attenta analisi dei dati per definire l'interfacciamento a questa fonte dati in modo da rendere l'applicazione il meno sensibile possibile a un eventuale cambiamento delle funzionalità lato server, o, in alternativa, è importante creare un software facilmente adattabile ai cambiamenti.

La stessa applicazione si trova a dover lavorare con dati provenienti da fonti diverse e con peculiarità diverse L'applicazione dovrà comunicare e elaborare dati provenienti da 2 distinte fonti

- Il CMS Programmazione contiene un considerevole numero di dati, con intervalli di aggiornamento mediamente lunghi, ed è esplicitamente richiesto dal committente che venga mantenuta una copia dei dati in locale per la consultazione offline, non è necessaria l'autenticazione per la consultazione dei dati e per il download dei file.

- Il CMS Esercizi2013 gestisce dati ad alta frequenza di aggiornamento, è necessaria l'autenticazione per l'accesso ai dati, il download dei file deve essere implementato interamente.

Obiettivi

Sviluppare una applicazione "moderna" Si vuole curare particolarmente la scelta dei pattern di programmazione e l'utilizzo delle metodologie più avanzate e coerenti con le indicazioni fornite dalla divisione *Development* di Google.

Progettare una applicazione flessibile Si vuole dare enfasi alla progettazione di un applicativo che possa essere evoluto e modificato in base ai cambiamenti del dominio applicativo e delle richieste del committente. Non è obiettivo del progetto l'implementazione di tutte le funzionalità possibili, bensì la creazione di una base solida che possa evolvere nel tempo senza dover subire stravolgimenti.

Larga compatibilità con versioni datate del sistema operativo Si vuole sviluppare un applicativo utilizzabile dal maggior numero di utenti possibili, di conseguenza è fondamentale considerare le problematiche derivanti dalla frammentazione di Android (vedi 1.2.3) e creare un applicativo il più possibile retrocompatibile, senza rinunciare alle funzionalità più avanzate.

3.2 Scelta delle tecnologie per l'implementazione

3.2.1 Content provider e content resolver

Un Content Provider è una parte di un'applicazione Android che si occupa di rendere disponibili dei dati alle altre applicazioni installate nel sistema.

Si occupa, sostanzialmente, di creare degli indirizzamenti univoci per le risorse che deve gestire, in modo che altri componenti del sistema possano effettuare operazioni sui dati conoscendo solo il loro *indirizzo* e nient'altro della loro implementazione e memorizzazione.

Ogni applicazione, pertanto, può definire una o più tipologie di dati e rendere poi disponibili tali informazioni esponendo uno o più Content Provider. Nell'ordine inverso, invece, qualunque applicazione può richiedere l'accesso ad un particolare tipo di dato: il sistema la metterà in contatto con il corrispondente Content Provider precedentemente installato nel sistema.

Content resolver La classe *ContentResolver* si occupa di mappare tutti i provider presenti nel dispositivo, basandosi sui file *manifest* dove vanno definiti, per ogni applicazione, i providers che essa espone al sistema.

Il funzionamento è simile a quello di un DNS, il content resolver riceve un URI ed è in grado di capire quale sia il content provider da avviare per fornire i dati richiesti.

Ogni Uri è così composto:

- Il prefisso *content:* indica che i dati sono gestiti da un content provider.
- La seconda parte è l'Authority, ovvero quella che si può paragonare al *domain name*, deve essere univoca, e identifica il singolo content provider. É prassi far sì che questa parte coincida con il package path dell'applicazione, in quanto deve anch'esso essere univoco per poter venire pubblicato su uno store di applicazioni.
- Il nome della risorsa a cui si fa riferimento.
- Eventualmente, in modo opzionale, può essere inserito un riferimento alla singola tupla di una tabella.

É il content resolver di una applicazione che si occupa di avviare e interrompere l'istanziazione e l'esecuzione dei content provider delle applicazioni del sistema, a seconda di quali di esse vengono richieste a runtime.

Utilizzo nel progetto In questo progetto si è scelto di utilizzare questo componente per strutturare l'applicazione in layer indipendenti. Il content provider funziona infatti da collegamento fra la parte che utilizza i dati (le listview contenute nei fragment) e il database che li contiene, rendendo le due parti invisibili l'una all'altra e indipendenti fra loro. Sarebbe quindi possibile, volendo, modificare la base di dati o il componente usato per memorizzare i dati senza modificare in nessun modo come la listview richiede e ottiene i dati dal content provider.

Inoltre, grazie a questo componente intermedio, vengono evitate una serie di problematiche relative alla gestione dei database, come per esempio la concorrenza. L'unico componente che utilizzerà il dabase sarà infatti il content provider, che riceverà e gestirà tutte le richieste di modifica e lettura dei dati.

Dal momento che si è scelto di immagazzinare i dati scaricati dal database remoto del CMS, strutturando fin da subito l'applicazione con questi metodi, si renderanno possibili future evoluzioni che permetteranno ad altre applicazioni di utilizzare i dati interni con minimi cambiamenti del codice e nessuna modifica alla logica applicativa.

Si descrivono quindi le principali caratteristiche dei content provider:

Ricerca dei contenuti Ogni tipo di contenuto esposto mediante Content Provider viene identificato attraverso un URI, cioè un indirizzo univoco. La forma tipica di questo genere di URI è

```
content://riferimento-di-base/bla/bla/bla
```

Android dispone di diversi Content Provider built-in, come quello per le immagini o quello per accedere ai contatti in rubrica. L'URI per accedere ai contatti, ad esempio, è:

```
content://com.android.contacts/contacts
```

Siccome questi URI sono arbitrariamente definiti dallo sviluppatore, per convenienza si è soliti fare in modo che qualche classe riporti l'indirizzo in maniera statica, in modo che non sia necessario digitarlo per esteso. I

Content Provider preinstallati in Android sono consultabili al package android.provider. In questo pacchetto, ad esempio, si trova la classe ContactsContract.Contacts, che è quella che fa da ponte per l'accesso al provider dei contatti in rubrica. Al suo interno c'è la costante statica CONTENT_URI, di tipo *android.net.Uri*, che riporta l'URI che identifica univocamente il provider.

Una volta che si conosce l'URI di una tipologia di contenuto, interagire con il provider che la eroga è più o meno come fare delle interrogazioni ad un database. Per prima cosa si deve recuperare un'istanza dell'oggetto android.content.ContentResolver. Nel codice di una Activity (o avendo a disposizione un oggetto android.app.Context) si può usare il metodo getContentResolver ().

Ad esempio:

```
1 ContentResolver cr = getContentResolver();
```

Gli oggetti ContentResolver permettono le interrogazioni attraverso il loro metodo:

```
1 public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String
      sortOrder)
```

I parametri da fornire sono i seguenti:

- Uri è l'indirizzo che identifica il tipo di contenuto ricercato.
- Projection è la lista con i nomi delle colonne i cui valori devono essere inclusi nella risposta. Se null vengono restituite tutte le colonne disponibili.
- Selection è la clausola WHERE. Se null vengono restituite tutte le righe disponibili.
- SelectionArgs è la lista degli argomenti della clausola WHERE al punto precedente.
- SortOrder è la clausola SQL di ordinamento. Se null non si applica un ordinamento specifico alla lista dei risultati restituiti.

Il tipo del risultato restituito è un oggetto android.database.Cursor, che possiamo sfogliare per sondare i record restituiti come risposta alla nostra query.

Il cursore restituito, in questo caso, potrebbe anche essere nullo: avviene quando l'URI fornito al metodo non corrisponde ad alcun provider registrato nel sistema. Per comporre query complesse, così come per prendere in esame i risultati restituiti, è necessario conoscere il nome ed il tipo delle colonne che costituiscono lo specifico tipo di dato richiesto. Anche in questo caso si è soliti includere tali informazioni all'interno delle costanti statiche di una classe. Nel caso della rubrica di sistema, ad esempio, le colonne disponibili sono elencate all'interno di ContactsContract.Contacts.

Per la gestione del cursore si è deciso di utilizzare i Cursor Loader (vedi: 3.2.2). Questo componente predefinito di Android richiede l'esistenza di un Content Provider per effettuare le operazioni di caricamento dei dati, infatti necessita unicamente dell'Uri della tabella contenente i dati, tutto il resto del lavoro di comunicazione con il database è demandato al Content Provider.

I Content Provider sono in grado di fornire anche funzionalità di inserimento, aggiornamento e cancellazione dei record (*C.R.U.D.*).

La classe ContentResolver, oltre al metodo query(), mette a disposizione i seguenti altri metodi:

- public Uri insert(Uri uri, ContentValues values) Inserisce un nuovo record del tipo specificato mediante il parametro uri. I valori per i campi del nuovo record devono essere specificati attraverso la mappa values, di tipo android.content.ContentValues. Il metodo restituisce l'URI di dettaglio assegnato all'elemento appena inserito.

- public int update(Uri uri, ContentValues values, String where, String[] selectionArgs) Aggiorna uno o più record del tipo specificato mediante il parametro uri. La selezione avviene attraverso l'uso combinato dei parametri where e selectionArgs. I nuovi valori da assegnare ai record selezionati devono essere specificati attraverso la mappa values, di tipo android.content.ContentValues. Il metodo restituisce il numero dei record aggiornati.

- public int delete(Uri uri, String where, String[] selectionArgs) Cancella uno o più record del tipo specificato mediante il parametro uri. La selezione avviene attraverso l'uso combinato dei parametri where e selectionArgs. Il metodo restituisce il numero dei record cancellati.

Creare un content provider Per creare un Content Provider bisogna estendere la classe android.content.ContentProvider, che richiede l'implementazione dei seguenti metodi:

- public boolean onCreate() Il codice da eseguirsi alla creazione del provider. Il metodo deve restituire un booleano: true, per segnalare che la creazione del provider è andata a buon fine; false, in caso contrario.

- public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) Il metodo richiamato per eseguire una ricerca tra i dati gestiti dal provider

- public String getType(Uri uri) Dato un URI di gestione del provider, questo metodo deve restituire il tipo MIME del contenuto corrispondente. Solitamente, con tipi di dati personalizzati, non bisogna far altro che inventare il nome della tipologia, seguendo però alcuni criteri. Se l'URI specificato corrisponde ad un contenuto specifico (in genere avviene quando l'URI contiene l'ID del contenuto), allora bisogna restituire un tipo MIME del tipo: vnd.android.cursor.item/vnd.il_nome_del_tipo

  Per gli URI che corrispondono a gruppi di contenuti (senza ID, quindi), la formula è del tipo: vnd.android.cursor.dir/vnd.il_nome_del_tipo

- public insert(Uri uri, ContentValues values) Il metodo richiamato per eseguire un inserimento nei dati gestiti dal provider.

- public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) Il metodo richiamato per eseguire un aggiornamento dei dati gestiti dal provider

- `public int delete(Uri uri, String selection, String[] selectionArgs)` Il metodo richiamato per eseguire una cancellazione fra i dati gestiti dal provider.

Una volta che il Content Provider è stato implementato, bisogna registrarlo nel file AndroidManifest.xml, osservando il seguente modello

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
<application ... >
...
<provider android:name="MioContentProvider"
android:authorities="mio_dominio/mio_tipo_di_dato" />
...
</application>
</manifest>
```

L'attributo name serve per specificare il nome della classe che implementa il provider; l'attributo authorities, invece, definisce la parte fondamentale dell'URI gestito dal provider, cioè quello che dovranno utilizzare le altre applicazioni per interagire con il nostro Content Provider. Ad esempio, facciamo il caso che autorities sia:

```
distributori.carburante
```

In questo caso, il Content Provider riceverà tutte le richieste il cui URI sia del tipo:

