Capitolo 1

Introduzione ai Google Play services

Per iniziare, vedremo cosa sono i Google Play services ed esamineremo, ad alto livello, le funzionalità di questa libreria, che andremo a utilizzare e descrivere nel dettaglio attraverso vari esempi pratici. Nella seconda parte ci dedicheremo invece alle procedure da seguire per preparare una qualunque applicazione che utilizzi queste librerie. La versione di riferimento è quella disponibile al momento, ovvero la 6.5, la quale ha portato, come vedremo, ad alcune novità per quello che riguarda la procedura di configurazione nelle applicazioni. Prima fra tutte la possibilità di importare solamente le API che interessano e non l'intera libreria, con conseguente risparmio di spazio. Fatto questo, vedremo come preparare lo scheletro di un'applicazione con Android Studio, descrivendone le parti fondamentali. Dove possibile faremo in modo di restare il più possibile indipendenti dall'IDE, anche se Android Studio si sta affermando come lo strumento al momento migliore per lo sviluppo di applicazioni con la piattaforma di Google. La versione da noi utilizzata è la 1.0, resa disponibile verso la fine del 2014 come evoluzione di quella annunciata all'ultimo Google I/O del 2014.

In questo capitolo

- **Che cosa sono i Google Play services?**
- **Installazione e setup di un'applicazione**
- **Aggiornamento dei Google Play services**

Che cosa sono i Google Play services?

Prima di addentrarci nelle diverse librerie messe a disposizione dai Google Play services è importante capire quali siano state le motivazioni che hanno spinto Google a realizzarle.

NOTA

Spesso si fa riferimento ai Google Play services attraverso l'acronimo GPs, che, ovviamente, non ha nulla a che fare con il *Global Positioning System*. Faremo attenzione nei casi di ambiguità.

Fin dalle prime versioni della piattaforma di Android, Google ha dovuto affrontare il problema della frammentazione. Quello che doveva essere un punto a favore di Android, ovvero la possibilità di essere installato e personalizzato su diversi tipi di dispositivi, era infatti diventato un problema, causa di un certo malcontento tra gli utenti. In corrispondenza di ogni nuova release del sistema operativo, era necessario informare i vari produttori di dispositivi, i quali fornivano la propria implementazione. Questo processo non solo richiedeva moltissimo tempo, ma spesso era accompagnato dalla scelta dei produttori di abbandonare alcuni modelli a favore di altri. Molti utenti si ritrovavano quindi con un dispositivo relativamente nuovo, ma non più compatibile con l'ultima versione di Android e questo per una pura scelta di marketing o di costi dell'azienda produttrice. Si trattava di un evidente problema per la diffusione di questo ormai non più nuovo sistema operativo per dispositivi mobili. Per alcuni maliziosi lo stesso Google vedeva in un certo senso perdere il controllo sulla propria creatura. Serviva quindi una soluzione che riducesse il più possibile la dipendenza di Google e dei servizi da essa offerti dai produttori di dispositivi. Si è fatto un ragionamento molto semplice: ci si è chiesti quali fossero gli strumenti legati all'hardware e quali potessero invece essere gestiti a livello software. Per i primi la dipendenza dai produttori è inevitabile, mentre per i secondi è possibile scegliere un approccio diverso, ovvero creare un'applicazione in grado di aggiornarsi automaticamente e di fornire una serie di servizi che, anche alla luce della filosofia di Android (*all applications are created equals*), possono essere utilizzati dalle altre applicazioni. Ecco il motivo della nascita di un'applicazione chiamata, appunto, Google Play services, che contiene tutte le API che descriveremo in questo testo attraverso la realizzazione di alcune applicazioni reali. In questo modo tutti i dispositivi potranno avere sempre l'ultima versione dei GPs ed essere aggiornati con gli ultimi servizi messi a disposizione da Google.

NOTA

Questo è un grosso passo in avanti verso una maggiore semplificazione nella gestione degli aggiornamenti, ma, ovviamente, non potrà mai sostituire completamente la fase di aggiornamento legata agli aspetti puramente hardware.

Prima di procedere con la descrizione dei servizi messi a disposizione dai GPs, verifichiamo se il nostro dispositivo ne è provvisto. Utilizzando il Nexus 5 con Android 5.0 a nostra disposizione andiamo nella parte dei *Settings* relativa alle applicazioni e cerchiamo appunto i Google Play services, che notiamo essere presenti come visualizzato nella Figura 1.1 insieme ad altre app, sempre fornite da Google.
Se selezioniamo la voce corrispondente, otteniamo quanto rappresentato nella Figura 1.2, nella quale possiamo notare come la versione disponibile sia proprio la 6.5.

NOTA

A questo punto il lettore potrebbe chiedersi perché stiamo utilizzando un dispositivo che monta la versione di Android 5.0 senza utilizzare le API di questa nuova versione, che

sappiamo chiamarsi *Lollipop*. La prima ragione riguarda la scarsa distribuzione (almeno attualmente) di questa versione, mentre la versione 4.x è ormai compatibile con la quasi totalità dei dispositivi. Il secondo motivo riguarda la volontà di concentrarsi sui Google Play services e non su aspetti che riguardano *Lollipop*. Le API che descriveremo sono perfettamente funzionanti su tutti i dispositivi Android, dalla versione 2.3 in avanti e quindi anche sulle versioni 4.x e ovviamente 5.0.

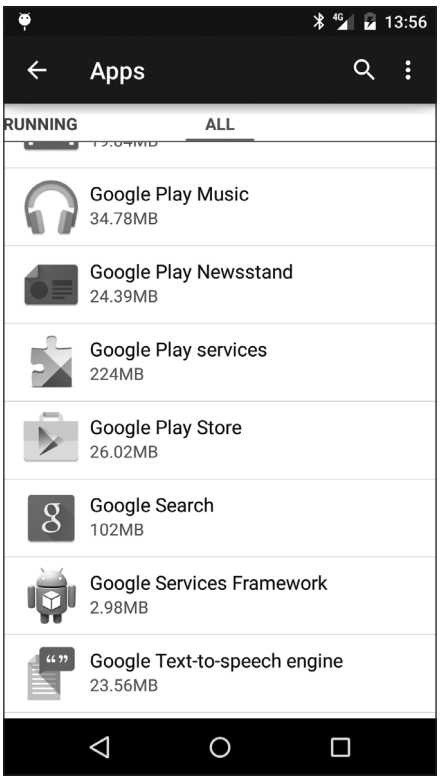

Figura 1.1 Applicazione Google Play services tra quelle disponibili.

Questo aggiornamento viene fatto in automatico, per cui non dovremo in alcun modo ricevere una notifica e procedere all'aggiornamento nel *Play Market*.
Osservando la Figura 1.2 viene la tentazione di selezionare il pulsante `Disable` come faremo più avanti per descrivere che cosa succede nel caso in cui l'applicazione non fosse disponibile. Al momento lasciamo al lettore la verifica, che porterebbe alla visualizzazione di un avvertimento e alla cancellazione dell'applicazione, con conseguente generazione di una serie di warning generati da tutte le applicazioni che fanno uso di questi servizi.

NOTA

Nel caso, niente paura: come vedremo più avanti, sarà sufficiente andare sul *Play Market* o semplicemente seguire le istruzioni successive alla selezione di una delle notifiche di warning per reinstallare il tutto.

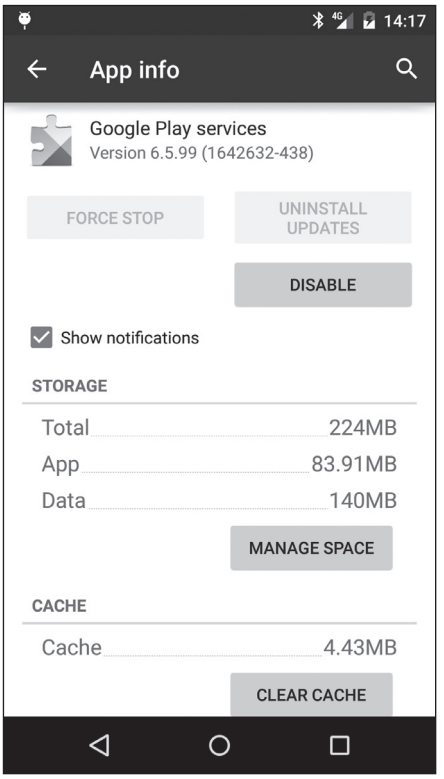

Figura 1.2 Le informazioni relative all'applicazione Google Play services nella versione 6.5.99.

Se andiamo a vedere le applicazioni installate di default sul dispositivo, notiamo anche quella con l'icona rappresentata nella Figura 1.3, relativa alle possibili configurazioni che ogni utente può personalizzare. Come vedremo nei prossimi capitoli, ogni utente potrà eventualmente disabilitare i servizi di localizzazione o gestire il proprio account Google Plus.

Figura 1.3 Icona per la configurazione dei Google Play services.

Si tratta di un'applicazione che viene mantenuta allineata con i Google Play services, permettendo la configurazione da parte degli utenti delle proprietà relative ai servizi che la piattaforma mette a disposizione.
Prima di iniziare a "sporcarci le mani" è comunque utile fare una panoramica di tutte le funzionalità dei Google Play services che andremo a utilizzare per realizzare un'applicazione che descriviamo brevemente e che amplieremo di capitolo in capitolo.
Si tratta, in sostanza, di un'applicazione che ci permetterà inizialmente di visualizzare su una mappa la posizione nostra e di un insieme di nostri amici che decideranno, dando ovviamente il loro consenso, di aggiungersi. In questo capitolo vedremo come creare lo scheletro dell'applicazione, mentre nei successivi vedremo come aggiungere funzionalità legate ai servizi di Google Play services che andremo di volta in volta a esplorare. Per completezza, ci aiuteremo anche con un insieme di altri esempi che ci permetteranno di testare funzionalità dei Google Play services non strettamente legate al nostro caso d'uso, ma che possono essere comunque importanti.

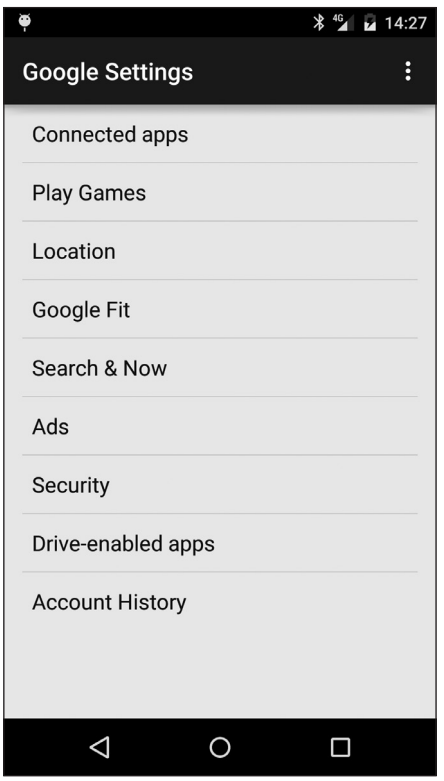

Figura 1.4 Elenco delle possibili configurazioni per i Google Play services.