```
content://  distributori.carburante/italia/lazio/roma
```

Per concludere, è poi necessario comunicare a chi dovrà utilizzare il provider quale sia il suo URI di base e quali i nomi ed i tipi delle colonne di ciascun tipo di contenuto gestito. Solitamente conviene realizzare una o più classi che contengano queste informazioni, da rendere poi disponibili a chi dovrà servirsi del provider.

Content resolver Per accedere ai dati di un content provider si utilizza un oggetto client: il *Content Resolver* Questo oggetto ha metodi che richiamano altri metodi presenti all'interno del content provider che hanno la stesso identica signature. I metodi del content resolver forniscono le operazioni C.R.U.D. di base: Create, Retrieve, Update e Delete sui dati immagazzinati

dalla base dei dati del content provider (che non necessariamente sarà un database).

L'oggetto content resolver dell'applicazione client e il content provider dell'applicazione che possiede i dati, gestiscono automaticamente il processo di comunicazione. Il content provider, come già detto, funge semplicemente da layer astratto fra il repository dei dati e come questi vengono forniti all'esterno (ovvero nella forma di dati e tabelle)

3.2.2 Loaders

Un loader è un componente di Android che si occupa del caricamento asincrono di dati [22].

Le caratteristiche principali che lo rendono estremamente potente ed utile sono:

- É indipendente dalle activities e dai fragments ed è quindi disponibile in ognuno di questi.
- Esegue il caricamento dei dati in modo asincrono, senza quindi andare ad intaccare la reattività dell'interfaccia grafica.
- Si occupa di monitorare la sorgente dei dati accorgendosi di quando questi cambiano e effettuando in modo automatico un aggiornamento dalla sorgente alla destinazione.
- Nel caso di utilizzo con basi di dati e cursori, nei casi in cui l'activity viene modificata (per esempio durante una rotazione dello schermo) dopo essere stati ricreati si riconnettono in automatico al cursore che contiene i dati precedentemente caricati, evitando quindi di dover eseguire una seconda volta il caricamento dal database sorgente.

I loader sono indipendenti dalle activity che li utilizzano, e ogni activity utilizza un'istanza di *Loader Manager* per gestire i diversi loaders, che vengono generalmente caricati durante la creazione della stessa activity e possono essere distrutti, ricreati o resettati in qualsiasi momento.

Il trasferimento dei dati dal loader che li ottiene all'activity che li utilizza avviene in modo asincrono tramite l'utilizzo di callbacks.

In pratica l'activity si occupa di inizializzare il loader ed implementare i metodi che saranno richiamati da quest'ultimo nei diversi momenti del suo ciclo di vita.

Nel momento in cui, per esempio, il loader completa il caricamento dei dati da un database locale, viene invocato il metodo onLoadFinished che si occupa di trasferire l'istanza del cursore che contiene i dati all'adapter che si occuperà della loro visualizzazione.

Lo stesso adapter, nel momento della sua istanziazione, viene creato senza un riferimento al cursore contenente i dati (viene passato il valore *null* in fase di istanziazione dell'adapter) e solo successivamente, nel momento in cui il loader invoca il metodo onLoadFinished, viene passato all'adapter il riferimento al cursore.

In questo progetto si utilizzano due tipologie di loaders.

Per ottenere i dati memorizzati nel database locale si utilizza un *Cursor Loader* predefinito, che richiede l'implementazione di un Content Resolver (vedi: 3.2.1) che si occupi della gestione delle basi di dati.

Quando invece i dati vanno recuperati da un server remoto il real time, si rende necessario creare un loader custom che si occupi del caricamento dei dati e che implementi le operazioni di aggiornamento asincrono degli adapter che dovranno gestirli.

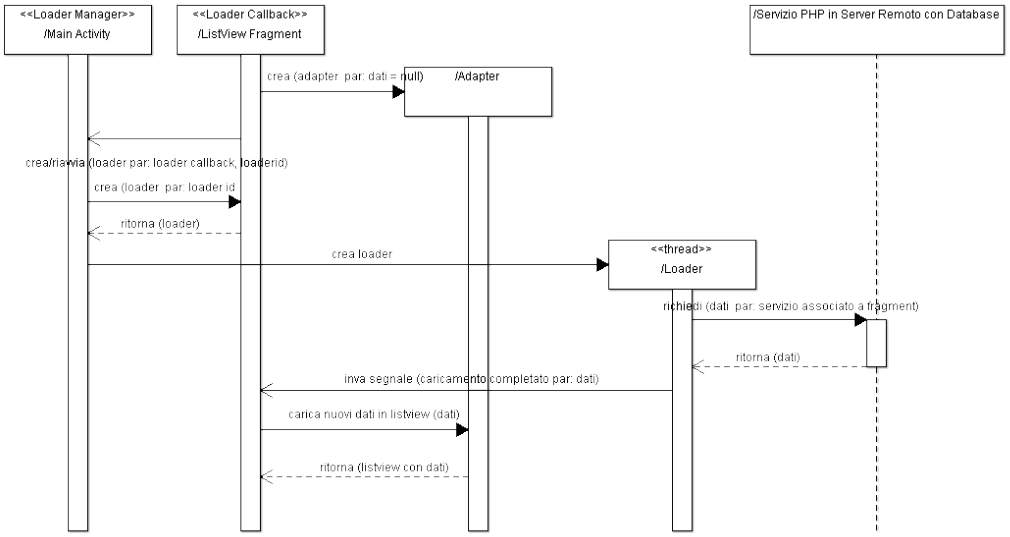
3.2.3 Fragments

La classe fragment è un elemento fondamentale per la composizione di una interfaccia grafica per una applicazione Android[23].

Mentre la activity è la radice di ogni schermata di una applicazione, il fragment ne rappresenta solo una porzione.

Il risultato, utilizzando questo componente, risulta una interfaccia grafica non più monolitica ma composta da sottoparti totalmente indipendenti l'una

Figura 3.2: Diagramma di Sequenza che illustra in modo semplificato l'interazione fra l'activity (in questo caso utilizzando un fragment al suo interno), il relativo adapter e il loader che effettua il caricamento dei dati da un server remoto



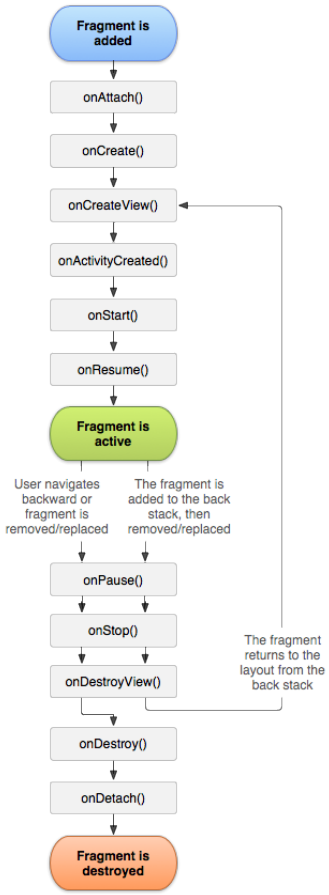
dall'altra, ogni fragment ha un proprio ciclo di vita (illustrato nell'immagine 3.3) e viene definito autonomamente.

Uno dei vantaggi fondamentali di questo design pattern consiste nella possibilità di modificare il layout grafico dell'applicazione a runtime, in base al tipo di device su cui si sta eseguendo l'applicazione. L'esempio più classico per rendere evidente questo vantaggio è rappresentato dalla situazione in cui si ha un menù e le relative finestre che si desidera visualizzare a seconda della selezione effettuata sulla lista del menù.

Se si sta eseguendo l'applicazione su uno smartphone, lo spazio di visualizzazione sarà ridotto, di conseguenza sarà preferibile visualizzare in modo alternativo o la lista o la vista di dettaglio dell'elemento selezionato.

Nel momento in cui l'applicazione viene eseguita su un tablet, ci troveremo ad avere uno schermo decisamente più ampio che è in grado di ospitare entrambe le viste contemporaneamente.

Figura 3.3: Il workflow illustra l'ordine di chiamata delle diverse funzioni che compongono il ciclo di vita di un fragment [23]



Se si utilizzano i fragment, la activity principale avrà la possibilità di richiamare le singole viste e visualizzarle nella maniera più consona allo spazio a disposizione, appunto perchè la activity in sè non è altro che un contenitore, e tutti i dettagli grafici sono implementati all'interno delle sottoparti, che quindi possono essere risistemate come più opportuno.

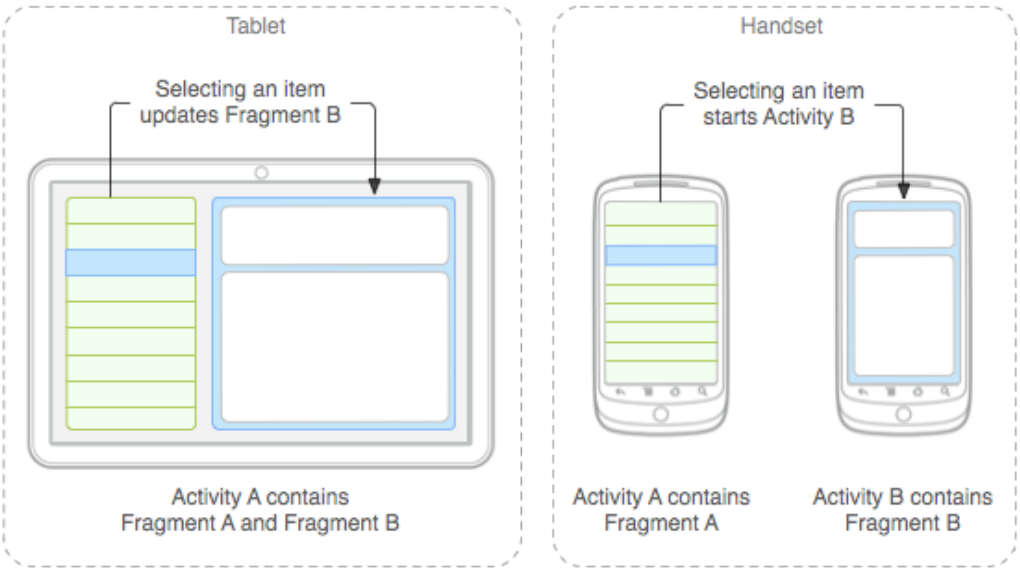
É possibile fare in modo che sia l'applicazione stessa a riconoscere su quale dispositivo è in esecuzione e scegliere il comportamento di visualizzazione più adatto al singolo caso.

L'indipendenza dei fragment dalle activity che li "contengono" permette di utilizzare uno stesso fragment in diverse activity, permettendo il riutilizzo

di una singola funzionalità semplicemente creando una nuova istanza di un fragment in una activity qualsiasi che ne necessiti.

Questa elasticità è possibile unicamente grazie a questo potente strumento, si è scelto quindi di utilizzarlo in modo da impostare una progettazione che preveda già l'evoluzione ad applicazione ottimizzata per tablet.

Figura 3.4: Esempio che rappresenta la possibilità di visualizzare due fragment in una stessa activity quando si utilizza uno schermo sufficientemente ampio, o in alternativa la visualizzazione di un singolo fragment alla volta, nel caso di spazi di visualizzazione più ridotti [23]



Comunicazione fra fragments I fragment sono, come già detto, elementi indipendenti fra loro e dall'activity che li contiene, ma nonostante questo hanno la possibilità di comunicare fra loro e con la stessa activity.

Questo processo viene implementato tramite l'utilizzo di *callbacks*.

Se si vuole implementare la possibilità per un fragment di comunicare un certo dato all'activity, il procedimento è il seguente:

- L'activity ospitante deve implementare l'interfaccia di callback relativa all'operazione che si vuole eseguire.

- Il fragment richiama l'istanza dell'activity che lo ospita tramite il comando `getActivity()` senza ovviamente avere conoscenza di quale activity possa essere.

- Si controlla se l'activity implementa l'interfaccia di callback richiesta.

- In caso positivo si utilizza il riferimento ottenuto per richiamare il metodo di callback, implementato nella classe dell'istanza in esame.

Si è inoltre detto che è possibile fra due fragment comunicare fra loro, il problema è che una singola istanza deve rimanere indipendente dal resto dell'interfaccia grafica. L'unico componente che ha le informazioni riguardo quali fragment sono visibili all'utente in un determinato momento è l'activity, che definisce esattamente questi comportamenti.

Il design pattern corretto quindi richiede che se due fragment, A e B vogliono scambiarsi un messaggio, non debbano interagire con l'istanza dell'altro, bensì con l'activity che li ospita (è infatti certo che se un fragment è in esecuzione, vi è una activity che lo contiene, ma non è certo se un secondo fragment sia visualizzato e istanziato nello stesso istante, o meno).

Dunque, utilizzando l'esempio utilizzato in precedenza (vedi figura 3.4), il fragment che contiene la lista non è a conoscenza se la vista di dettaglio sia presente nella stessa activity, utilizza quindi il metodo di callback implementato dall'activity, la quale potrà inviare l'informazione al fragment di dettaglio se questo è già istanziato e visibile, o procedere a una nuova istanziazione.

Risulta quindi nuovamente evidente quali possibilità di flessibilità offre un componente come il fragment, e quanto sia importante progettare l'applicazione già implementando fin da subito la logica applicativa adatta, che sarebbe molto problematica da modificare in un secondo tempo, portando probabilmente a uno stravolgimento profondo di tutta la progettazione esistente fino a quel momento.

3.2.4 Retrocompatibilità

A causa della frammentazione delle versioni del sistema operativo Android presenti attualmente sul mercato (vedi: 1.2.3) risulta quantomai importante rendere l'applicazione compatibile con versioni in più possibile datate, così da avere un bacino di utenza il più ampio possibile.

Ottenere questo risultato rinunciando all'utilizzo delle API più aggiornate non è sicuramente una soluzione ideale, in quanto con l'evoluzione del sistema operativo sono stati forniti da Google strumenti sempre più avanzati e ottimizzati e rinunciarvi può portare solo a creare un prodotto di scarsa qualità.

É inoltre importante ricordare che uno degli aspetti fondamentali che caratterizzano una applicazione mobile di qualità è l'interfaccia grafica e la capacità di quest'ultima di essere intuitiva e familiare per l'utente fin dal primo utilizzo. Questo obiettivo si raggiunge con l'utilizzo di template e design pattern, basati, ovviamente, sulle API e sui componenti di ultima generazione.

Seppure a prima vista questi due obiettivi possano sembrare opposti e incompatibili, la soluzione è fornita dalla stessa Google che mette a disposizione degli sviluppatori una serie di librerie di supporto che implementano tutti i componenti più recenti mantenendo la compatibilità con le API più datate.

Si utilizzano quindi componenti della libreria che implementano in tutto e per tutto le funzionalità delle controparti originali senza la necessità delle relative API più recenti.

Per questo progetto si è fatto un uso massiccio della *v4 Support Library* per quasi tutti i componenti utilizzati, dai *fragment* ai *loaders* fino ai *content providers*.

É stato inoltre utilizzata la *v7 Library* per implementare la *ActionBar* al fine di creare una interfaccia grafica quanto più conforme ai template più recenti.

3.2.5 Shared preferences

La classe Android *SharedPreferences* mette a disposizione un framework che permette utilizzare e salvare in modo persistente dati nel formato chiave-valore, indipendentemente dal ciclo di vita dell'applicazione che, dunque, manterrà i dati anche in caso di chiusura forzata o spegnimento del dispositivo.

Il funzionamento è molto semplice e intuitivo: tramite l'oggetto *Editor* è possibile inserire una coppia chiave-valore che, se già esistente, sovrascriverà il valore della precedente.

Il processo di modifica dei dati è gestito con *metodo transazionale*, i dati vengono salvati solo quando viene eseguito il *commit()* delle modifiche.

Per recuperare le informazioni salvate si utilizzano metodi di *get* specifici per il tipo di valore restituito (`getString()`, `getInt()`, ecc.).

La memorizzazione avviene tramite file .xml salvato in una area dati privata dell'applicazione, non accessibile esternamente.

Vi è distinzione fra *Preferences* e *SharedPreferences*: nel primo caso il file e i valori contenuti al suo interno sono accessibili solo dall'activity proprietaria e non dalle altre pur facenti parte della stessa applicazione.

Nel caso delle *SharedPreferences* invece la fonte dati è comune a tutte le activities.

3.3 Definizione database locale e database remoto Esercizi2013

3.3.1 Database Esercizi2013

Per la parte di sviluppo riguardante il *CMS Esercizi2013* è stato fornito dal committente un file in formato .SQL contenente i comandi DDL completi per la definizione dello schema. Il progetto prevedeva quindi di sviluppare l'applicativo Android in modo che fosse in grado di comunicare con il server del database e fornire all'utente una visualizzazione dei dati in esso contenuti.

Non vi era quindi la necessità di valutare eventuali modifiche allo schema, ma unicamente di conoscerne il funzionamento per poter definire le query necessarie al servizio PHP per caricare i dati richiesti e fornirli all'applicativo su richiesta e per implementare alcune funzionalità presenti sul portale web.

Si è quindi definito lo schema ER della base di dati basandosi sul codice DDL fornito.