Gestione della Location

Il primo passo consiste nell'individuare la posizione dei vari utenti. La determinazione della `Location` è stata una delle prime funzionalità trasferite a livello dei Google Play services, in quanto le informazioni fisiche necessarie, in termini di latitudine e longitudine, erano comunque già disponibili dalle prime versioni della piattaforma. Si è trattato, quindi, di implementare nuovi algoritmi in grado di elaborare queste informazioni e di dare maggiore accuratezza nella posizione nel caso di luoghi aperti e, soprattutto, di luoghi chiusi. Per quanto riguarda la nostra applicazione, svilupperemo l'opzione `Where I Am`, che permetterà di visualizzare le coordinate in termini di *latitudine* e *longitudine*, cui cercheremo di dare una località e quindi un nome. Sempre in corrispondenza di questa voce di menu non ci accontenteremo di sapere la nostra posizione, ma anche il modo in cui ci stiamo spostando, attraverso le funzionalità di `ActivityRecognition`.

> **NOTA**
>
> È curioso come il termine `Activity`, in questo caso, non faccia riferimento alla classe che utilizziamo come base per tutte le schermate, ma al concetto di attività ovvero al movimento dell'utente.

In questa fase vedremo tutto ciò che riguarda la posizione, rimandando la visualizzazione delle informazioni raccolte in una mappa al capitolo successivo. Si tratterà di un capitolo molto importante, in quanto ci permetterà di affrontare concetti anche non strettamente legati a quello di `Location`. Per realizzare queste funzionalità abbiamo scelto la strada migliore, ma anche meno convenzionale, non documentata in modo ufficiale, se non attraverso il *JavaDoc*. Da un punto vista puramente tecnico vedremo, in sintesi, come ottenere le informazioni di `Location` all'interno di un `Service` per poterle rendere persistenti in un `ContentProvider`. Ulteriore sfida è la possibilità di affrontare concetti di programmazione concorrente, non sempre descritti in altri testi.

Utilizzo delle Google Maps

Un'altra importantissima funzionalità fornita dai Google Play services è rappresentata dalla possibilità di visualizzare e gestire le Google Maps. Nel terzo capitolo vedremo come sia semplice visualizzare una mappa e quali siano i principali strumenti che ci permetteranno di evidenziare particolari `Location`. Vedremo che cosa siano i `Marker` e come sia semplice personalizzarli attraverso icone o colori e come gestire gli eventi associati attraverso la visualizzazione delle *Info Window*. Nel secondo capitolo abbiamo imparato a tracciare delle posizioni, che ora vogliamo visualizzare all'interno di una mappa sfruttando gli strumenti per il disegno di percorsi o di figure geometriche. Le stesse API per il disegno verranno utilizzate anche nella creazione e gestione dei `Geofence`, regioni nella mappa per le quali è possibile ricevere notifiche associate a particolari eventi. Alcuni di questi sono relativi alla semplice entrata o uscita da tali zone, ma possiamo creare una notifica nel caso un utente rimanesse nella stessa zona per un tempo maggiore di un periodo fissato. Questo caso d'uso ci permetterà anche di vedere come testare un servizio di localizzazione attraverso *Mock Location*. Concluderemo il terzo capitolo spiegando come utilizzare le *Street View* per visualizzare un percorso. Si tratta di un capitolo molto impegnativo, che ci permetterà di porre le basi per realizzare applicazioni con funzionalità molto interessanti.

Google Drive

Finora abbiamo visto come creare delle `FenceSession` ovvero una collezione di informazioni di `Location` e di attività caratterizzate da un nome, una data di inizio e di fine. Nel capitolo dedicato a Google Drive vogliamo utilizzare le Google Drive API per rendere queste informazioni persistenti all'interno di un file, per ripristinarle, eventualmente, in un secondo tempo.Vedremo in particolare come creare un file in Google Drive, gestendone il contenuto. Allo stesso modo vedremo come leggere questi file per importare informazioni di una specifica `FenceSession`.Vedremo come utilizzare questi strumenti per eseguire ricerche oltre che per gestire `Folder` privati della nostra applicazione (*App Folder*) o comunque accessibili anche in altro modo, per esempio tramite il Web.

Google Plus

Nel capitolo dedicato a Google Plus ci occuperemo dell'aspetto *social* della nostra applicazione, attraverso le API che i Google Play services ci mettono a disposizione per l'integrazione con Google Plus. Inizialmente vedremo come eseguire la login con Google+ in modo da ottenere le informazioni dell'utente senza doverle richiedere in modo esplicito. Poi vedremo come accedere alle informazioni relative agli utenti nelle proprie cerchie, per condividere informazioni di vario genere.

Google Cloud Messaging

Una delle tecnologie più affascinanti che sono state aggiunte ai Google Play services è quella che permette l'invio di notifiche *push* e che va sotto il nome di Google Cloud Messaging. Attraverso questa tecnologia è possibile inviare dei messaggi in *push* a un insieme di dispositivi che si sono in precedenza registrati a un server di terze parti che contiene la logica dell'applicazione. Nel nostro contesto il server potrebbe, per esempio, memorizzare le informazioni relative ai `Geofence` creati e quindi inviare notifiche ai nostri amici quando si è nella stessa zona. Le possibili applicazioni sono moltissime. Nel capitolo dedicato al Google Cloud Messaging descriveremo i punti fondamentali della tecnologia attraverso la realizzazione di una semplicissima chat. Come detto si tratta di una tecnologia che prevede un'architettura nel quale il server ha un'importanza fondamentale. Nel nostro caso abbiamo realizzato un semplicissimo server con *Node.js*, ma il lettore potrà realizzare il proprio con la tecnologia che ritiene più opportuna, purché vengano rispettate le convenzioni relative ai servizi `Rest` che il server espone. In particolare abbiamo realizzato un servizio `/register` per la registrazione del client di quello che si chiama `Registration Id` e che viene ottenuto dal server GCM di Google, e il servizio `/send` per l'invio di una notifica di *push* ai dispositivi associati a un particolare utente. Si tratta di una tecnologia che è possibile utilizzare in moltissimi modi diversi, secondo le specifiche caratteristiche dell'applicazione che intendiamo realizzare.

Google Wear

Nel capitolo dedicato a Google Wear ci occupiamo dei nuovi strumenti di gestione dei dispositivi *wearable*. Stiamo parlando di tutti quegli strumenti che possono essere

indossati e che permettono modalità di interazione diversa con le varie applicazioni. In questo momento i dispositivi *wearable* sono principalmente orologi, caratterizzati da display molto piccoli e modalità di interazioni limitate alla selezione di piccoli tasti o all'invio di comandi vocali. In questo capitolo vedremo tutto questo partendo dalla gestione e personalizzazione delle notifiche, fino alla realizzazione di applicazioni che girano interamente sul dispositivo *wearable*. Vedremo nel dettaglio tutti i componenti principali di una libreria di supporto specifica per questo tipo di dispositivi rilasciata da Google, concludendo con la descrizione delle diverse modalità con cui i diversi tipi di *device* comunicano tra loro, ovvero attraverso sincronizzazione di dati oppure messaggi. Si tratta di un capitolo abbastanza impegnativo, che ci permetterà di aprire un nuovo mondo nello sviluppo delle applicazioni Android.

Google Cast

Nel capitolo precedente abbiamo imparato a programmare un dispositivo esterno allo smartphone o al tablet che sta al polso come un orologio, ma ovviamente ci aspettiamo che altri dispositivi di vario genere, sempre *wearable*, vengano lanciati nel mercato e programmati allo stesso modo. In questo capitolo ci occupiamo invece di un altro genere di dispositivi esterni che si possono considerare come un'estensione dei *device* tradizionali per quello che riguarda la loro capacità di riprodurre media. Stiamo pensando per esempio a schermi molto grandi, come TV, oppure a dispositivi in grado di riprodurre suoni. In questo capitolo vedremo inizialmente come funzionano le MediaRouter API, per poi applicarle a un caso particolare ovvero l'utilizzo del Chromecast. Si tratta di un dispositivo dal prezzo molto accessibile, di poche decide di euro, che permette di riprodurre suoni o video su schermi diversi da quello dello smartphone e che solitamente sono molto più grandi e di ottima risoluzione. Vedremo che cos'è un `Sender` e soprattutto che come si programma un `Receiver` per questo tipo di dispositivi. È un argomento che si discosta leggermente da quelli trattati durante lo sviluppo dell'applicazione `FriendFence`, ma che permette comunque di realizzare un certo insieme di applicazioni molto interessanti.

Google Fit

Nel capitolo dedicato a Google Fit ci occuperemo invece di alcune API che sono state annunciate all'ultimo Google I/O come parte di Lollipop, ma che sono state integrate ai Google Play services nella release 6.5. Le Google Fit sono un ecosistema di API che permette di raccogliere informazioni relative alla nostra attività fisica attraverso dei sensori, per poi renderle persistenti all'interno di un database condiviso che si chiama Google Fitness Store. Si tratta di una tecnologia appena all'inizio e in piena evoluzione. Vedremo i concetti fondamentali e faremo alcuni esempi su come utilizzare le API per la raccolta, memorizzazione e visualizzazione dei dati.

Google Game

Nel capitolo dedicato a questo argomento ci occuperemo della descrizione delle Game API, ovvero degli strumenti che Google ci mette a disposizione per gestire l'aspetto *social* e non solo di un gioco. Vedremo infatti come creare e gestire degli obiettivi che

i giocatori dovranno raggiungere per poter sbloccare nuove funzionalità o comunque progredire nel gioco. Il raggiungimento di un obiettivo permette la raccolta di "punti esperienza", che possono essere poi visualizzati attraverso opportune classifiche (*leaderboard*) che impareremo a personalizzare e visualizzare. La seconda parte del capitolo è dedicata agli strumenti che permettono il *multiplayer* in modalità *real-time* e a turno. Vedremo gli strumenti per invitare altri giocatori attraverso i nostri amici nelle cerchie Google+ o attraverso la modalità *Quick Start*, che permette la selezione casuale di un giocatore in attesa di giocare. Vedremo come mettere in contatto questi giocatori e gestire lo scambio di messaggi. Vedremo poi come gestire i *Gift* e *Wish* e soprattutto come raccogliere informazioni sui giocatori del nostro gioco, in modo da tararlo di conseguenza. Il capitolo permette di capire, e soprattutto mettere in pratica, i concetti principali nello sviluppo di un gioco.

Google In-app Billing

Quindi ci occuperemo delle API che permettono la vendita di prodotti multimediali dalla nostra applicazione. Vedremo quali passi seguire per configurare l'applicazione e per creare il nostro catalogo. Poi ci occuperemo dell'integrazione di queste API all'interno della nostra applicazione. Vedremo come sia possibile ottenere informazioni sui prodotti del nostro catalogo, ma soprattutto come poterli acquistare. Faremo molta attenzione a quelli che sono gli aspetti di test, attraverso la creazione di una semplice applicazione.

Google Analytics

Segue un capitolo dedicato all'utilizzo di alcuni strumenti molto importanti per comprendere i nostri utenti e creare applicazioni che li soddisfino il più possibile. Vedremo come, attraverso le Google Analytics API sia possibile comprendere quali siano le schermate utilizzate dagli utenti e le modalità di navigazione. Vedremo come è possibile registrare tutte le azioni dell'utente, raggruppate per categorie, per poi visualizzarle attraverso la console di Google Analytics. Esamineremo inoltre gli strumenti disponibili per misurare le performance della nostra applicazione e per tenere traccia degli eventuali errori ed eccezioni.