Figura 3.5: É qui rappresentato uno schema ideale della base di dati, divergente dall'effettiva implementazione, ma utile per avere una visione di insieme del dominio.

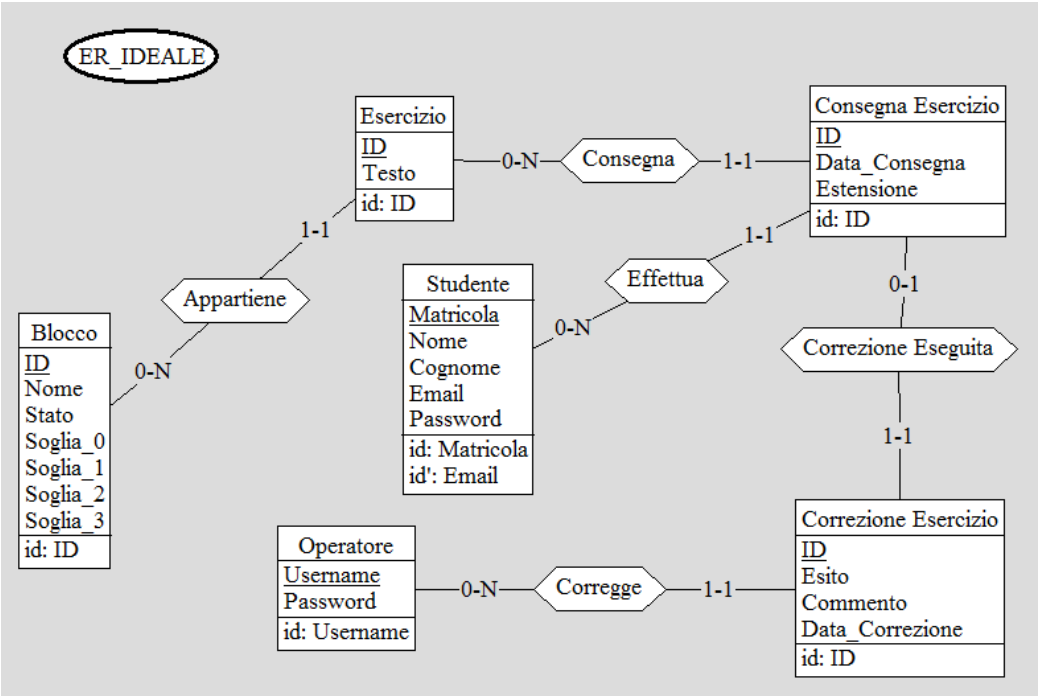


Figura 3.6: Una ulteriore raffinazione dello schema concettuale rappresenta una soluzione ideale per la definizione dello schema.

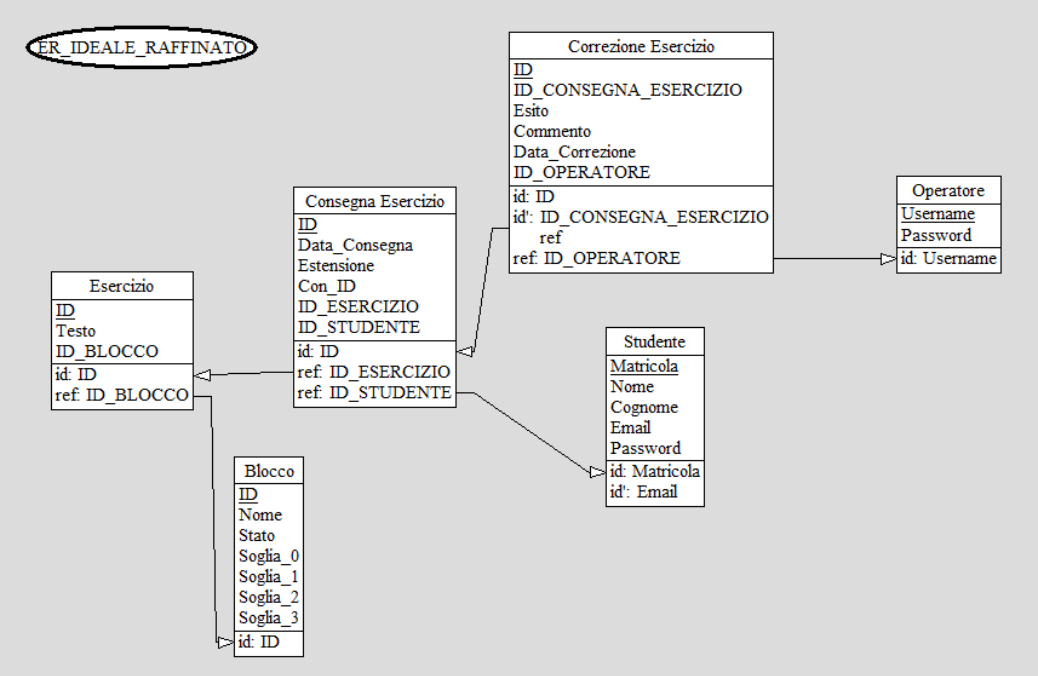
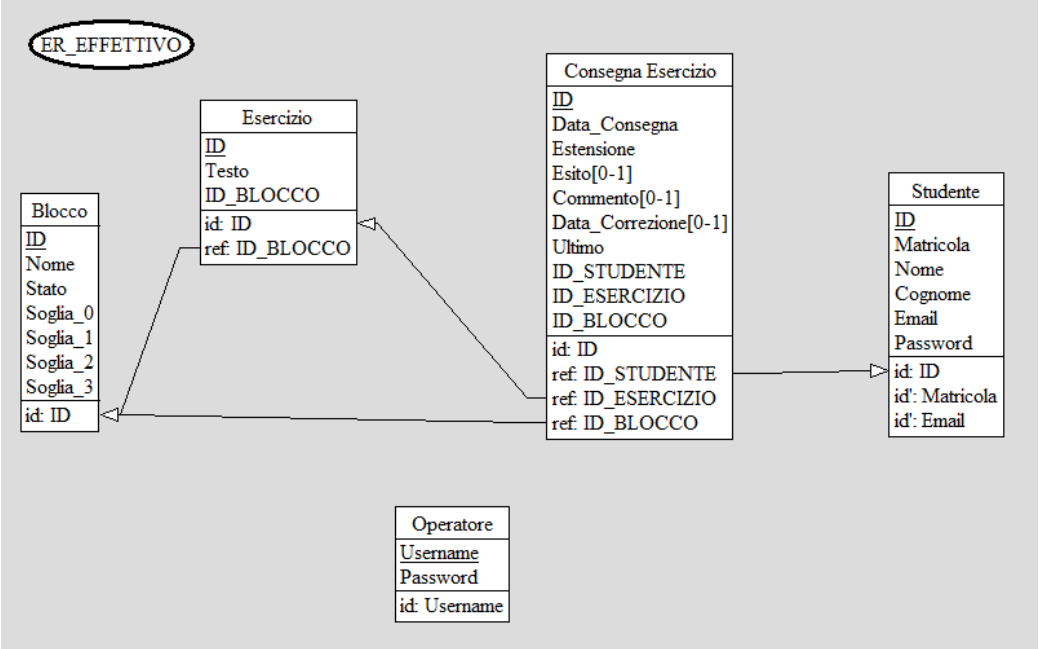


Figura 3.7: In seguito viene definito invece quello che è lo schema concettuale raffinato relativo all'effettiva implementazione della base di dati



Dal modello concettuale reale (vedi figura 3.7) si definisce quindi lo schema logico relativo all'implementazione esistente

blocco: (ID_BLOCCO, NOME, STATO, SOGLIA_0, SOGLIA_1, SOGLIA_2, SOGLIA_3)

esercizio: (ID_ESERCIZIO, ID_BLOCCO, TESTO)
FK: ID_BLOCCO **REFERENCES** blocco

consegna_esercizio: (ID_CONSEGNA_ESERCIZIO, ID_STUDENTE, ID_ESERCIZIO, ESITO, DATA_CONSEGNA, ESTENSIONE, COMMENTO, ID_BLOCCO, ULTIMO, DATA_CORREZIONE)
FK: ID_BLOCCO **REFERENCES** blocco
FK: ID_STUDENTE **REFERENCES** studente
FK: ID_ESERCIZIO **REFERENCES** esercizio

studente: (ID_STUDENTE, NOME, COGNOME, MATRICOLA, E_MAIL, PASSWORD)

operatore: (USERNAME, PASSWORD)

Le differenze dello schema reale rispetto a quello ideale risultano poche ma di sostanza:

- quattro tabelle su cinque risultano in *terza forma normale*, mentre la tabella *consegna_esercizio* risulta in *prima forma normale* a causa della ridondanza della chiave importata *id_blocco* che risulta funzionalmente dipendente dal campo *id_esercizio*. La ridondanza risulta giustificata dal risparmio di una operazione di *join* nel momento in cui si desidera visualizzare le correzioni di una singola settimana, operazione che viene molto utilizzata nell'interfaccia web del CMS. Il risparmio dell'operazione di *join* risulta significativa in quanto la tabella *consegna_esercizio* è quella con maggior cardinalità e il prezzo pagato in termini di occupazione di memoria è minimo visto il tipo di dato del campo ridondante. Problematiche quali incoerenza dei dati non risultano preoccupanti in quanto non sono previste modifiche dei dati che possano modificare la dipendenza funzionale fra un determinato *id_esercizio* e il relativo *id_blocco*.

- Non è tenuta traccia in alcun modo di quale utente di tipo "operatore" effettui una modifica a una tupla di *consegna_esercizio*.

- Viene introdotto il campo *ultimo* per ottimizzare il processo di selezione della consegna più recente per un determinato esercizio da parte di un determinato studente. Per l'esecuzione di tale operazione era sufficiente il campo *data_consegna*, ma è stato previsto questo ulteriore dato per ragioni di ottimizzazioni derivanti dall'implementazione dell'interfaccia web. Il dato più significativo di questa modifica è l'introduzione di un vincolo che richiede una modifica dei dati esistenti in seguito a un nuovo inserimento. Nel momento in cui viene effettuata, da uno stesso studente, una nuova consegna successiva alla prima, di un determinato esercizio, risulta necessario modificare da *1* a *0* il valore del campo *ultimo* di quella che era, fino a quel momento, la consegna più recente. In termini di prestazioni questo vincolo risulta significativo in quanto richiede, per ogni singolo inserimento, l'esecuzione di una query di update, in quanto senza quest'ultima il sistema non ha possibilità di

sapere se l'inserimento che sta per essere effettuato è il primo per una precisa coppia *studente-esercizio* o l'ennesimo.

- Nella tabella *studente* viene utilizzato un surrogato di tipo integer per la chiave primaria della tabella, quando sono presenti ben 2 campi (*matricola* e *e_mail*) che risultano chiavi candidate visto lo specifico dominio dei dati.

Risulta inoltre utile conoscere la cardinalità delle tabelle, basandosi sul dump fornito dal committente.

Tabella 3.6: Cardinalità Tabelle DB_Esercizi2013

Tabella	**Cardinalità**
Blocco	11
Consegna_Esercizio	8502
Esercizio	51
Operatore	3
Studente	239

3.3.2 Database Programmazione

I dettagli del database presente sul server del CMS Programmazione non sono stati forniti dal committente. Ci si è trovati quindi nella situazione di dover scaricare dati in formato JSON da una "black-box" di cui non si conoscevano i dettagli e doverli memorizzare in un database locale all'applicazione Android, che andava definito interamente da zero.

Si è quindi deciso di partire analizzando i dati ottenuti tramite il servizio remoto. I dati con cui si è avuto a che fare consistevano in diversi oggetti di tipo *JSONArray* che a loro volta contenevano *JSONObject* che rappresentavano l'informazione di una tupla del database.

I dati suggerivano quindi una divisione in tabelle (con schemi molto simili fra loro) differenziate per tipologia di informazione.

Figura 3.8: Una prima analisi concettuale è utile per tracciare un quadro generale dei dati con cui si è avuto a che fare. Risultano evidenti le similitudini degli schemi di tutte le tabelle.

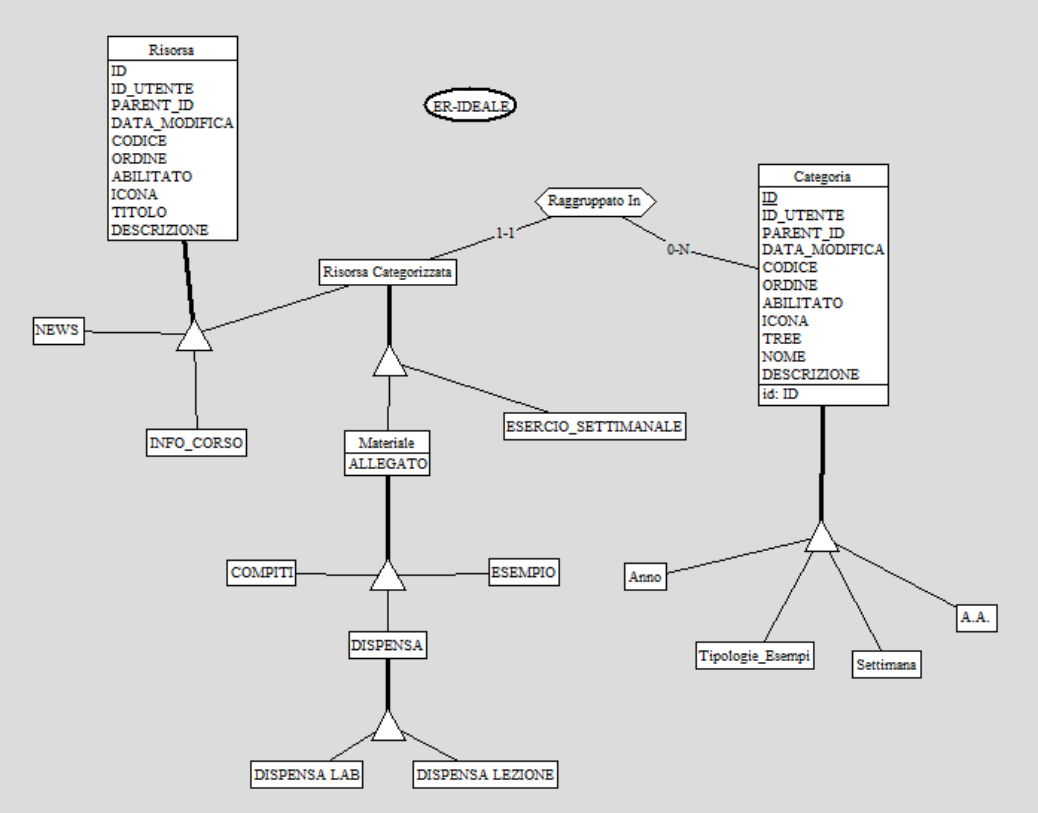
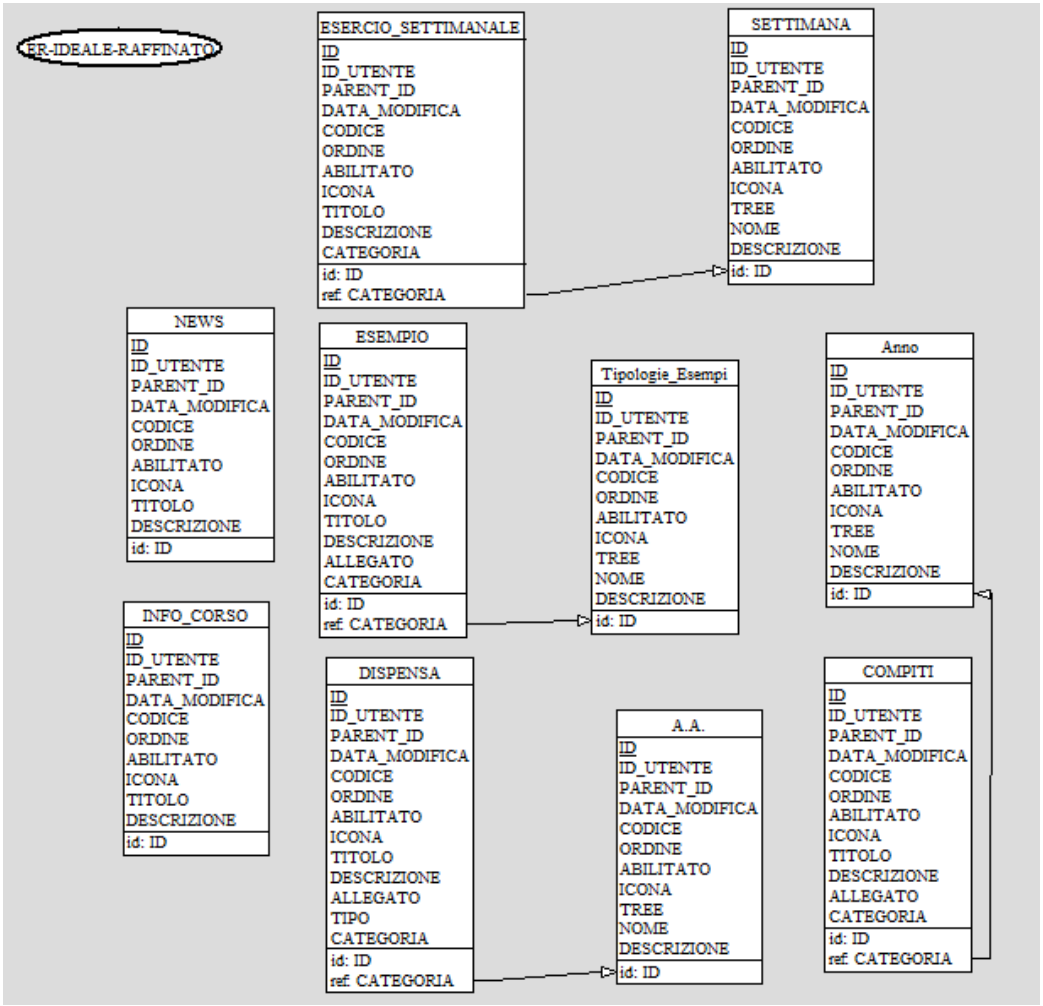


Figura 3.9: Apportando ulteriori raffinazioni allo schema si eliminano le classi astratte e si esegue un *collasso verso il basso* per evidenziare i vincoli che specificano che a seconda del tipo di materiale vi è una tipologia particolare di categoria ad esso collegato. Rispetto allo schema concettuale iniziale vengono separati tutte le classi in tabelle a sé stanti, ad eccezione delle classi relative alle dispense essendo queste ultime concettualmente molto simili e essendo associate alla medesima tipologia di categoria. Per quest'ultimo caso si è quindi ritenuto opportuno aggiungere un attributo *Tipo* che rappresentasse la distinzione fra i due concetti del dominio.



Implementazione finale Nonostante le indicazioni emerse durante la fase di analisi, ci si è trovati a dover gestire una problematica ulteriore che ha reso inopportuna una implementazione di quel tipo.

L'obiettivo di questa fase del progetto consiste nella creazione di un database locale all'applicazione che necessita di *rimanere sincronizzato* con il database remoto, è quindi stato necessario limitare all'estremo i cambiamenti dei dati nel passaggio da un database all'altro perché la rielaborazione dei dati scaricati da remoto per adattarli a uno schema diverso dall'originale può risultare molto onerosa per hardware modesti come quelli che caratterizzano i dispositivi mobili a cui è destinata l'applicazione in esame

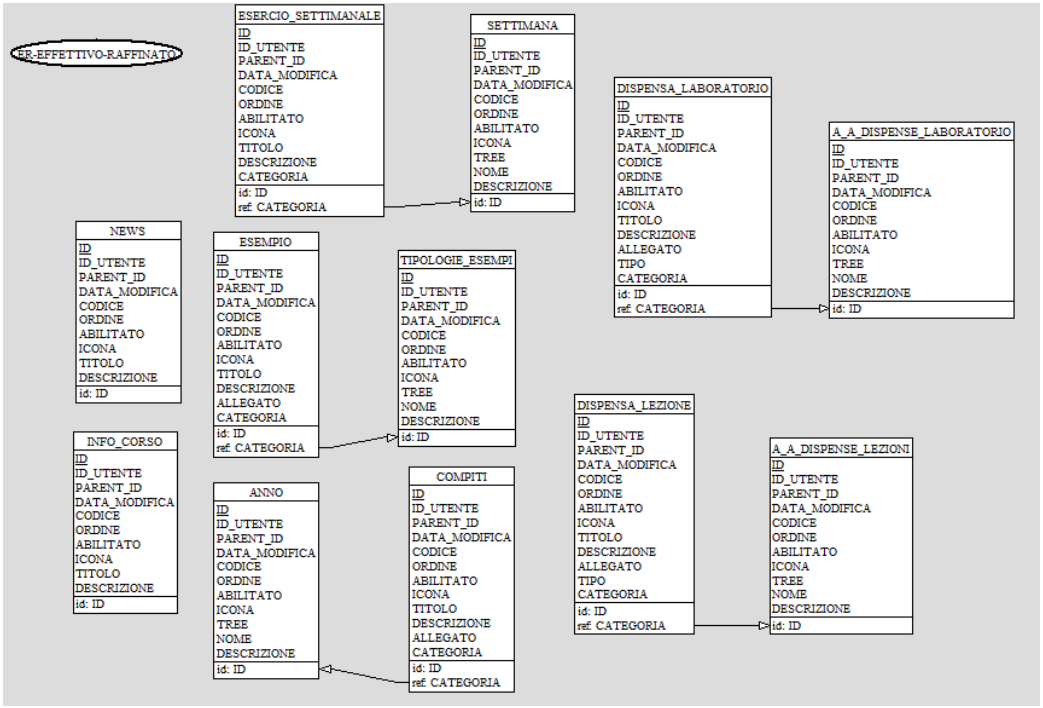
A tale scopo, per citare un esempio, si è preferito mantenere gli stessi valori di chiave primaria anche nel database locale, senza aggiungere surrogati, questo però ha reso l'implementazione estremamente dipendente dalle modalità di memorizzazione dei dati all'origine.

Una delle problematiche principali, che hanno determinato un cambiamento dello schema rispetto a quello definito nella fase di analisi, è risultata essere la memorizzazione delle dispense delle lezioni di teoria e di laboratorio. Come già illustrato le due entità sono estremamente simili, e condividono la stessa tipologia di classificazione.

Analizzando i dati si è però notato che le tuple degli anni accademici i cui riferimenti si trovano nelle istanze delle tabelle delle dispense, non sono comuni. Ovvero, la tupla "A.A. 2012-2013" la cui chiave è importata nelle istanze della tabella *Dispense_Lezione* è diversa e indipendente dalla tupla che reca lo stesso contenuto e a cui si riferiscono le istanze della tabella *Dispense_Laboratorio*.

Questa e altre problematiche derivanti dall'assenza di informazioni sulla base di dati di origine, unite all'esigenza di limitare al massimo il sovraccarico di lavoro per l'applicazione mobile, ha portato alla definizione di un ulteriore modello concettuale (vedi figura 3.10), poi tradotto in logico e infine implementato.

Figura 3.10: Schema concettuale alla base dell'effettiva implementazione della base di dati locale



Si è quindi derivato il sequenze schema logico:

categoria_dispense_lezione: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TREE, NOME, DESCRIZIONE)

categoria_dispense_laboratorio: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TREE, NOME, DESCRIZIONE)

categoria_compiti: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TREE, NOME, DESCRIZIONE)

categoria_esempi: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TREE, NOME, DESCRIZIONE)

categoria_esercizio_settimanale: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TREE, NOME, DESCRIZIONE)

dispensa_lezione: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TITOLO, DESCRIZIONE, ALLEGATO, CATEGORIA)
FK: CATEGORIA **REFERENCES** categoria_dispense_lezione

dispensa_laboratorio: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TITOLO, DESCRIZIONE, ALLEGATO, CATEGORIA)
FK: CATEGORIA **REFERENCES** categoria_dispense_laboratorio

esempio: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TITOLO, DESCRIZIONE, ALLEGATO, CATEGORIA)
FK: CATEGORIA **REFERENCES** categoria_esempi

compito: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TITOLO, DESCRIZIONE, ALLEGATO, CATEGORIA)
FK: CATEGORIA **REFERENCES** categoria_compiti

esercizio_settimanale: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TITOLO, DESCRIZIONE, CATEGORIA)
FK: CATEGORIA **REFERENCES** categoria_esercizio_settimanale

news: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TITOLO, DESCRIZIONE)

info_corso: (ID, ID_UTENTE, PARENT_ID, DATA_MODIFICA, CODICE, ORDINE, ABILITATO, ICONA, TITOLO, DESCRIZIONE)

Una soluzione alternativa Durante la fase di analisi si è trovata una ulteriore soluzione per l'implementazione, illustrata brevemente in quanto non utilizzata.

A causa dello schema quasi identico per tutte le tabelle, sarebbe stata facilmente impresentabile una soluzione che vedeva il collasso di più tabelle con dati "concettualmente" diversi fra loro, utilizzando un campo *tipo* su cui effettuare operazioni di selezione basandosi sul nome della tabella presente nei dati in formato JSON. Dovendo semplicemente copiare e mantenere sincronizzati i dati provenienti da una fonte esterna, si sarebbe potuta dare per scontata la consistenza dei dati e limitarsi a conservarli in locale con la possibilità di selezionarli a seconda della richiesta.

Questa soluzione è attualmente resa praticabile dal fatto che tutti gli identificatori presenti nelle diverse tabelle risultano univoci fra loro, evitando quindi conflitti di chiave primaria anche nel caso estremo in cui tutti i dati venissero raggruppati in una unica tabella. É però evidente quanto questa situazione sia sensibile ai cambiamenti della base di dati sorgente, rischiando

quindi di diventare un problema in caso di modifiche al livello intensionale di quest'ultimo.

Una ulteriore, evidente, motivazione per non attuare questa strategia ma quella illustrata in precedenza è che si preferisce suddivide il carico di lavoro mantenendo divisi i dati in base al loro effettivo valore informativo. Questo poterà ad utilizzare tabelle con cardinalità minori, portando un netto vantaggio sulle prestazioni (particolarmente importanti trattandosi di software per il mobile).

Capitolo 4

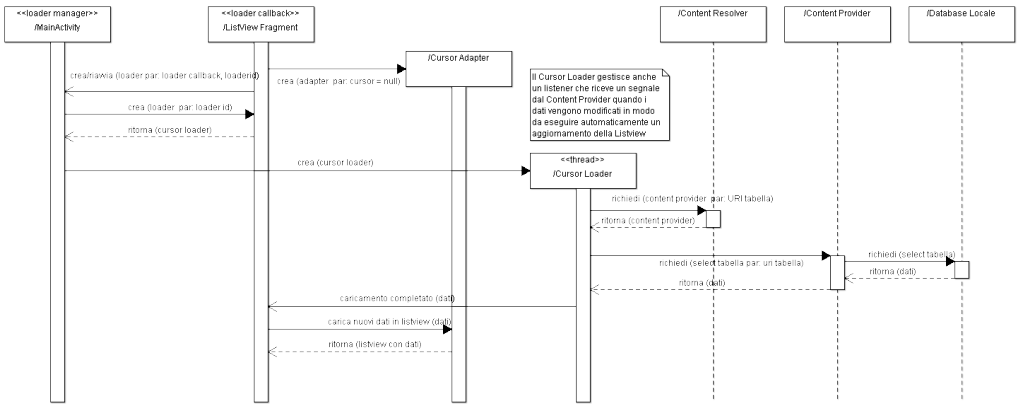
Implementazione del progetto

4.1 Descrizione dell'implementazione dell'applicazione

In questo capitolo viene approfondita nei dettagli l'implementazione dell'applicazione, descrivendo le parti principali del codice java prodotto e descrivendo il servizio remoto sviluppato in PHP utilizzato per la comunicazione con il server del CMS Esercizi. Vengono inoltre descritte le funzionalità implementate e forniti alcuni accenni su elementi significativi dell'interfaccia grafica

4.1.1 Content provider

Figura 4.1: Diagramma di sequenza che illustra l'interazione fra Content Provider e Cursor Loader



Utilizzo del Content Provider all'interno dell'Applicazione Nel diagramma di sequenza in figura 4.1 viene mostrato in modo semplificato il funzionamento e l'interazione dei componenti quando viene utilizzato un Content Provider per la gestione dei dati e un loader per il loro caricamento.

I particolari più importanti da evidenziare sono:

- Il caricamento dei dati è gestito dal loader, un thread separato che quindi non inficia il funzionamento dell'interfaccia grafica e del resto dell'applicazione.

- In caso di distruzione e ricreazione del fragment e della view, i dati rimangono salvati all'interno del cursore del loader, e vengono successivamente richiamati senza dover eseguire una seconda query sul database.

- L'adapter viene inizialmente creato senza una fonte dati, nel momento in cui gli verrà assegnato un cursore eseguirà il caricamento dei dati sulla ListView.

- Il sistema gestisce in automatico un *observer* sui dati, al verificarsi del loro cambiamento sul database viene dato segnale al loader di ripetere il caricamento dei dati e di conseguenza l'aggiornamento della listview.

Implementazione Per implementare il Content Provider relativo ai dati del database locale, si è creata una classe figlia della classe di sistema *Content Provider*.

Al suo interno devono essere presenti i seguenti elementi:

La fonte dei dati In questo caso viene istanziato un oggetto database di tipo SQLiteDatabase, dichiarato *private* e con pattern Singleton.

Authority del provider Identifica univocamente il provider.