Google Ads

L'ultimo capitolo è dedicato alle API per la visualizzazione dei banner all'interno della nostra applicazione. Vedremo inizialmente come creare normali banner, per poi passare ai cosiddetti *Interstitial*. Un capitolo molto breve che permette di farsi un'idea dei meccanismi di advertisement nelle applicazioni Android.

Installazione e setup di un'applicazione

Dopo aver descritto i servizi offerti dai GPs, iniziamo a creare il nostro progetto utilizzando Android Studio nella versione disponibile al momento, ovvero la 1.0. Per creare

lo scheletro della nostra applicazione utilizzeremo un semplice wizard, messo a disposizione dall'IDE.

NOTA

Anche se l'ambiente dovesse cambiare, il lettore potrà comunque creare la stessa struttura da zero seguendo la descrizione di seguito. Preferiamo non legarci troppo all'IDE e specialmente ad Android Studio, in continua evoluzione.

Iniziamo selezionando l'opzione di creazione di un nuovo progetto, ottenendo la schermata rappresentata nella Figura 1.5, che ci mette a disposizione diverse opzioni.

NOTA

Con le ultime versioni di Android Studio è possibile che venga visualizzata una finestra diversa, che permette di selezionare una configurazione standard o personalizzata dell'IDE. In questo caso, basta selezionare il pulsante *Cancel* per arrivare alla schermata per la creazione di un nuovo progetto.

In questa prima schermata inseriamo semplicemente il nome dell'applicazione, il corrispondente *package* e il percorso in cui salvare il progetto nel nostro file system.

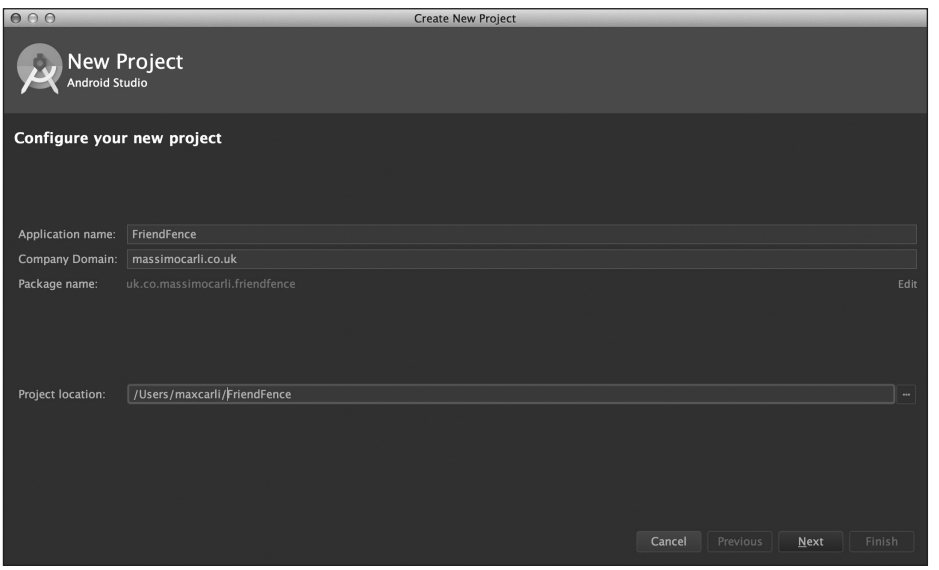

Figura 1.5 Finestra di creazione del progetto FriendFence.

Ovviamente il lettore potrà avere un percorso diverso.

NOTA

Supponiamo che il lettore sia già a conoscenza dei concetti base della programmazione Android. Descriveremo solamente gli aspetti in qualche modo collegati ai GPs.

Fino a qui nessun problema, se non quello di sottolineare come il *package* sia di fondamentale importanza, in quanto definisce in modo univoco la nostra applicazione e non potrà più essere modificato una volta che questa sia stata pubblicata.

NOTA

Forse per questo motivo viene chiesto il dominio dell'azienda o persona che crea l'applicazione e quindi viene creato in automatico il *package*. L'errore in questa fase è abbastanza frequente.

Selezionando il pulsante `Next` otteniamo la schermata rappresentata nella Figura 1.6 che, alla luce delle novità annunciate al Google I/O del 2014, assume un'importanza fondamentale.

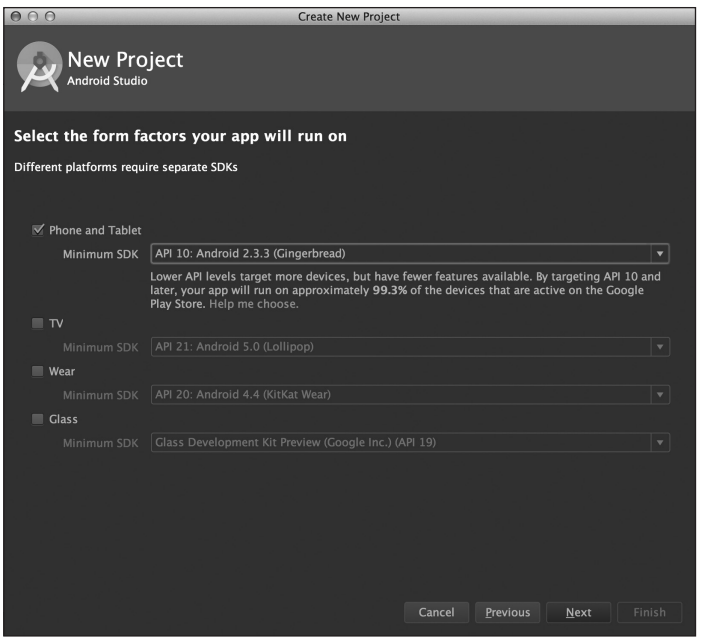

Figura 1.6 Definizione degli ambienti in cui l'applicazione potrà essere eseguita e relative versioni.

In questa fase andiamo infatti a definire gli ambienti in cui la nostra applicazione potrà essere eseguita, con relative versioni minime. Come possiamo notare ci sarà la possibilità di eseguire l'applicazione su un ambiente TV, su un dispositivo `Wear` che al momento è rappresentato da un orologio oppure attraverso i Google Glasses. Al momento selezioniamo solamente la prima delle opzioni, scegliendo come minima versione di riferimento quella relativa a *Gingerbread* ovvero un API Level 10 o SDK di versione 2.3.3. Come possiamo vedere nella figura, questo ci permette di raggiungere il 99,2% degli utenti che possiamo considerare più che accettabile.

NOTA

In realtà i Google Play services esistono anche per *Froyo* (API Level 8), ma sono contenuti in una libreria distinta. L'aggiunta di questa versione avrebbe comportato l'introduzione di una complessità non giustificata da un aumento minimo della percentuale di dispositivi (0,8%) peraltro in continua diminuzione.

A questo punto facciamo clic sul pulsante `Next` e otteniamo un'altra schermata che ci permette di scegliere tra un insieme di `Activity` (Figura 1.7). Il lettore potrà notare come esista la possibilità di creare una Google Play services Activity e generare in modo automatico tutto il codice di inizializzazione. Nel nostro caso abbiamo deciso di essere il più possibile indipendenti dall'IDE, per cui scegliamo l'opzione evidenziata in figura: `Blank Activity`.

NOTA

Come sappiamo, l'utilizzo dei *Fragment* è da preferire in quanto ci permette di essere più elastici nel caso di un *Tablet*. Nel nostro caso la prima attività da creare sarà quella di *Splash* che non necessita di questo accorgimento. È comunque disponibile anche una *Blank Activity with Fragment*.

Selezioniamo ancora il pulsante `Next`, arrivando alla schermata rappresentata nella Figura 1.8, la quale ci permette di inserire le informazioni relative alla nostra `SplashActivity`.
A questo punto il pulsante `Finish` è abilitato e possiamo procedere con la creazione del progetto e la conseguente generazione della struttura, che possiamo vedere in Figura 1.9. Coloro che hanno già utilizzato una versione precedente di Android Studio troveranno una modalità di visualizzazione diversa. In questa nuova vista possiamo vedere come la definizione del modulo *app* è separata dai file di configurazione di `Gradle`, che possiamo vedere nella parte inferiore.

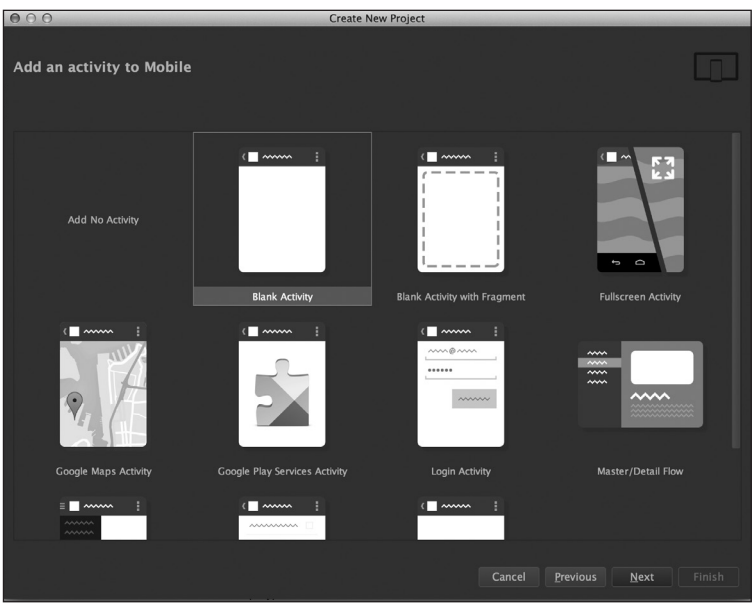

Figura 1.7 Wizard per la creazione di una Activity.

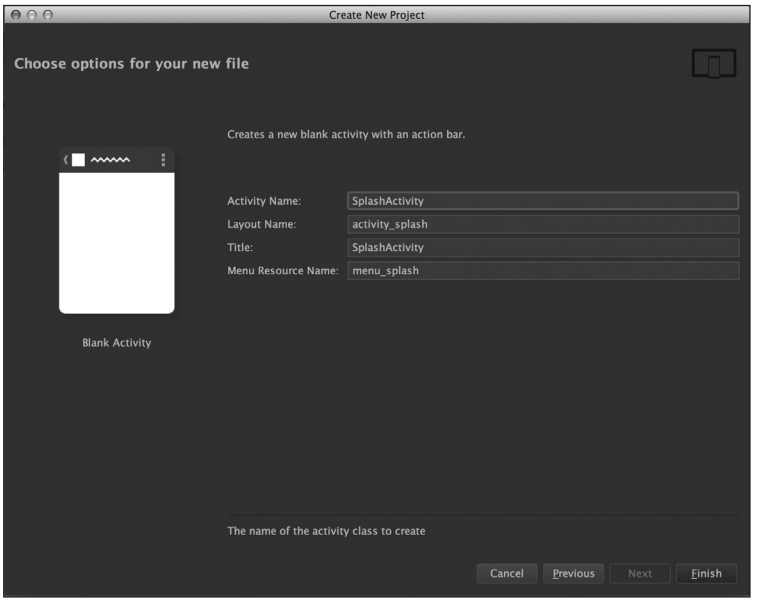

Figura 1.8 Creazione della nostra SplashActivity.