```
1 private static final String AUTHORITY = "it.unibo.tesi.programmazione.main.contentProvider";
```

Paths delle tabelle Sono costanti utilizzate all'interno del Content Provider, sono create utilizzando i dati delle classi che descrivono le tabelle.

```
1 private static final String TAB_NEWS_PATH = DBTGnews.TABLE_NAME;
```

Uri delle tabelle Vengono definiti gli URI di ogni tabella tramite composizione di elementi definiti precedentemente insieme a stringhe standard specifiche della sintassi degli URI (content://).

Metodo statico per ottenere l'Uri della tabella partendo dal suo nome

UriMatcher Metodo che si occupa di controllare se un Uri è gestito o meno da questo Content Provider.

ID univoci per ogni Uri Ad ogni Uri è associato un codice univoco che a sua volta viene dichiarato staticamente nella classe.

Aggiunta dei parametri del Content Provider all'UriMatcher Vengono aggiunti gli elementi precedentemente creati, *authority*, *path* e *codice univoco* all'UriMatcher che in questo modo potrà riconoscerli.

Definizione del tipo di dato esposto dal Content Provider Il tipo dovrà essere nella forma *tipo/sottotipo* ad esempio "image/jpg". In questo caso vengono restituiti dei cursori a tabelle.

Metodo per convertire il codice restituito dall'UriMatcher nel relativo tipo di dato fornito

Override del costruttore della classe In fase di creazione del Content Provider si ottiene un'istanza del database tramite il relativo helper.

Definizione delle operazioni C.R.U.D. Il Content Provider fornisce una interfaccia delle operazioni C.R.U.D. standard, al suo interno è quindi necessario definire quali operazioni verranno effettivamente eseguite. Questo dipende da come sono memorizzati i dati nel layer sottostante al Content Provider. In questo caso abbiamo un database e le relative operazioni SQL standard.

Versione dei Dati Un dettaglio specifico del dominio applicativo in esame è il concetto di *versione dei dati.*: è infatti necessario fornire al servizio remoto la versione attuale dei dati presenti nell'applicazione per ottenere in risposta solo le modifiche effettuate successivamente a quella specifica versione. Per la gestione di questo dato è stata implementata una soluzione non ottimale per massimizzare la compatibilità dell'applicazione con il maggior numero di versioni del sitema operativo, si vuole quindi illustrare ciò che è stato implementato e quale sarebbe la soluzione migliore al problema.

Soluzione ottimale Il concetto di versione dei dati è una informazione prettamente inerente ai dati stessi, sarebbe quindi corretto che la sua

gestione fosse demandata al Content Provider, come per tutto il resto delle operazioni. Per ottenere questo risultato, è possibile aggiungere dei metodi personalizzati all'interno del Content Provider. Dal momento che l'applicazione non accede mai, direttamente, al Content Provider ma utilizza sempre un Content Resolver e un Uri per farlo, dalle API 11 di Android è stata implementata una nuova funzione:

```
1 public final Bundle call (Uri uri, String method, String arg, Bundle extras)
```

Il suo scopo è appunto far eseguire un metodo aggiuntivo al Content Provider indirizzato dall'Uri dato input. Nel metodo richiamato verrà poi gestito il salvataggio del dato o all'interno di una tabella apposita o tramite altre metodi di salvataggio forniti dal sistema o il recupero di un valore precedentemente salvato, quando richiesto.

Il motivo per cui si è deciso di non realizzare questa soluzione è la volontà di mantenere il livello di API minimo a 9, ovvero alla versione di Android 2.3.3. in quanto ancora largamente diffusa. (vedi: 1.2.3)

Soluzione implementata Per ovviare alla problematica si è scelto di distaccare l'informazione sulla versione dei dati dagli stessi dati. Durante le operazioni di aggiornamento del database, il valore fornito dal server viene salvato nei dati privati dell'applicazione tramite l'utilizzo delle SharedPreferences (vedi: 3.2.5) e e richiamato dalle stesse per poter fornire il parametro alla richiesta HTTP-POST da effettuare verso il server remoto.

Tramite opportuna implementazione del gestore delle SharedPreferences si è fatto in modo che il dato non venga mai cancellato, quindi l'unico modo di perderne il valore è disinstallare l'applicazione, ma in tal caso verrebbe anche eliminato il database facendo perdere dunque al dato ogni significato.

Con questa soluzione si garantisce quindi che il dato sia sempre disponibile e abbia un ciclo di vita identico a quello del database, seppur demandando la gestione dei dati a due diversi moduli del sistema.

4.1.2 Database SQLite locale

DbHelper

La classe fondamentale per gestire un database SQLite su Android è SQLiteOpenHelper, questo oggetto si occupa di creare, aggiornare e gestire un database. Ogni qualvolta una parte dell'applicativo (in questo caso unicamente il content provider) necessiterà di effettuare operazioni sul database, dovrà richiedere l'istanza di quest'ultimo all'oggetto *helper*, che potrà restituirlo con permessi di scrittura o di sola lettura.

La classe SQLiteOpenHelper mantiene al suo interno tutte le informazioni del datbase, come nome del file e versione dello schema.

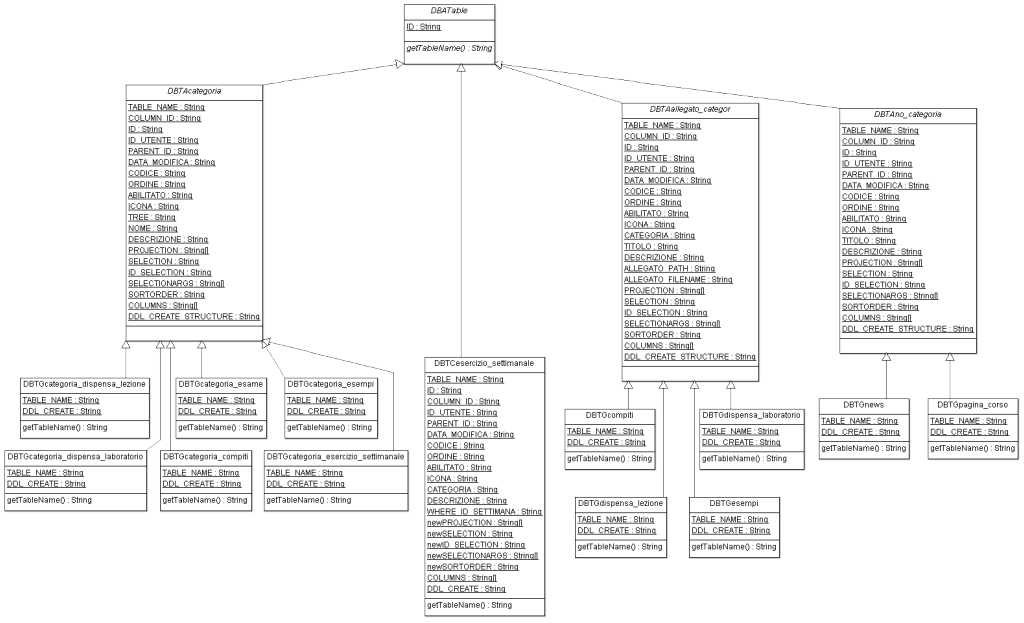
In questo progetto si è scelto di implementare l'helper in modalità singleton per ottimizzare le performances.

Nel costruttore dell'helper viene eseguita l'istanziazione del database, l'eventuale pre-esistenza del file viene gestita automaticamente dalla classe madre. Solo la prima volta che viene creato il database viene eseguito il metodo `onCreate()` che si occuperà quindi di eseguire le query di DDL per la definizione delle tabelle.

Esecuzione del codice DDL delle tabelle Per motivi di divisione logica delle informazioni, ogni classe che definisce la singola tabella mantiene al suo interno il codice DDL per la definizione del proprio schema, in modo che l'helper debba semplicemente eseguire il codice DDL indipendentemente dal suo contenuto.

Le Classi delle tabelle Android fornisce l'interfaccia BaseColumns per la creazione di classi che contengano le informazioni riguardanti una tabella di un database. Sfruttando le proprietà di ereditarietà si è creata una struttura ad albero di classi raggruppando nelle classi padre le caratteristiche comuni a più tabelle.

Figura 4.2: Diagramma delle classi che mostra lo schema di ereditarietà delle tabelle del database interno all'applicazione



Il primo livello, che consiste nella sola classe DBATable, è una classe astratta che ha lo scopo di contenere i metodi e le proprietà comuni a tutte le tabelle.

Il secondo livello formato dalle tre classi astratte con prefisso DBTA (DataBase Table Abstract) contengono

- Le costanti di tutti i nomi dei campi della tabella
- I vettori che raggruppano i nomi dei campi in base all'utilizzo
- La parte comune del DDL di creazione della tabella

Infine nelle classi foglie vengono dichiarati il nome della tabella, viene creata la stringa completa del DDL di creazione, e vengono implementati i metodi astratti dichiarati nelle classi madri.

Vi sono infine una classe che eredita direttamente dalla classe DBATable in quanto particolare e con schema diverso dalle altre: *DBTCeserci-*

zio_settimanale (*Data Base Table Custom*), avendo campi unici per la specifica tabella ha tutte le dichiarazioni di costanti e metodi collassate in un'unica classe che deriva dalla classe madre.

RowWrapper La classe RowWrapper è stata creata come contenitore per le informazioni riguardo una tupla di una tabella, al suo interno contiene tutte le costanti e i metodi per gestire una qualsiasi tupla di una qualsiasi delle tabelle gestite dall'applicazione, in modo che si possa unificare il metodo di gestione in un insieme misto di tuple contenenti le informazioni di tipo diverso.

4.1.3 CMS Esercizi2013

Una parte delle funzionalità offerte dall'applicativo sono ottenute tramite l'interfacciamento al CMS Esercizi2013. Per lo sviluppo di questo componente dell'applicazione è stato fornito una copia dei sorgenti PHP che compongono il CMS, sono stati quindi installati in una macchina di test configurata con webserver Apache e PhpMyAdmin come tool di gestione del database locale. Contrariamente alla parte relativa al CMS Programmazione in questo caso non ci si è quindi trovati a dover interagire con una blackbox ma si è avuto accesso ai dettagli dell'implementazione dell'applicativo server side esistente, compreso il database, elemento estremamente importante per poter effettuare scelte implementati efficienti e performanti. Vi era comunque il vincolo di non poter intaccare l'applicativo esistente, si è quindi optato per la creazione di un servizio PHP aggiuntivo che si affiancasse a quelli esistenti, talvolta implementando funzionalità simili.

Login

L'operazione di login è univoca sia per gli studenti che devono accedere ai dati delle proprie consegne in sola lettura, sia per gli operatori che invece hanno accesso a tutti i dati con la possibilità di modificarli per inserire nuove

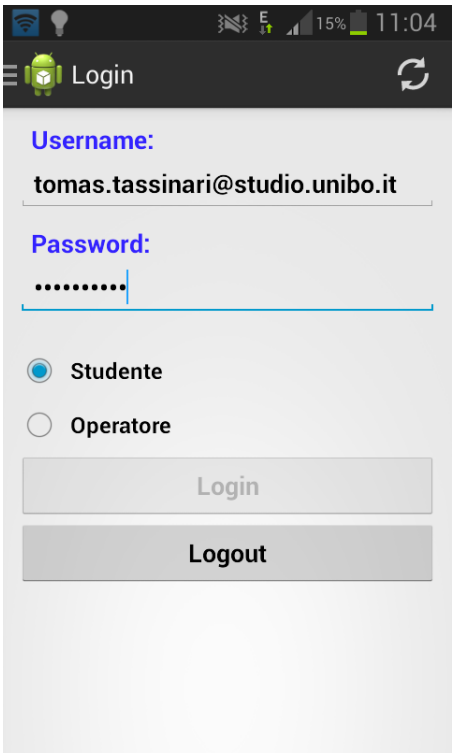


Figura 4.3: La schermata di login dell'applicazione, in questo caso mostra un utente già autenticato correttamente come studente

correzioni. In seguito al processo di login, viene identificata la tipologia di utente, e di conseguenza vengono fornite le diverse funzionalità.

Il fragment presenta una semplice interfaccia che permette di inserire le credenziali e selezionare la tipologia di utenza, in seguito alla pressione del pulsante per l'invio dei dati, viene effettuata una chiamata asincrona al servizio PHP allegando come parametri di tipo POST i dati inseriti dall'utente unitamente a un parametro di tipo GET che differenzia le casistiche di login di uno studente da quello di un operatore.

Il valore di ritorno restituito dal server consiste in una stringa JSON che contiene la conferma di avvenuta registrazione e un token che dovrà essere utilizzato per mantenere la sessione aperta e non dover ripetere il login.

Il task asincrono si occupa di decodificare la stringa e registrare i dati dell'utente che ha effettuato il login salvandoli in un file locale che li conserverà anche dopo la chiusura dell'applicazione, oltre a renderli disponibili e consultabili da altri processi in modo totalmente indipendente.

Reperimento dati

La gestione dei dati residenti nel CMS Esercizi2013 avviene senza il salvataggio degli stessi in locale sul dispositivo. Avendo la possibilità di implementare il servizio server side è infatti stato possibile creare una serie di query specifiche per le funzionalità che si intendeva offrire all'utente, permettendo quindi di ottimizzare le risorse disponibili e minimizzare il carico di lavoro. Ogni fragment effettua una chiamata al server richiedendo una specifica tipologia di dati, delineati dalla vista in cui si trova l'utente e dai suoi dati personali (memorizzati lato server sfruttando il meccanismo dei token di sessione), riceve i dati codificati in una stringa JSON, per poi venire caricati in modo asincrono sulla struttura di visualizzazione scelta, come una listview (processo schematizzato nell'immagine 3.2 a pagina 58).

Oltre alla possibilità di implementare i servizi lato server, ci sono stati altri fondamentali motivi per cui si è scelto di reperire i dati direttamente online senza memorizzazione locale:

Ogni utente ha accesso a una ristretta porzione dei dati risulta evidente che ogni utente, che sia di tipo studente o operatore, ha accesso a una ristretta parte dei dati rispetto all'insieme, nel caso degli studenti perché hanno accesso unicamente ai dati delle proprie consegne, mentre gli operatori necessitano unicamente delle consegne non ancora corrette, generalmente in numero inferiore alla quantità totale dei dati presenti sul server.

I dati cambiano velocemente altra caratteristica della tipologia di dati trattati è la loro volatilità, le consegne degli esercizi vengono modificate spesso, rendendo più efficiente una lettura in real time delle istanze dei dati.

Descrizione delle componenti dell'interfaccia

Tutti i componenti che formano il modulo di interfacciamento al CMS in esame seguono il pattern *Fragment - Adapter - Loader*. Per ottimizzare l'or-

ganizzazione del codice, sono state create 3 classi astratte che implementano i comportamenti comuni a tutti i diversi componenti dello stesso tipo, ogni componente è quindi completato nella specifica classe che si differenzia dalle altre dello stesso tipo per lievi ma importanti differenze. Si è ritenuto che implementare una singola classe per ogni tipologia di componente con istanziazione parametrizzata avrebbe portato ad avere classi molto complesse e difficili da rimaneggiare al bisogno, si è quindi scelta la strada di aumentare il numero di classi (seppur ottimizzando il riutilizzo del codice) per ottenere un risultato più snello e di facile comprensione.

Segue una descrizione dei tre componenti fondamentali utilizzando una singola classe di ogni tipologia:

Fragment La classe astratta definisce l'implementazione dell'interfaccia di callback e delle operazioni comuni, ad esempio quelle che seguono il termine di caricamento del loader.

```
@Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);

        try
        {
            mCallback = (MainFragListener) activity;
        }
        catch (ClassCastException e)
        {
            throw new ClassCastException(activity.toString() + " must implement Callback");
        }
    }

  @Override
  public void onLoadFinished(Loader<List<RowWrapper>> loader, List<RowWrapper> data)
  {
    // Trasferisce i dati letti nell'adapter
    adapter.setData(data);

    if(isResumed()){
      setListShown(true);
    } else {
      setListShownNoAnimation(true);
```

```
26      }
27    }
28
29    @Override
30    public void onLoaderReset(Loader<List<RowWrapper>> loader)
31    {
32      adapter.setData(null);
33    }
```

La classe finale di un fragment si occupa inizialmente di istanziare il proprio loader, identificato tramite ID univoco rispetto agli altri fragment e comune fra fragment dello stesso livello e della stessa sezione del sito).

Viene inoltre istanziato l'adapter, inizialmente vuoto, e mandato in esecuzione il loader che si occuperà di reperire i dati e trasferirli nell'adapter in modo asincrono.

```
1  @Override
2    public void onActivityCreated(Bundle savedInstanceState)
3    {
4      super.onActivityCreated(savedInstanceState);
5
6      // Configura la listview per la visualizzazione in assenza di dati
7      setEmptyText("No Data");
8
9      adapter = new AdaptEserciziDaCorreggere(getActivity());
10     setListAdapter(adapter);
11
12     // Inizialmente la lista e' nascosta
13     setListShown(false);
14     getLoaderManager().initLoader(LOADER_ID, null, this);
15   }
```

Adapter La classe astratta si limita a reperire il LayoutInflater con il comando `inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);` e implementare i setter e i getter di base oltre al metodo utilizzato per trasferire i dati all'interno dell'istanza di un adapter.

```
1    public AbstrAdapter (Context context)
2      {
3        inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
4      }
5
6      public void setData(List<RowWrapper> data)
```

```
7     {
8       this.data = data;
9       notifyDataSetChanged();
10    }
```

La classe finale definisce il processo con cui vengono istanziate le diverse view che compongono la UI dai relativi file .xml, tramite il pattern *ViewHolder*, e come vengono caricati i dati in ognuna di esse, con tutte le particolarità specifiche di ogni diversa UI propria di ogni fragment.

```
1     private static class ViewHolder
2     {
3       public TextView id;
4       public TextView nome;
5     }
6
7     @Override
8     public View getView(int position, View convertView, ViewGroup parent)
9     {
10        ViewHolder holder;
11        /**
12         * Il processo di inflating avviene solo se non vi sono views da riutilizzare
13         */
14        if(convertView == null)
15        {
16          convertView = inflater.inflate(rowLayout, parent, false);
17          holder = new ViewHolder();
18                holder.id = (TextView) convertView.findViewById(R.id.textListDetail1);
19                holder.nome = (TextView) convertView.findViewById(R.id.textListDetail2);
20                convertView.setTag(holder);
21
22        }else
23        {
24                holder = (ViewHolder) convertView.getTag();
25        }
26
27      /**
28       * Si effettua l'assegnamento dei valori alle views
29       */
30      RowWrapper mRow = data.get(position);
31      holder.id.setText(mRow.getValue(CL_ES.C_ID_BLOCCO));
32      holder.nome.setText(mRow.getValue(CL_ES.C_NOME));
33
34      return convertView;
35    }
```

Nell'esempio più semplice fra i diversi adapter, si riescono comunque a evidenziare le diverse fasi del processo svolto dall'adapter:

- Viene utilizzato un ViewHolder statico per mantenere in memoria i riferimenti alle viste, in questo modo non sarà necessario effettuare l'inflating per ogni nuova riga visualizzata.
- Si utilizza la convertView fornita da Android: il sistema in automatico ricicla le viste, mantenendo in memoria quelle attualmente visualizzate sullo schermo e riutilizzando quelle non più visibili per assegnarvi i dati delle righe che verranno visualizzate in seguito.
- Dopo aver ottenuto i riferimenti alle viste tramite la convertView o tramite l'inflating, si procede ad assegnare i valori della riga corrente.

Loader La classe astratta implementa la maggior parte delle operazioni che risultano comuni a tutti i fragment.

Nello specifico, in questa classe viene implementato tutto tranne il processo di caricamento dei dati da remoto, che sarà specifico della classe finale.

I metodi implementati sono quelli standard di gestione dei dati:

- `protected void onStartLoading()` Avvia esplicitamente il processo di caricamento dei dati
- `public void deliverResult(List<RowWrapper> data)` Salva i dati letti nella struttura locale dell'oggetto, che verrà utilizzata dal metodo `public void onLoadFinished (Loader<List<RowWrapper>> loader, List<RowWrapper> data)` del fragment, invocato dal sistema quando il loader ha completato le operazioni di caricamento.

```
1  public void onLoadFinished(Loader<List<RowWrapper>> loader, List<RowWrapper> data
       )
2  {
3          // Trasferisce i dati letti nell'adapter
4          adapter.setData(data);
5
6          if(isResumed()
```

```
7           {
8               setListShown(true);
9           }
10          else
11          {
12              setListShownNoAnimation(true);
13          }
14      }
```

- I restanti metodi si occupano di eliminare dalla memoria i dati quando il loader viene resettato o chiuso

La classe finale si limita a implementare l'effettivo reperimento in remoto delle informazioni: ogni classe di tipo loader contiene al suo interno le specifiche del servizio che viene richiesto al server (questo determina l'unico dato di tipo GET della richiesta http effettuata) Viene eseguito l'Override del metodo `public List<RowWrapper> loadInBackground()` che implementa le operazioni di reperimento dei dati

```
1   @Override
2   public List<RowWrapper> loadInBackground()
3   {
4     List<RowWrapper> data = new ArrayList<RowWrapper>(0);
5
6     session  = new PrefManager(getContext());
7
8     HashMap<String, String> hm = new HashMap<String, String>();
9
10    hm.put(CL_ES.C_sID, session.getsID());    // carica il session ID dalle Shared Preferences di
       sistema
11    hm.put(CL_ES.C_ID_ESERCIZIO, mId_esercizio); // imposta ulteriori parametri necessari alla
       query che sara' eseguita lato server
12
13    // Istanza la classe che si occupa di comunicare con il server e restituire la risposta in
       formato JSON
14    JSONDownloader mDownloader = new JSONDownloader();
15    JSONObject mJSO = mDownloader.getJSOFromUrl(mURL, CMSInterface.getPostArgs(hm));
16    InterfaceParser mParser = new ParserEsercizi2013();
17
18    // Se la richiesta al server e' avvenuta con successo il JSONObject conterra' i dati della
       risposta, se le operazioni di comunicazione con il server
19    // o le operazioni lato server hanno avuto dei problemi, l'oggetto JSON restituito e' null
20    if (mJSO != null)
21    {
```

```
22        try
23        {
24          data = (List<RowWrapper>) mParser.parseData(mJSO);
25        }
26        catch (UnsupportedEncodingException e)
27        {
28          e.printStackTrace();
29        }
30      }
31      return data;
32    }
```

4.1.4 CMS Programmazione

CMSInterface

Questa classe racchiude i metodi specifici per l'interfacciamento con il CMS Programmazione. L'invio di richieste post al CMS per ottenere la stringa JSON richiede specifici parametri e in questa classe vengono gestite queste impostazioni tramite la definizione di metodi che saranno utilizzati durante le operazioni di comunicazione con il portale

Il metodo

```
1 public static List<NameValuePair> getPostArgs(String version)
```

richiede in input i parametri necessari per interrogare il servizio del CMS e li trasforma in una lista di coppie chiave-valore pronta per essere utilizzata nella chiamata di tipo POST al servizio PHP del portale.

Interfacce

Sono state definite 2 interfacce con scopi molto diversi fra loro, per poter standardizzare il modo in cui alcuni componenti dell'applicazione eseguono determinate operazioni.

La prima interfaccia è quella per il parsing dei dati JSON:

```
1 public interface InterfaceParser
2 {
3   public List<RowWrapper> parseData(JSONObject jso);
4
```

```
}
```

Definisce che l'input del parsing deve essere un JSONObject e l'output viene già inserito in una lista di contenitori precedentemente definiti per la memorizzazione dei dati di una singola tupla di una tabella.

La seconda interfaccia è per l'esecuzione dell'aggiornamento dei dati interni all'applicazione. Viene definito che per eseguire un'operazione di update dei dati sono necessari un'istanza di un content resolver e una lista di row wrapper che contengono i dati da aggiungere, cancellare o modificare nel database interno.

```
public interface InterfaceDataUpdater
{
  public void doUpdate(ContentResolver CR, List<RowWrapper> data);

}
```

Aggiornamento dei dati locali

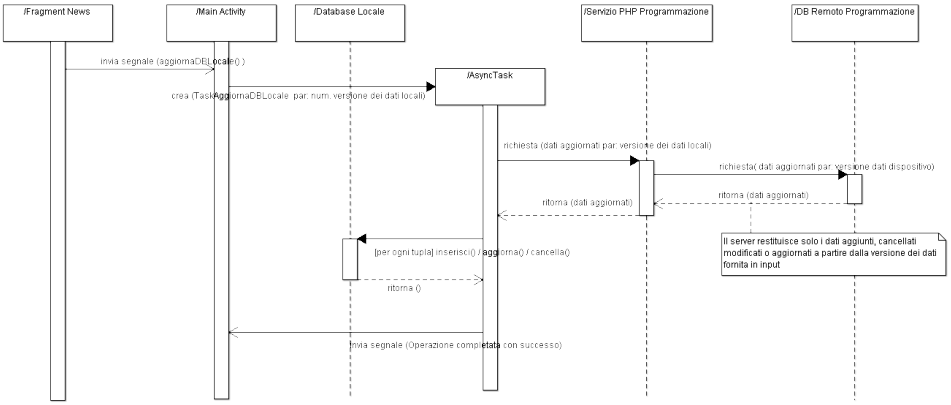
L'effettivo aggiornamento dei dati del database locale viene gestito da un metodo che riceve un'istanza di content resolver dal chiamante, indipendentemente da quale parte dell'applicazione sia, e un insieme di dati salvati nell'apposita struttura, e si occupa di effettuare per ogni singola tupla l'operazione richiesta.

Una cosa particolare da notare è che l'aggiornamento della versione dei dati locali viene gestita mascherando l'informazione da tupla e inserendola nella lista con tutte le altre. Verrà poi "riconosciuta" l'istanza di rowwrapper che contiene l'informazione sulla versione e verranno effettuate le relative operazioni di aggiornamento del dato locale (vedi: 4.1.1). In tutti gli altri casi, a seconda dei parametri impostati per ogni tupla, viene effettuato tramite il content resolver l'operazione di inserimento o cancellazione.

Ogni qualvolta una parte del software dovrà modificare i dati sul database dovrà necessariamente utilizzare questo metodo che è stato progettato per essere l'unico a comunicare con il content provider tramite le diverse istanze di content resolver.

Per permettere all'utente di effettuare la sincronizzazione del database locale con quello remoto si è creato un apposito pulsante sulla ActionBar, sempre visibile durante la navigazione nelle diverse schermate.

Figura 4.4: Diagramma di Sequenza che riassume le fasi principali del processo di aggiornamento e sincronizzazione del database locale con quello remoto



Struttura e logica del fragment

Questa parte dell'applicazione è quella che produce la maggior parte delle viste con cui l'utente interagisce, ci sono infatti numerose sezioni derivate da quelle previste originariamente sul sito del portale. La navigazione radice ->foglia consiste in tre o quattro livelli:

Primo Livello Il menu dell'applicazione, con l'elenco delle sezioni.

Secondo Livello É opzionale, consiste nella categoria del tipo di dato selezionato, come l'anno accademico per le dispense delle lezioni, o il numero della settimana per l'elenco degli esercizi.

Terzo Livello Elenca tutti gli elementi della categoria selezionata.

Quarto Livello Visualizza i dettagli dell'elemento selezionato, nel caso siano presenti allegati, fornisce la possibilità di effettuare il download sul dispositivo.

Il primo livello è realizzato dal menu, non vi è lettura del database bensì gli elementi sono elencati caricando i valori da un vettore di stringhe salvato nei file locali della applicazione. Dal secondo livello in poi i dati sono il risultato di una query sul database locale che carica di volta in volta solo le informazioni necessarie per l'elenco visualizzato, ciò significa che per realizzare l'elenco delle news verranno letti dal database solo i titoli, e non il contenuto della news. Nel momento in cui viene selezionata una voce di uno degli elenchi di una vista, si passa al livello inferiore fornendo come parametro l'ID dell'elemento selezionato precedentemente, che risulterà il parametro del loader che si occuperà del caricamento dei dati nel cursore della vista successiva.

Risulta dunque evidente come il funzionamento di tutte le viste in esame possa ricondursi a un pattern comune, con solo minime variazioni:

- Il layout della singola riga.
- I dati caricati sulle diverse righe e il loro numero.
- Il comportamento al verificarsi di un *tap* su un elemento.
- La tabella sorgente dei dati.

Il componente che effettua il caricamento dei dati è però lo stesso e la logica di funzionamento delle viste è identica.

La soluzione adottata è stata quindi quella di realizzare una singola classe di tipo Fragment che riceve in input, in fase di creazione, un bundle con all'interno tutte i parametri che stabiliscono il preciso funzionamento.

Si mostra un esempio di creazione del bundle

```
1 case ELENCO_NEWS:
2       {
3         mb = new MyBundle(  ELENCO_NEWS, // Identificativo del tipo di vista che sara' realizzata
```

```
4            DBTGnews.newPROJECTION, // Elenco dei campi della tabella che si vogliono caricare
5            DBTGnews.newSELECTION, // Stringa di selezione di WHERE
6            DBTGnews.newSELECTIONARGS, // Argomenti per la selezione
7            DBTGnews.newSORTORDER, // Eventuale string che definisce l'ordinamento
8
9            MyContentProvider.TAB_NEWS_CONTENT_URI, // Uri della tabella da cui si vogliono
     caricare i dati
10           ELENCO_NEWS,      // ID del loader che effettuera' il caricamento dati
11
12           R.layout.row_progr_elenco, // identificativo della risorsa per il layout della riga
13           R.layout.activity_prog_elenco, // identificativo della risorsa per il layout della vista
14           R.id.listProgrElenco,      // riferimento alla ListView presente nel layout XML
15
16           new String[]{DBTGnews.DESCRIZIONE}, // Elenco di campi che l'adapter carichera'
     sulla ListView
17                                   // E' un sottoinsieme dei campi della PROJECTION
18           new int[]{R.id.textTitolo}        // Elenco delle TextView da associare a ogni singolo
     valore
19           );
20         return mb;
21       }
```

SimpleCursorAdapter e CustomCursorAdapter Un altro particolare importante consiste nella selezione dell'adapter: Nei livelli 2 e 3 precedentemente illustrati, la vista consiste in una semplice ListView che mostra dei dati, per la realizzazione di questo tipo di interfaccia è sufficiente e ottimale l'utilizzo di un *SimpleCursorAdapter*, classe di libreria Android preposta esattamente a questo tipo di utilizzo.

Nel livello 4 invece, quello che visualizza i dettagli di un elemento, è necessaria una vista più articolata e con diverse funzionalità quali pulsanti, radiobutton, ecc.

Per questo motivo è necessario implementare un *CursorCustomAdapter* estendendo la classe CustomAdapter e definire come deve essere strutturata la vista.

Quindi all'interno del fragment, nel momento in cui viene istanziato e assegnato l'adapter, verrà scelto quale dei due utilizzare basandosi sul codice identificativo della vista corrente, fornito nel bundle.