Nel modulo *app* notiamo come il file di configurazione `AndroidManifest.xml` venga visualizzato in una parte separata rispetto al codice. Infine notiamo come le risorse con qualificatore diverso vengano comunque visualizzate come se appartenessero a una stessa cartella; è il caso delle risorse nei file `dimens.xml`.

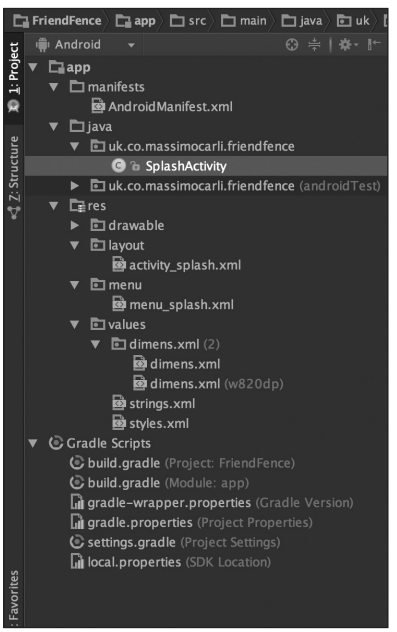

Figura 1.9 Struttura del progetto generata in automatico da Android Studio.

Prima di descrivere tutto ciò che è stato generato, dobbiamo però assicurarci un paio di cose. La prima riguarda la disponibilità di tutte le librerie di cui andremo a definire le dipendenze nel file di configurazione di `Gradle` ovvero il file `build.gradle` associato al modulo app. Per fare questo andiamo a selezionare l'icona per l'avvio dell'*SDK Manager* (Figura 1.10) per verificare la presenza delle librerie relative ai Google Play services, della libreria di supporto e dei vari *Repository* cui `Gradle` accede in fase di build.

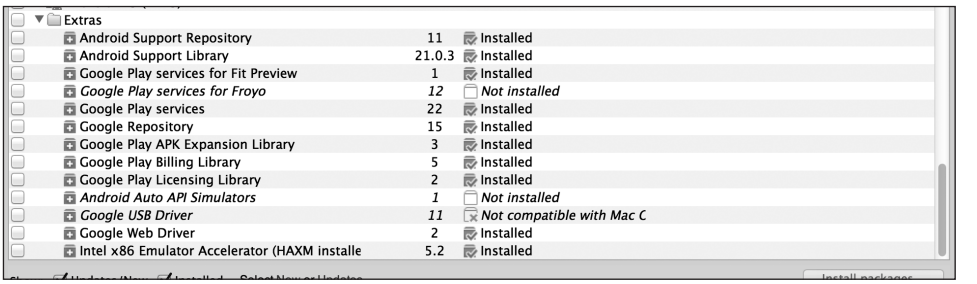

Figura 1.10 Verifichiamo la presenza delle librerie necessarie per la nostra applicazione.

> **NOTA**
>
> Notiamo come la versione dei Google Play services 6.5 disponibile al momento corrisponda alla revisione 22. Qualora Google rilasciasse nuove versioni, tale valore sarà ovviamente diverso.

seconda riguarda l'utilizzo delle librerie in `Gradle` oltre alla configurazione dei Google Play services, come evidenziato nel seguente listato:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 21
    buildToolsVersion "21.1.2"
    defaultConfig {
        applicationId "uk.co.massimocarli.friendfence"
        minSdkVersion 10
        targetSdkVersion 21
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                    'proguard-rules.pro'
        }
    }
}
```

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
    compile 'com.android.support:support-v4:21.+'
    compile 'com.google.android.gms:play-services:6.5.+'
}
```

Ovviamente la versione cui fare riferimento sarà quella disponibile al momento di creazione del progetto. Come possiamo vedere abbiamo aggiunto la dipendenza dalle librerie di supporto e ovviamente dai Google Play services, che rappresenta il primo passo di configurazione per il loro utilizzo. Nel precedente codice abbiamo evidenziato la definizione della seguente dipendenza, che permette di importare nel nostro progetto tutte le API contenute all'interno dei Google Play services:

```
compile 'com.google.android.gms:play-services:6.5.+'
```

In realtà questo non è sempre necessario. Se realizziamo un'applicazione che utilizza le Location API, forse non abbiamo bisogno anche delle API per la gestione del *gaming* o dei dispositivi *wearable*. In realtà per la maggior parte delle applicazioni questo non è un problema. Per applicazioni di una certa dimensione può invece accadere che si verifichi un errore in fase di *build* dovuto al fatto che esiste comunque un limite nel numero dei metodi (framework compreso) del codice contenuto all'interno di un APK. Non possiamo, infatti, avere un numero di metodi superiore a *65.536*. Per questo motivo, dalla versione 6.5, è possibile importare anche solo le API che servono. Nel caso della nostra applicazione vedremo di volta in volta quali API utilizzare e quali dipendenze impostare. Facciamo quindi subito una prima modifica, sostituendo la definizione precedente con :

```
compile 'com.google.android.gms:play-services-base:6.5.87'
```