```
1    /** Viene selezionato l'adapter da utilizzare per questa istanza del fragment
```

```
2       * Se si sta visualizzando il dettaglio di un elemento si utilizza un CustomCursorAdapter
3       * Nei casi di liste semplici si utilizza un SimpleCursorAdapter
4       */
5     @Override
6     public void onActivityCreated(Bundle savedInstanceState)
7     {
8       super.onActivityCreated(savedInstanceState);
9
10      if (BGen.isThirdLevel(mTypeOfThis))
11      {
12        mDataSource = new CustomCursorAdapter
13                    (
14                      getActivity().getApplicationContext(),
15                    null,// viene passato un cursore NULL, sara' il loader a aggiungerlo in seguito
16                      SimpleCursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER
17                    );
18      }
19      else
20      {
21        mDataSource = new SimpleCursorAdapter
22                    (
23                      getActivity().getApplicationContext(), // Context
24                    mLayoutRow, // Layout della singola riga
25                    null, // Il cursore, inizialmente null, verra' aggiunto dal loader alla fine delle
       operazioni di caricamento
26                    mAdapterColumnList, // Elenco delle colonne del cursore di cui si vogliono visualizzare
       i dati
27                    mAdapterTextviewList, // Elenco dei riferimenti alle TextView che ospiteranno i valori
28                    SimpleCursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER // Flag per attivare il
       monitoraggio dei cambiamenti dei dati nel cursore
29                    );
30      }
31
32      mListView.setAdapter(mDataSource);
33      getActivity().getSupportLoaderManager().restartLoader(LOADER_ID, null, this);
34    }
```

InitLoader vs RestartLoader e istanze di loaders Estremamente importante è il comando `getActivity().getSupportLoaderManager().restartLoader(LOADER_ID, null, this);`. Questa istruzione si occupa di inizializzare il loader con l'ID corrispondente se non esistente, se invece esiste viene distrutto il contenuto attuale e effettuato un nuovo caricamento. Questo dettaglio è importante perchè ogni "livello logico delle viste" condivide lo stesso loader, per evitare

un aumento esponenziale delle istanze in caso di navigazione in numerose viste diverse, e quindi se si utilizza il tasto *Indietro* per tornare al livello superiore, per poi selezionare un elemento diverso e tornare al livello inferiore, è necessario che venga effettuato un nuovo caricamento.

Il comando InitLoader al contrario, se il cursore esiste già, restituisce quest'ultimo con i dati al suo interno, portando quindi a una errata visualizzazione dei dati richiesti.

Questo problema si risolverebbe facendo in modo che ogni istanza di fragment creasse un loader univoco, portando a sfruttare una delle caratteristiche più importanti dei loaders, cioè il fatto che se la vista viene distrutta e ricreata, il loader rimane invece istanziato e restituisce nuovamente i dati senza effettuare un nuovo caricamento.

Il motivo per cui si è scelta una diversa soluzione deriva dalla volontà di non sovraccaricare il sistema con un numero eccessivo di istanze di loaders. Inoltre, per come è progettata la navigazione all'interno dell'applicazione, si ritiene che sia un evento tendenzialmente poco frequente la visualizzazione di una stessa vista con gli stessi dati al suo interno, mentre è molto più probabile un susseguirsi di viste dello stesso "livello logico" ma con dati caricati diversi.

Per queste motivazioni si è scelta questa soluzione implementativa ritenendola quella più performante per questo specifico caso.

4.1.5 Main activity

L'applicazione consiste in una singola activity e viene suddivisa in fragment, che risultano essere i veri componenti dell'UI con cui interagisce l'utente. La Main Activity risulta avere quindi pochi, specifici, compiti:

- Gestire la ActionBar.
- Gestire il DrawerLayout.
- Gestire la comunicazione fra fragment.

Quest'ultimo punto è forse il dettaglio più interessante: La activity implementa le interfacce che usano i fragment per inviare all'esterno dati e chiamate a altri fragment. Per passare, per esempio, dalla visualizzazione di un fragment che contiene una listview a un diverso fragment che visualizza il dettaglio dell'elemento precedentemente selezionato, il primo richiama il metodo implementato dalla main activity, fornendo eventuali parametri necessari alle operazioni, ed è solo la activity madre che si occupa di ricevere le informazioni e passarle al fragment destinatario.

Questo è un aspetto fondamentale del funzionamento dei fragment, ognuno di essi non conosce i dettagli dell'ambiente al proprio esterno ed è in grado di comunicare unicamente con la activity che lo contiene. Questa comunicazione è implementata nei listener che la main activity implementa e che specificano cosa deve avvenire in risposta a una specifica chiamata di un fragment.

4.1.6 Interpretazione delle stringhe JSON

Per questo progetto, si è deciso di unire il processo di decodifica automatica di una stringa JSON con le operazioni per la formattazione dei dati in wrapper personalizzati. Sono state quindi create due classi, con interfaccia comune, che implementano le operazioni specifiche per il tipo di stringa JSON restituita dai due server con cui dialoga l'applicativo.

Il concetto alla base è molto semplice, si sfruttano le classi JSONObject e JSONArray per navigare l'albero DOM e poi ciclicamente si estraggono i dati dall'oggetto che li contiene inserendo i valori con le rispettive chiavi in istanze della classe RowWrapper, suddividendoli per tupla.

Ogni oggetto foglia è infatti un JSONObject che racchiude un array associativo con la chiave rappresenta il campo della tabella e il valore corrispondente per la tupla corrente.

Il vantaggio dell'utilizzo di queste classi custom è che in caso di modifica del metodo per la creazione delle stringhe lato server, queste saranno le uniche due parti di codice dell'intero progetto da aggiornare.

Alcuni dettagli dell'implementazione La gestione della versione dei dati viene gestita mascherando il dato da tupla, appositamente parametrizzata per essere riconosciuta dal metodo che esegue l'inserimento.

```
1          if (mTN.equalsIgnoreCase(CL_PROGR.C_VERSION))
2          {
3            // Se sto leggendo il nodo Versione, creo un oggetto RowWrapper particolare
4            mVersion = mListOfTables.getString(CL_PROGR.C_VERSION);
5            mRow = new RowWrapper(true, mVersion);
6            mRowList.add(mRow);
7          }
```

Durante la fase di decodifica viene eseguita anche la pulizia dei dati, che consiste nell'eliminazione delle parti di stringa JSON inutili ai fini dell'Applicazione e l'ottimizzazione di alcuni dati che vengono già formattati per l'utilizzo che ne verrà fatto. Ogni valore è inoltre convertito in codifica *UTF-8* e ripulito di eventuali tag HTML.

```
1  public static String cleanStringWithKey(String key, String dirtyValue)
2  {
3    String cleanValue = null;
4
5    byte[] b;
6    try
7    {
8      b = dirtyValue.getBytes(CL_APP.ENCODE_UTF8);    // codifica del testo
9      cleanValue = new String(b, CL_APP.ENCODE_APP);
10     cleanValue = Html.fromHtml(cleanValue).toString(); // rimozione dei tag html
11   }
12   catch (UnsupportedEncodingException e)
13   {
14     // TODO Auto-generated catch block
15     e.printStackTrace();
16   }
17   if (cleanValue == null)
18   {
19     return dirtyValue;
20   }
21   else
22   {
23     return cleanValue;
24   }
25 }
```

Il campo *allegato*, che contiene le informazioni riguardo il file di cui si vuole permettere il download, risulta un caso particolare in quanto non risulta essere una coppia chiave-valore bensì un oggetto contenente a sua volta diverse informazioni. Ai fini dell'implementazione dell'Applicazione sono necessari solo due di queste informazioni, quindi da ogni oggetto *allegato* vengono generati due diverse coppie di valori, scartando il resto delle informazioni

```
else if ( mKey.equals(CL_PROGR.C_ALLEGATO) )
                {
                  /**
                   * Il campo allegato non e' una semplice coppia chiave-valore bensi'
                   * un oggetto di cui ci interessa il path del file per il download
                   * ottenuto da 2 campi dell'oggetto, e il filename per la visualizzazione
                   * all 'utente
                   * Quindi eseguo 2 inserimenti in una volta sola
                   */
                  mValue = getFilePath(mJSO.getJSONObject(mKey));
                  mRow.addValue(DBTAallegato_categor.ALLEGATO_PATH, mValue);
                  mValue = getFileName(mJSO.getJSONObject(mKey));
                  mRow.addValue(DBTAallegato_categor.ALLEGATO_FILENAME, mValue);
                }
```

4.2 Descrizione dell'implementazione del servizio PHP

4.2.1 Servizio PHP

Per implementare i servizi necessari al funzionamento dell'applicazione Android senza dover modificare lo stato attuale del CMS, si è deciso di creare un file PHP aggiuntivo che racchiuda tutte le funzionalità necessarie.

Il funzionamento generale consisterà quindi in una chiamata HTTP da parte dell'applicazione verso il file PHP ospitato sul server del CMS Esercizi2013. La selezione del servizio desiderato avverrà tramite parametro di tipo Http-GET, mentre il passaggio dei parametri avverrà con protocollo Http-POST.

Al momento della ricezione della chiamata HTTP, il server eseguirà le seguenti operazioni:

1. Apertura del database e del file contenente i dati dei session-id (utilizzando la libreria esterna Flintstone) [24].

```
/*Configurazione per l'accesso al database*/
include '_slash-config.php';
/*Libreria per il salvataggio dei session id degli utenti su file, senza utilizzare il
    database*/
include 'flintstone.class.php';
error_reporting(E_ALL);
ini_set('display_errors', '1');

/*Collegamento al database*/
$mysqli = new mysqli(_DB_URL, _DB_USERNAME, _DB_PASSWORD, _DB_NAME);


/*Controllo errori di connessione al DB*/
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
```

2. Controllo dell'input

```
$email = $mysqli->real_escape_string($_POST['e_mail']);
$password = $mysqli->real_escape_string($_POST['password']);
```

3. Definizione ed esecuzione della query relativa al servizio richiesto

```
$query = "Select esercizio.id_esercizio, esito, id_consegna_esercizio from esercizio
                LEFT OUTER JOIN consegna_esercizio
                ON esercizio.id_esercizio = consegna_esercizio.id_esercizio
                AND ultimo = 1 AND id_studente = '$id_studente'
                WHERE esercizio.id_blocco = '$id_blocco'";
```

4. Nel caso di operazione di login o logout verranno eseguite le relative operazioni sul file dei session-id

   É importante sottolineare che la creazione del sID utilizzata in questo caso è estremamente esemplice e banale, per ottimizzare la sicurezza è

necessario creare stringhe univoche e generate in modo casuale. Questo risulta una importante modifica da effettuare prima di mandare il servizio in produzione.

```
$sID = $email . $password; // Creo il sID
$siddata["sID"] = $sID;
$siddata["user_class"] = "studente";
$users->set($sID,$siddata ); // Inserisco la coppia chiave-valore nel file salvato sul
    server
$json["data"][0]["sID"] = $sID; //Inserisco il sID nella stringa JSON di risposta
```

5. Esecuzione della query sul database

```
$result = $mysqli->query($query) or die($mysqli->error.__LINE__);
```

6. Encoding dei dati in formato JSON

```
while ($row = $result->fetch_assoc())
  {
    array_push($json["data"], $row);
  }
```

7. Restituzione al chiamante di una stringa JSON contenente i dati relativi al risultato dell'operazione unitamente a un flag di conferma della conclusione corretta del processo

```
header('Content-type: application/json');
echo json_encode($json);
```

4.3 Funzionalitá

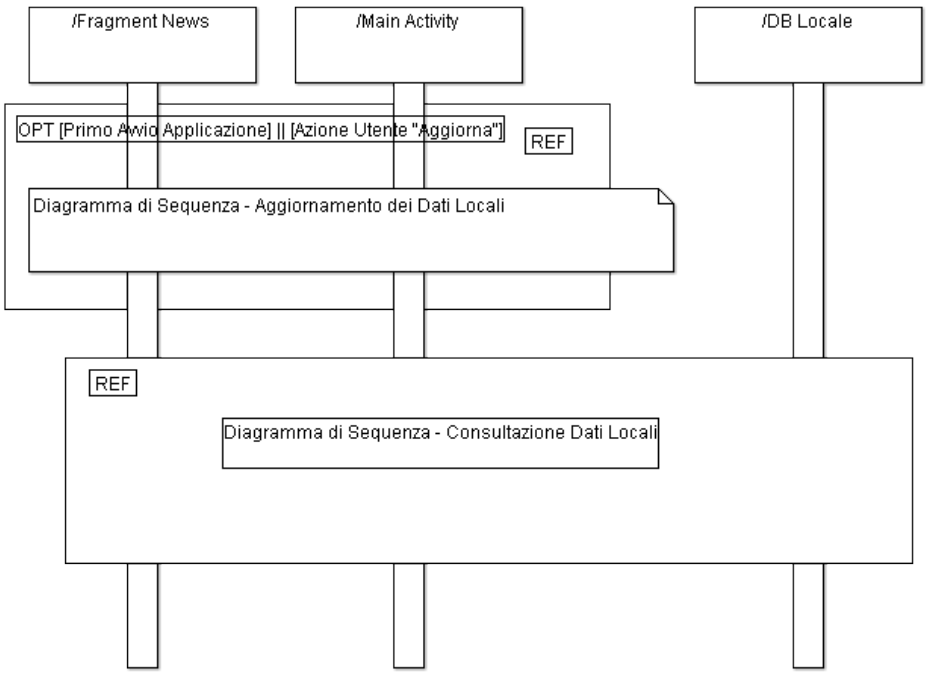
4.3.1 Visualizzazione news del portale Programmazione.info

La schermata di benvenuto dell'applicazione mostra le news del portale di Programmazione, con l'obiettivo di fornire agli studenti un accesso immediato alle ultime comunicazioni. I dati mostrati nella listview sono caricati

Figura 4.5: L'elenco delle news presenti sul portale Programmazione.Info, selezionando una voce si accede al dettaglio

dal database locale, che a sua volta viene sincronizzato con il database remoto tramite un processo in background attivabile dall'utente. Durante il primo avvio dell'applicazione viene effettuata una sincronizzazione iniziale, le successive avverranno su azione esplicita dell'utente.

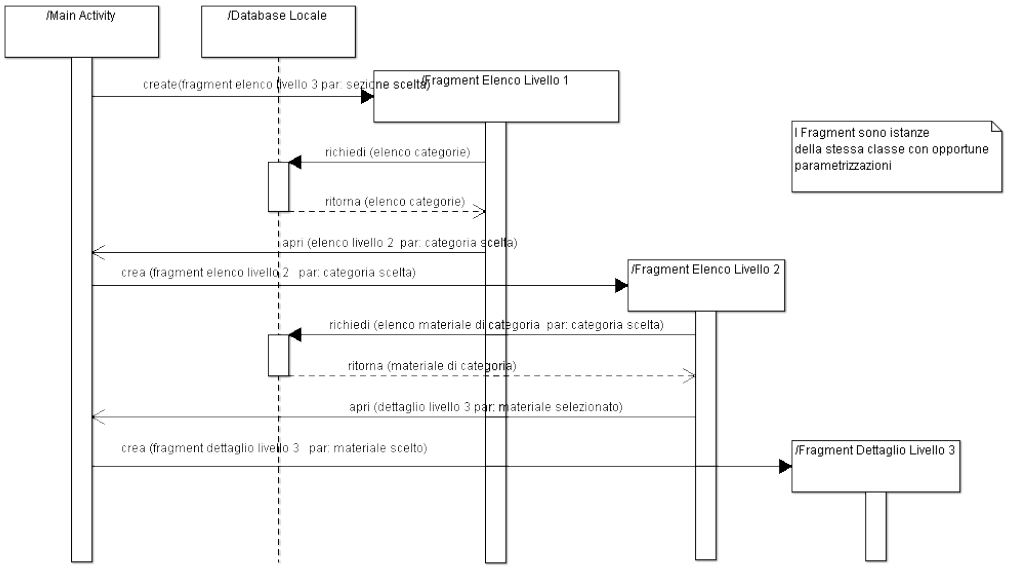
Figura 4.6: Diagramma di Sequenza che mostra il processo di consultazione di una news dando enfasi allo specifico caso del primo avvio assoluto della applicazione



4.3.2 Visualizzazione delle informazioni sul corso di Programmazione

Nella apposita sezione è possibile consultare le informazioni riguardanti il corso, i dati sono salvati in locale sull'applicazione e quindi sempre disponibili.

Figura 4.7: Diagramma di sequenza che illustra il processo generale di consultazione di materiale salvato in locale sul dispositivo



4.3.3 Consultazione e download dei materiali didattici

Viene fornito accesso a tutti i materiali presenti sul Portale Programmazione.info:

- Dispense delle lezioni in aulta e di laboratorio dell'attuale anno accademico e dei precedenti.

- Tutti i testi dei compiti d'esame degli ultimi anni.

- Esempi di codice a scopo didattico forniti dai docenti.

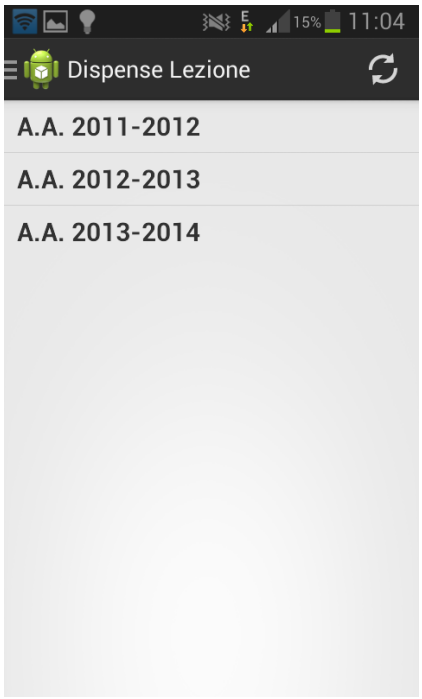
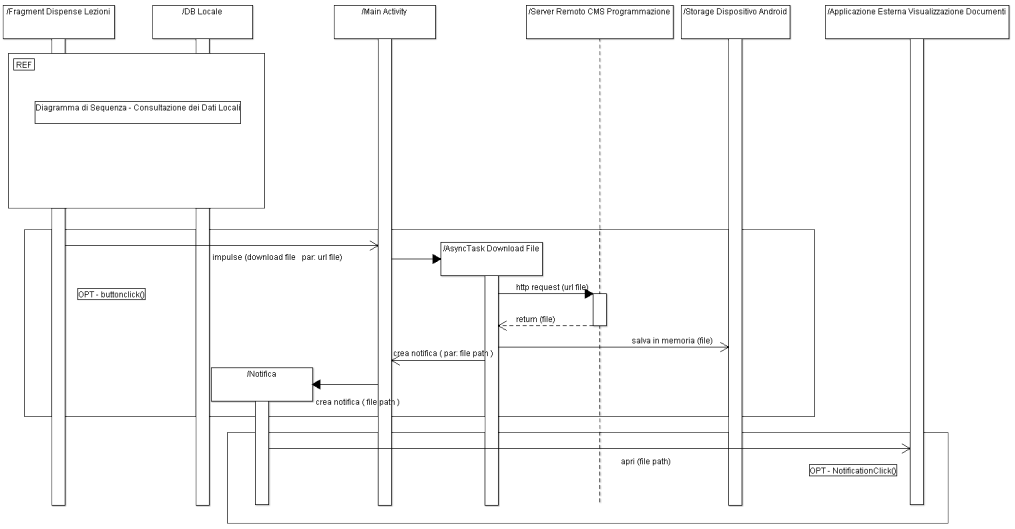


Figura 4.8: L'elenco degli anni accademici che suddividono l'insieme delle dispense delle lezioni

Figura 4.9: Diagramma di Sequenza che illustra il processo necessario per il download di materiale didattico dal Portale Programmazione.Info



4.3.4 Consultazione dell'elenco degli esercizi settimanali propedeutici per l'accesso all'esame finale

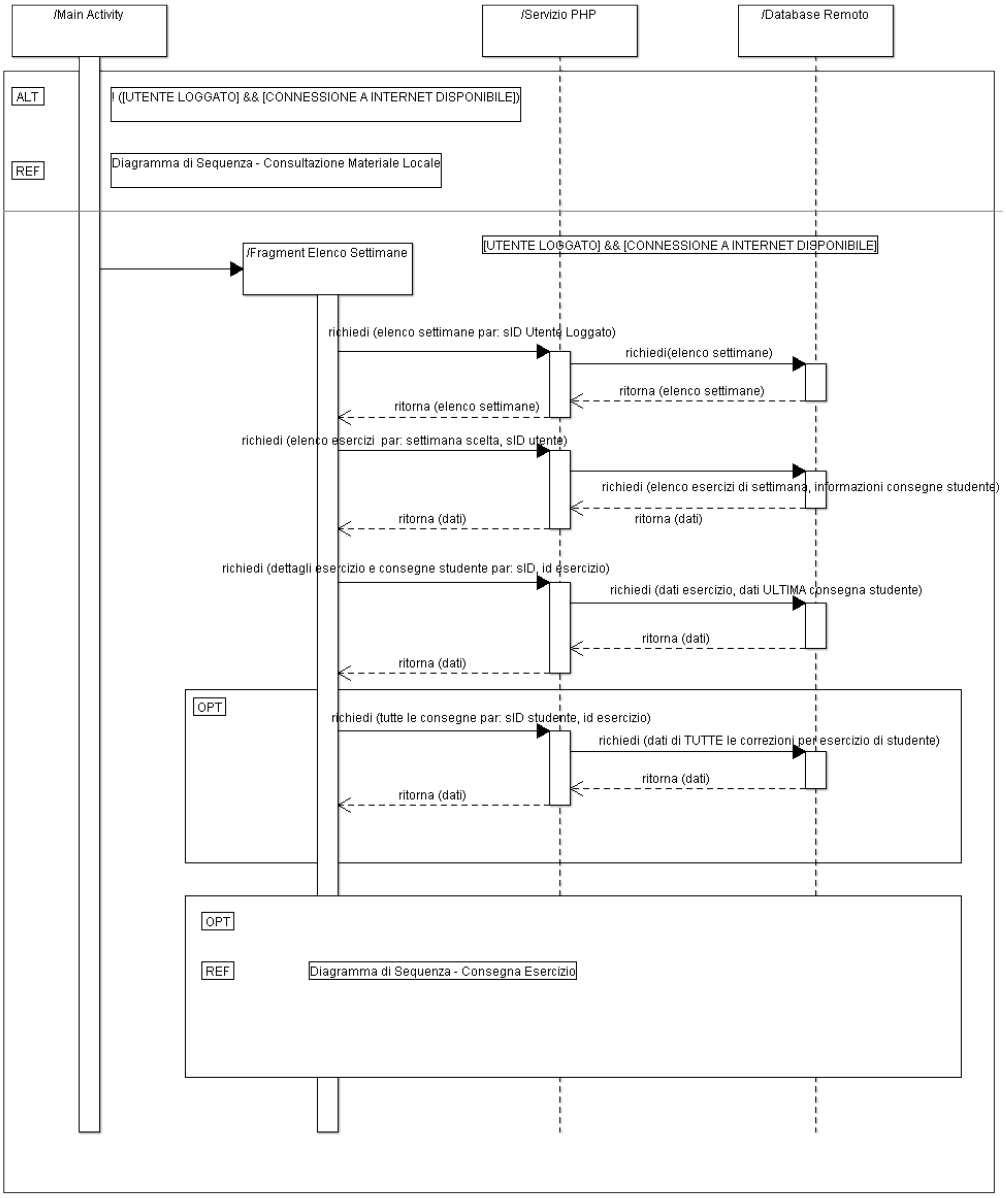
Lo studente ha accesso all'elenco degli esercizi che dovrà consegnare per poter accedere alla prova finale. Queste informazioni sono consultabili da due differenti fonti:

CMS Programmazione Le informazioni consistono nel testo di ogni esercizio suddiviso per settimana, sono salvate sul dispositivo e quindi consultabili in modalità offline.

CMS Esercizi Le informazioni comprendono, oltre al testo dell'esercizio, anche le informazioni sullo stato delle consegne da parte dello studente. I dati sono scaricati ad ogni consultazione dal server remoto e richiedono che lo studente effettui il login con il proprio account.

Nel momento in cui lo studente seleziona la relativa sezione nel menù viene effettuato un controllo sulla presenza di connessione internet e viene controllato se l'utente risulta loggato regolarmente come studente, se i due controlli hanno esito positivo viene visualizzato l'elenco degli esercizi con annessi i dati di consegna (vedi figura 4.11), in caso negativo vengono visualizzati solo titolo e testo dell'esercizio, salvati in locale sul dispositivo. La sequenza di operazioni è schematizzata in figura: 4.10.

Figura 4.10: Diagramma di Sequenza che illustra i differenti casi per la consultazione degli esercizi settimanali



In caso di login effettuato con successo, nella schermata di dettaglio dell'esercizio vengono visualizzate le informazioni riguardanti l'ultima consegna effettuata dallo studente, in ordine temporale, per l'esercizio selezionato (vedi

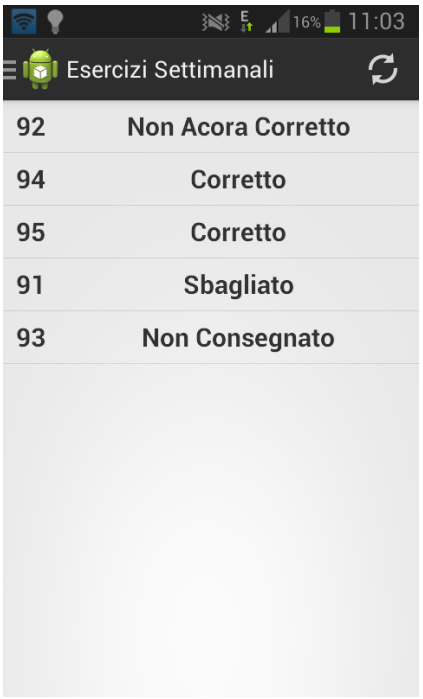


Figura 4.11: L'elenco degli esercizi di una settimana con annesso lo stato di correzione dei singoli esercizi

figura 4.12. Nella stessa schermata è possibile accedere all'elenco di tutte le consegne precedenti e effettuare una nuova consegna, procedendo all'upload del relativo file contenente il codice sorgente che risolve l'esercizio (vedi figura 4.13).

Consegna degli esercizi In fase di analisi ci si è posti il problema se questa funzionalità avesse concrete ragioni per essere implementata in una applicazione di questo tipo. Risultava infatti improbabile che uno studente potesse salvare i propri files sorgente in uno smartphone (o in un tablet) per poi effettuarne la consegna tramite il dispositivo, si riteneva decisamente più realistico che il codice fosse prodotto, salvato e caricato direttamente da PC.

L'idea che ha segnato la soluzione di questa problematica è rappresentata da una delle caratteristiche principali del sistema operativo Android: gli *Intents*. Tramite questa funzionalità è stata infatti implementata la possibilità per lo studente di selezionare il file sorgente da consegnare recuperandolo da uno qualsiasi dei servizi cloud disponibili sul mercato e installabili, tramite applicazione proprietaria, su un dispositivo Android.

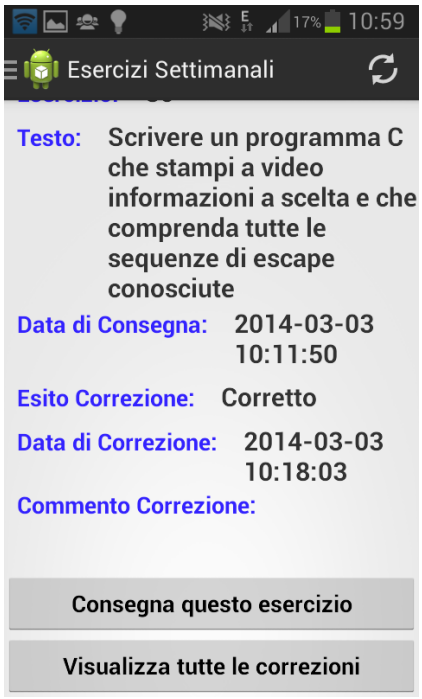


Figura 4.12: La schermata di dettaglio di un esercizio, in caso sia stata effettuata almeno una consegna, vengono visualizzati i dettagli dell'ultima in ordine temporale

Figura 4.13: Lo storico delle consegne di un esercizio, visualizzabile da uno studente selezionando il relativo bottone nella schermata di dettaglio di un esercizio

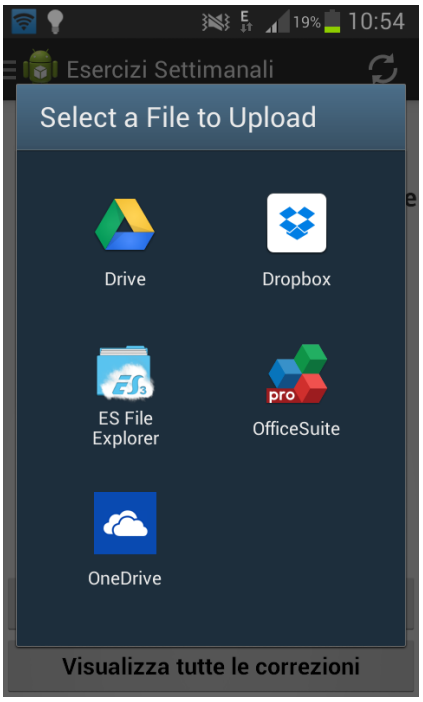
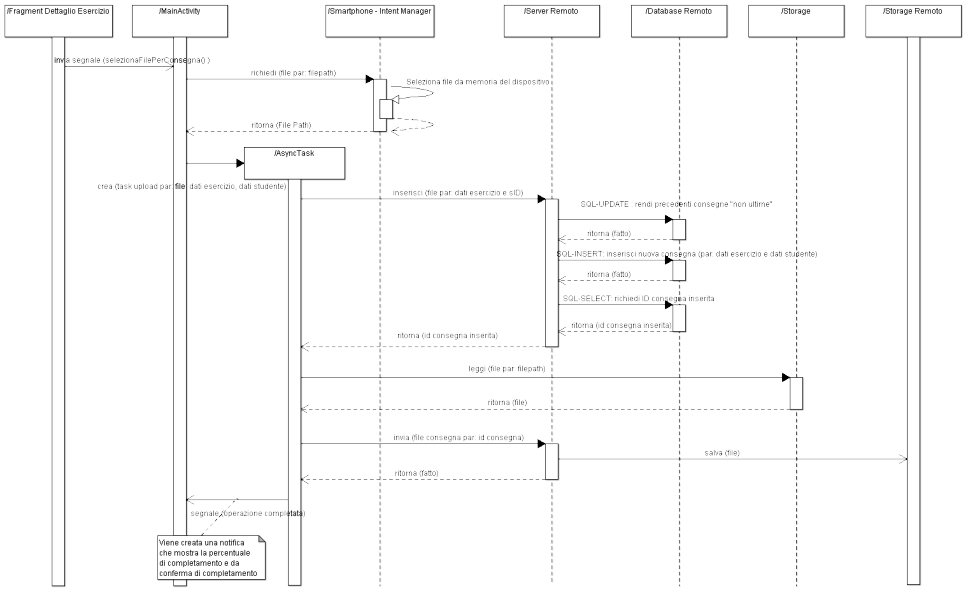


Figura 4.14: Il pop-up mostra la selezione fra diversi servizi di *cloud storage* installati sul dispositivo, lo studente potrà utilizzarli per caricare i file sorgenti dal proprio PC, per poi inviarli tramite l'applicazione in un secondo momento. In alternativa è possibile selezionare un file dalla memoria del telefono con l'applicazione preferita dall'utente.

Lo studente potrà quindi produrre l'elaborato su PC, salvarlo su *Dropbox*, *Google Drive*, *Microsoft OneDrive*, etc, e consegnare l'esercizio "in mobilità" con la comodità di un dispositivo portatile.

Figura 4.15: Diagramma di Sequenza che illustra le fasi necessarie per la consegna di un esercizio da parte di uno studente



4.3.5 Correzione degli esercizi da parte di un Operatore registrato

L'applicazione è stata progettata con lo scopo non solo di risultare un utile strumento per gli studenti, ma anche per gli amministratori del portale e per gli operatori addetti alla correzione degli elaborati consegnati dagli studenti.

In fase di login è infatti possibile selezionare la modalità *Operatore* che permette, tramite username e password pre-registrati, l'autenticazione con permessi diversi da quelli di un semplice studente.

In questa modalità è infatti possibile accedere all'elenco di tutte le consegne non ancora corrette di tutti gli studenti.

I dati sono caricati sul momento dal server remoto, risultando quindi sempre aggiornati.

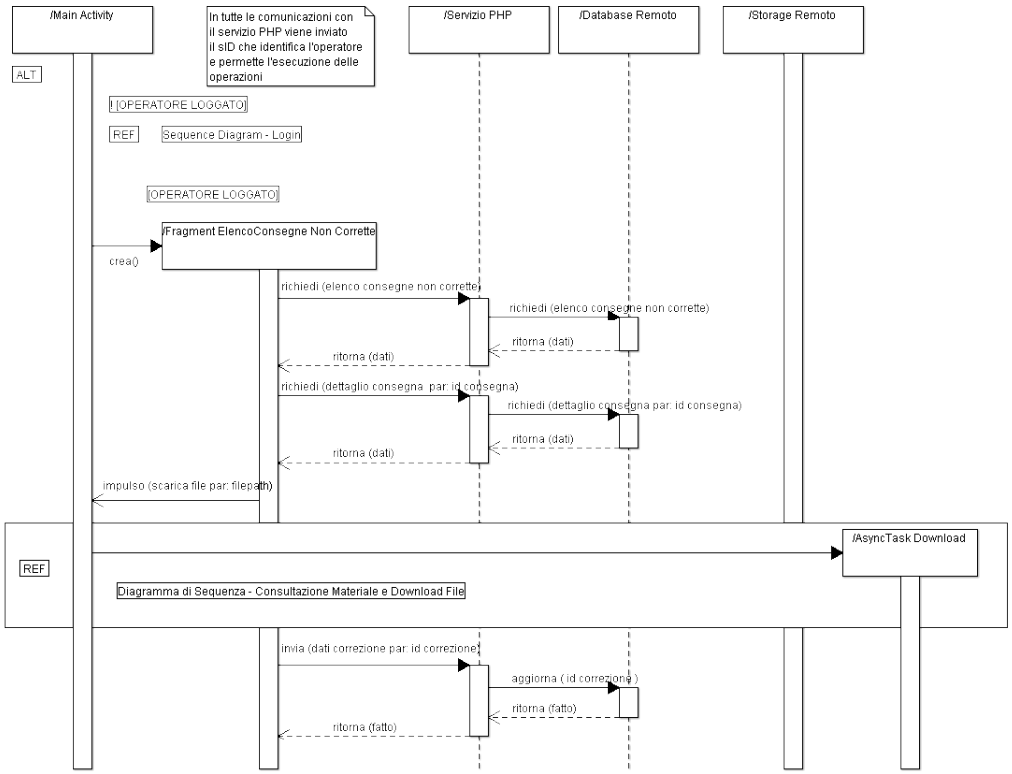
Figura 4.16: Elenco delle consegne non ancora corrette, visualizzate da un operatore. Vengono ordinate in ordine cronologico dalla più recente.

Le consegne sono elencate in ordine cronologico e permettono in primis la consultazione, in seguito la visualizzazione del dettaglio dell'esercizio, del testo di quest'ultimo, e dei dati relativi alla singola consegna (vedi figura 4.16).

É quindi possibile per l'operatore *effettuare il download del file* consegnato dallo studente, visualizzarlo tramite applicazioni esterne adatte alla consultazione di codici sorgente, ed eseguire il *commit* della correzione dell'esercizio, inserendo l'esito (*Corretto* / *Non Corretto*), e l'eventuale commento per lo studente, che faciliterà il riconoscimento degli errori e la loro correzione per una futura consegna (vedi figura 4.17).

Figura 4.17: Schermata di dettaglio della correzione di un esercizio da parte di un operatore. Viene visualizzato il testo dell'esercizio e le relative informazioni, e viene data la possibilità di effettuare il download del file consegnato dallo studente per poterlo valutare. Infine è possibile aggiungere un commento relativo alla correzione e deciderne l'esito.

Figura 4.18: Diagramma di Sequenza che illustra le fasi necessarie per la correzione di una consegna, da parte di un Operatore autenticato



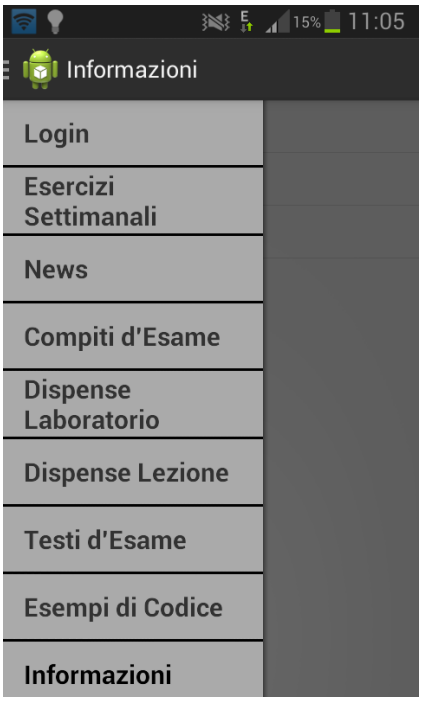


Figura 4.19: Il menù dell'applicazione, implementato utilizzando il Navigation Drawer, con uno stile grafico semplice ed essenziale

4.4 Componenti dell'interfaccia grafica

4.4.1 Navigation drawer

Il Navigation Drawer è uno dei widget più recenti forniti dal sistema Android per il design dell'interfaccia grafica delle applicazioni [25]. Consiste in un pannello che viene visualizzato con uno *swipe*, generalmente dalla sinistra dello schermo verso destra. Il pattern standard prevede che questo vada a coprire circa i $\frac{2}{3}$ della larghezza dello schermo e l'intera altezza, ad eccezione dell'Action Bar (vedi figura 4.19)

É inoltre previsto che nell'Action Bar venga inserito un pulsante sulla sinistra che sostituisca la gesture di swipe e renda noto all'utente della presenza del menu durante il primo avvio.

Uno dei vantaggi di questo pattern per il menù dell'applicazione è che risulta accessibile in qualsiasi fase della navigazione dell'applicazione e rende disponibile molto spazio per l'elenco delle voci di sezione, numerose in questa applicazione.

É infine importante sottolineare che si tratta di un pattern utilizzato praticamente in tutte le applicazioni ufficiali di Google e che sta diventando lo standard di riferimento più diffuso in tutto il Play Store, dunque, utilizzandolo si fornisce all'utente un layout familiare e intuitivo, caratteristica fondamentale per una applicazione ben progettata [26]

Implementazione La prima cosa da fare è inserire il Navigation Drawer nel layout XML:

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <!-- The main content view -->
   <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!-- The navigation drawer -->
    <ListView
        android:id="@+id/drawer"
        android:layout_width="200dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:background="@android:color/background_dark"
        android:choiceMode="singleChoice"
        android:divider="@android:color/background_dark"
        android:dividerHeight="0dp" ></ListView>

</android.support.v4.widget.DrawerLayout>
```

Alcuni aspetti da sottolineare:

- La view principale, in questo caso il frame per il fragment, deve essere il primo elemento.
- L'altezza del drawer deve essere match_parent.
- La view del Navigation Drawer (in questo caso la ListView) deve specificare la sua gravità tramite l'attributo android:layout_gravity.

Successivamente è necessario implementare un listener per l'azione da effettuare in risposta al *tap* da parte dell'utente, e andare a specificare le azioni relative ad ogni voce del menù

```
switch (position)
{
  case MENUNEWS:
  {
    mFragment = new FragmentMorphing();
        newBundle = BGen.generateBundle(BGen.ELENCO_NEWS, // ottengo il fragment successivo a quello chiamante
                                        null); // la selezione del fragment precedente, ad esempio l'id di una news. in questo caso e' il livello 1 quindi non c'e'
        args.putParcelable(MyBundle.ARG_TYPE_NAME, newBundle);
  }
  break;
  case MENUCOMPITI:
  {
    mFragment = new FragmentMorphing();
        newBundle = BGen.generateBundle(BGen.ELENCO_ANNO_COMPITI_DI_ESAME, // ottengo il fragment successivo a quello chiamante
                                        null); // la selezione del fragment precedente
        args.putParcelable(MyBundle.ARG_TYPE_NAME, newBundle);
  }
  break;
  case MENUELENCOCONSEGNENONCORRETTE:
        {
          if (session.isLoggedIn() && session.isOperator())
          {
            mFragment = new FragmentElencoEserciziDaCorreggere();
          }
          else
          {
            mFragment = new FragmentLogin();
          }
        }
  break;
```

Nel caso di selezione di una sezione relativa al CMS Programmazione, si genera il bundle dei parametri necessari alla configurazione del fragment, mentre nel caso si voglia accedere all'area riservata agli utenti *Operatore* si controlla se è già stato effettuato il login e, in caso negativo, si rimanda l'utente alla relativa schermata.

Una volta completata la configurazione del fragment si procede alla visualizzazione tramite l'utilizzo del *Fragment Manager*:

```
1   if (mFragment != null)
2   {
3     mFragment.setArguments(args);
4
5     FragmentManager mFragmentManager = getSupportFragmentManager();
6
7     FragmentTransaction mTransaction = mFragmentManager.beginTransaction();
8     mTransaction.replace(R.id.content_frame, mFragment);
9           mTransaction.addToBackStack(null);
10          mTransaction.commit();
11
12    mDrawerListview.setItemChecked(position, true);
13    getSupportActionBar().setTitle(mElementiMenu[position]);
14    mDrawerLayout.closeDrawer(mDrawerListview);
15  }
```

É interessante notare il comando `mTransaction.addToBackStack(null);` che garantisce la possibilità di navigare fra un fragment e il precedente tramite il tasto *Indietro* del dispositivo.

Infine, estendendo la classe ActionBarDrawerToggle è possibile definire le operazioni da eseguire nel momento in cui il Navigation Drawer viene aperto o chiuso

```
1   private class CustomActionBarDrawerToggle extends ActionBarDrawerToggle
2   {
3     public CustomActionBarDrawerToggle(Activity mActivity,DrawerLayout mDrawerLayout)
4     {
5       super(
6           mActivity,
7           mDrawerLayout,
8           R.drawable.ic_drawer,
9           R.string.ns_menu_open,
10          R.string.ns_menu_close);
11    }
12
13    @Override
14    public void onDrawerClosed(View view)
15    {
16      // invoca onPrepareOptionsMenu(Menu menu)
17      supportInvalidateOptionsMenu();
18    }
19
```

```
20      @Override
21      public void onDrawerOpened(View drawerView)
22      {
23        // invoca onPrepareOptionsMenu(Menu menu)
24        supportInvalidateOptionsMenu();
25      }
26    }
27
28      /**
29       * Invocato da invalidateOptionsMenu()
30       * Si occupa di nascondere e riattivare le opzioni di menu della actionbar
31       */
32        @Override
33        public boolean onPrepareOptionsMenu(Menu menu) {
34            // Se il navigation drawer e' aperto, si nascondono le azioni relative al contenuto della
        view principale
35            boolean drawerOpen = mDrawerLayout.isDrawerOpen(mDrawerListview);
36            return super.onPrepareOptionsMenu(menu);
37        }
```

4.4.2 Actionbar

La Action Bar è una funzionalità dell'interfaccia grafica di una applicazione che contiene l'icona identificativa dell'applicazione e il suo nome. Risulta inoltre il "contenitore" standard per i pulsanti per le azioni più generali (es. *Ricerca*, *condividi* ecc.) e per il pulsante di accesso al menù delle opzioni.

Utilizzando questo widget si offre un importante elemento di continuità rispetto alle altre applicazioni, aumentando la facilità d'uso dell'applicazione. Esistono inoltre una serie di linee guida ufficiali per la progettazione di una interfaccia che comprenda sia la ActionBar che il Navigation Drawer [27].

Le api per questo componente sono quelle di Android 3.0 (API 11) e sono superiori al livello minimo che si vuole mantenere per lo sviluppo di questo progetto, perciò si è deciso di utilizzare la Action Bar Combat, una versione fornita con la *Android Support Library v7* che risolve questo problema fornendo le stesse funzionalità della versione più recente.

Per questo progetto si utilizza questo widget per fornire due importanti funzionalità:

Navigation Drawer Toggle I design pattern riguardanti il Navigation Drawer vogliono che sia associato all'utilizzo dell'ActionBar, uno degli elementi fondamentali di questo accoppiamento è il pulsante che apre e chiude il Navigation Drawer senza richiedere la gesture *swipe*, è posto in alto a sinistra e porta anche il risultato di *comunicare* all'utente che esiste un menù a scomparsa.

Pulsante di aggiornamento dei dati locali Il secondo pulsante fornito è quello che manda in esecuzione un processo asincrono che si occupa di reperire i dati del CMS Programmazione dal server remoto e salvarli nel database locale all'applicazione.

Conclusioni

L'obiettivo alla base di questa tesi di laurea era fornire agli studenti un accesso ai servizi del portale *Programmazione.Info* ottimizzato per smartphones.

Il risultato del lavoro consiste in una applicazione Android che permette la consultazione dei materiali presenti sul portale con interfaccia ottimizzata per schermi di ridotte dimensioni. Viene inoltre fornita allo studente la possibilità di controllare la situazione delle proprie consegne degli esercizi, consultare i risultati delle correzioni dai docenti, e sottomettere a verifica nuovi esercizi inviandoli direttamente dal proprio smartphone.

Tramite la stessa applicazione è inoltre possibile, per gli operatori addetti alla correzione degli esercizi, accedere alla lista delle consegne in attesa di verifica, visualizzare le richieste del singolo esercizio e il codice sorgente fornito dallo studente e definire poi l'esito della consegna aggiungendo, se necessario, le note riguardanti gli errori riscontrati.

Durante il lavoro si è cercato di dare particolare importanza alle scelte implementative, utilizzando i design patterns indicati dalla documentazione ufficiale di Android e scegliendo sempre i componenti più aggiornati per implementare le singole funzioni, creando quindi un software più complesso da progettare e scrivere ma più efficiente ed ottimizzato.

Non potendo creare l' "applicazione perfetta" ci si è concentrati sull'ottenere qualcosa di funzionale e di ben progettato e strutturato, facendo sì che future evoluzioni possano essere integrate senza stravolgere il lavoro fatto finora. Vi è ovviamente grande margine di miglioramento e ampliamento

del prodotto ma si ritiene di aver creato una solida base che soddisfa tutte le richieste funzionali del committente.

É stato inoltre possibile, tramite l'utilizzo di specifiche librerie di supporto fornite da Google, creare un'applicazione compatibile con il 100% delle distribuzioni di Android presenti oggi sul mercato senza che questo obiettivo portasse alla rinuncia delle funzionalità più recenti e ottimizzate.

Sviluppi Futuri

Una delle caratteristiche fondamentali per il successo di una applicazione per smartphone è senza dubbio la sua interfaccia grafica. Durante il processo di sviluppo ci si è concentrati sul creare una buona usabilità e utilizzare elementi grafici all'avanguardia per fornire all'utente un ambiente familiare e di facile utilizzo, ma è anche di fondamentale importanza creare elementi grafici di impatto, gradevoli alla vista e allo stesso tempo funzionali e che si uniformino il più possibile agli standard presenti nella maggior parte delle applicazioni moderne.

Sarebbe quindi importante poter creare, collaborando con il committente, uno stile grafico adeguato, che richiami gli stili utilizzati per il portale web esistente, così da ottenere un impatto migliore di quello attuale, che risulta estremamente semplice ed essenziale.

La sicurezza delle comunicazioni con i server dei due CMS andrebbe migliorata utilizzando il protocollo HTTPS e migliorando il processo di autenticazione dell'utente. Un ottimo risultato si avrebbe riuscendo a integrare i due sistemi lato server, uno per il portale web e l'altro per l'applicazione mobile, che attualmente utilizzano sistemi di autenticazione diversi e indipendenti, pur basandosi sulla medesima base dati.

Attualmente l'applicazione può essere distribuita tramite file *.apk* liberamente installabile su ogni dispositivo Android, sarebbe tuttavia importante pubblicare l'applicazione sul Google Play Store così da standardizzare il reperimento da parte degli utenti, senza contare i vantaggi che lo store

ufficiale offre nel momento in cui l'applicazione viene aggiornata, notificando automaticamente a tutti coloro che la utilizzano l'esistenza della nuova versione.

Un'altra utile migliorìa sarebbe l'implementazione delle notifiche push per la notifica di nuove news pubblicate sul portale, questa funzionalità richiede un servizio lato server che comunichi con i server di Google, e l'implementazione del servizio specifico GCM (*Google Cloud Messaging for Android*).

Inoltre, l'applicazione è stata sviluppata utilizzando rigorosamente le indicazioni della documentazione ufficiale di Android riguardo i testi [28] Ogni messaggio che l'utente può leggere è salvato in appositi file XML e caricato a runtime, perciò è possibile, semplicemente creando una copia dello stesso file con i testi scritti in una lingua diversa dall'italiano, far sì che l'applicazione risulti multilingua e si adatti al linguaggio di default impostato sul terminale che la esegue.

Infine, grazie all'utilizzo dei fragment per l'implementazione dell'interfaccia utente, è possibile, con poche modifiche al codice, ottimizzare l'applicazione per la visualizzazione su tablet, sfruttando al meglio il maggior spazio messo a disposizione dagli schermi di questi devices.

Bibliografia

[1] Randy Pausch. Randy pausch's last lecture: Really achieving your childhood dreams. In *The Last Lecture*, 2007. URL http://www.cs.cmu.edu/~pausch/Randy/pauschlastlecturetranscript.pdf. Given at Carnegie Mellon University © Copyright Randy Pausch, 2007.

[2] Five Star Equities. Number of smartphones around the world top 1 billion – projected to double by 2015. http://finance.yahoo.com/news/number-smartphones-around-world-top-122000896.html. URL http://finance.yahoo.com/news/number-smartphones-around-world-top-122000896.html.

[3] http://mobithinking.com. Global mobile statistics 2013 part a: Mobile subscribers; handset market share; mobile operators. http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a. URL http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#chinasubs.

[4] Matteo Vitali. Il 132012 venduti nel mondo 1,1 miliardi di device. http://news.vodafone.it/2012/12/05/il-13-del-traffico-internet-proviene-da-smartphone-e-tablet-nel-2012-venduti-nel-mondo-11-miliardi-di-device/. URL http://news.vodafone.it/2012/12/05/il-13-del-traffico-internet-proviene-da-smartphone-e-tablet-nel-2012-venduti-nel-mondo-11-miliardi-di-device/.

[5] Jon Evans. In five years, most africans will have smartphones. http://techcrunch.com/2012/06/09/feature-phones-are-not-the-future/, June 2012. URL http://techcrunch.com/2012/06/09/feature-phones-are-not-the-future/.

[6] Mike Butcher. Time for an african valley? sub-saharan accelerators start to emerge. http://techcrunch.com/2012/06/07/time-for-an-african-valley-sub-saharan-accelerators-start-to-emerge/. URL http://techcrunch.com/2012/06/07/time-for-an-african-valley-sub-saharan-accelerators-start-to-emerge/.

[7] Nokia: The rise and fall of a mobile giant. Dave lee. http://www.bbc.com/news/technology-23947212. URL http://www.bbc.com/news/technology-23947212.

[8] Joel West and David Wood. Tradeoffs of open innovation platform leadership: The rise and fall of symbian ltd. *To be presented at the Stanford Social Science and Technology Seminar*, 1:46, 2011. URL http://siepr.stanford.edu/system/files/shared/documents/2011-03_WestWood_SIEPR.pdf. Joel West is the corresponding author.

[9] Wikipedia. Multitasking (ios). http://en.wikipedia.org/wiki/Multitasking_(iOS), . URL http://en.wikipedia.org/wiki/Multitasking_(iOS).

[10] Darrell Etherington. Android nears 80and blackberry share slides, per idc. http://techcrunch.com/2013/08/07/android-nears-80-market-share-in-global-smartphone-shipments-as-ios-and-blackberry-share-slides-per-idc/, 2013. URL http://techcrunch.com/2013/08/07/android-nears-80-market-share-in-global-smartphone-shipments-as-ios-and-blackberry-share-slides-per-idc/.

[11] http://opensignal.com/. Android fragmentation visualized. http://opensignal.com/reports/fragmentation-2013/, July 2013. URL http://opensignal.com/reports/fragmentation-2013/.

[12] Android Developers. Android developer dashboard. https://developer.android.com/about/dashboards/index.html, . URL https://developer.android.com/about/dashboards/index.html.

[13] Rob van der Meulen Janessa Rivera. Gartner says mobile app stores will see annual downloads reach 102 billion in 2013. http://www.gartner.com/newsroom/id/2592315, September 2013. URL http://www.gartner.com/newsroom/id/2592315.

[14] Android Developers. Application fundamentals. http://developer.android.com/guide/components/fundamentals.html, . URL http://developer.android.com/guide/components/fundamentals.html.

[15] Minhyuk Ko, Yong jin Seo, Bup ki Min, Seunghak Kuk, and Hyeon Soo Kim. Extending uml meta-model for android application. *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, 1:6, 2012.

[16] Wikipedia. Dalvik (software). http://en.wikipedia.org/wiki/Dalvik_(software), . URL http://en.wikipedia.org/wiki/Dalvik_(software).

[17] Sqlite.org. About sqlite. http://www.sqlite.org/about.html. URL http://www.sqlite.org/about.html.

[18] Wikipedia. phpmyadmin. http://en.wikipedia.org/wiki/PhpMyAdmin, . URL http://en.wikipedia.org/wiki/PhpMyAdmin.

[19] Wikipedia. Json. http://en.wikipedia.org/wiki/JSON, . URL http://en.wikipedia.org/wiki/JSON.

[20] Php.net. Php.net. http://www.php.net/, . URL http://www.php.net/.

[21] Php.net. Che cosa può fare php? http://www.php.net/manual/it/intro-whatcando.php, . URL http://www.php.net/manual/it/intro-whatcando.php.

[22] Android Developers. Loaders. http://developer.android.com/guide/components/loaders.html, . URL http://developer.android.com/guide/components/loaders.html.

[23] Android Developers. Fragments. http://developer.android.com/guide/components/fragments.html, . URL http://developer.android.com/guide/components/fragments.html.

[24] XEWeb. Flintstone php library. http://www.xeweb.net/flintstone/. URL http://www.xeweb.net/flintstone/.

[25] Android Developers. Navigation drawer pattern. https://developer.android.com/design/patterns/navigation-drawer.html, . URL https://developer.android.com/design/patterns/navigation-drawer.html.

[26] Android UI Patterns. The new navigation drawer design pattern. http://www.androiduipatterns.com/2013/05/the-new-navigation-drawer-pattern.html, May 2013. URL http://www.androiduipatterns.com/2013/05/the-new-navigation-drawer-pattern.html.

[27] Android Developers. Actionbar. http://developer.android.com/guide/topics/ui/actionbar.html, . URL http://developer.android.com/guide/topics/ui/actionbar.html.

[28] Android Developers. Supporting different languages. http://developer.android.com/training/basics/supporting-devices/languages.html, . URL http://developer.android.com/training/basics/supporting-devices/languages.html.