Questo ci permette di importare le sole API base. Un'altra modifica riguarda l'eliminazione del + con un valore esplicito di versione. La parte delle dipendenze diventa quindi la seguente. Ricordiamo che le versioni potrebbero essere diverse per il lettore, che rimandiamo alla documentazione ufficiale, dove troverà il valore corretto.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.3'
    compile 'com.android.support:support-v4:21.0.3'
    compile 'com.google.android.gms:play-services:6.5.87'
    compile 'com.google.android.gms:play-services-maps:6.5.87'
}
```

L'applicazione generata in modo automatico permette la semplice visualizzazione di un messaggio `HelloWorld`, mentre nel nostro caso vogliamo realizzare una `Splash`, ma soprattutto introdurre la logica che ci permetterà di fare in modo che gli stessi Google Play services siano disponibili nella versione corretta.
Partiamo da quanto generato in modo automatico e iniziamo con la dichiarazione dei Google Play services nel file di configurazione della nostra applicazione, ovvero all'interno dell'`AndroidManifest.xml`. Avendo utilizzato il *template* relativo a una *Blank Activity*, questa

definizione non è avvenuta in automatico, per cui dovremo apportare manualmente la modifica evidenziata nel seguente documento:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="uk.co.massimocarli.friendfence">

    <application
            android:allowBackup="true"
            android:icon="@drawable/ic_launcher"
            android:label="@string/app_name"
            android:theme="@style/AppTheme">
    <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version"/>
        <activity
                android:name=".SplashActivity"
                android:theme="@style/FullscreenTheme"
                android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

In realtà l'operazione è molto semplice, in quanto il riferimento della versione dei Google Play services è verso una risorsa di tipo intero, definita nella stessa libreria importata in precedenza. Si tratta quindi di un valore che viene modificato automaticamente nel momento in cui viene aggiornata la versione della libreria dei Google Play services utilizzata.

NOTA

Questo significa che il valore non dovrà più essere modificato in caso di aggiornamenti dei Google Play services.

Nel codice abbiamo evidenziato anche l'utilizzo di un tema che ci permette di visualizzare l'`Activity` di `Splash` a tutto schermo, definita nel seguente modo nel file `styles.xml`. D'ora in poi non ci occuperemo più di questi dettagli, che il lettore potrà comunque cogliere nel codice relativo agli esempi.

```
    <style name="FullscreenTheme" parent="android:Theme.NoTitleBar.Fullscreen"/>
```

L'ultimo passo di configurazione riguarda *Proguard*, il tool di offuscamento e ottimizzazione del codice, che viene richiamato nel caso di creazione dell'*APK* firmato con il certificato di produzione. È infatti necessario fare in modo che le nuove classi non vengano offuscate e quindi rese inutilizzabili dall'applicazione. Anche in questo caso si tratta di aggiungere il seguente testo al file `proguard-rules.pro` nella cartella associata al modulo del progetto. Per i dettagli relativi a questa configurazione rimandiamo alla documentazione ufficiale di *Proguard* (`http://proguard.sourceforge.net/`).

```
-keep class * extends java.util.ListResourceBundle {
    protected Object[][] getContents();
}

-keep public class com.google.android.gms.common.internal.safeparcel.SafeParcelable {
    public static final *** NULL;
}

-keepnames @com.google.android.gms.common.annotation.KeepName class *
-keepclassmembernames class * {
    @com.google.android.gms.common.annotation.KeepName *;
}

-keepnames class * implements android.os.Parcelable {
    public static final ** CREATOR;
}
```

Per creare questo file è possibile passare alla visualizzazione del progetto, selezionando la corrispondente opzione (Figura 1.11).

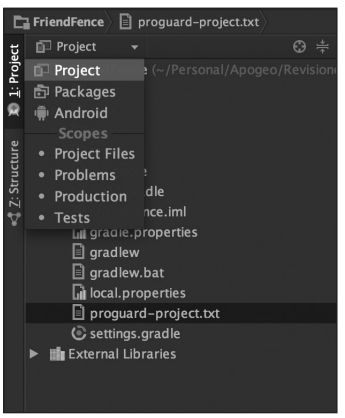

Figura 1.11 Vista di progetto per la visualizzazione del file di Proguard.

Terminata la fase di configurazione iniziamo quella relativa al codice Java da aggiungere alla nostra `SplashActivity`.Vogliamo infatti che venga visualizzato il nome dell'applicazione e che dopo un certo tempo si proceda verso la `Activity` principale, che al momento non abbiamo ancora preparato. Per inserire la logica relativa alla gestione dei Google Play services facciamo pulizia partendo dal seguente codice:

```
public class SplashActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
    }
}
```

Ecco il documento di *layout* nel file `activity_splash.xml`:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#0099cc"
        tools:context=".SplashActivity">

    <TextView
            android:id="@+id/fullscreen_content"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:keepScreenOn="true"
            android:textColor="#33b5e5"
            android:textStyle="bold"
            android:textSize="50sp"
            android:gravity="center"
            android:text="@string/app_name"/>

</FrameLayout>
```

Per eliminare la `ActionBar` abbiamo anche dovuto modificare le risorse relative allo stile per i dispositivi di API Level maggiore o uguale a 11 nel file `/res/values-v11/styles.xml`:

```
<resources>
    <style name="FullscreenTheme"
            parent="android:Theme.Holo.NoActionBar.Fullscreen"/>
</resources>
```

E anche nel file `/res/values/styles.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
    </style>
    <style name="FullscreenTheme" parent="android:Theme.NoTitleBar.Fullscreen">
    </style>
</resources>
```

L'utilizzo di queste risorse di tipo `style` sarebbe causa di un errore nel caso in cui la nostra `Activity` estendesse la classe `ActionBarActivity` della relativa libreria come nel codice generato automaticamente da Android Studio. La nostra Splash non necessita di `ActionBar`, per cui abbiamo semplicemente esteso la classe `Activity` come evidenziato nel precedente codice. Il tema dovrà quindi essere associato alla `SplashActivity`, come evidenziato nell'`AndroidManifest.xml`. Se eseguissimo l'applicazione in questo momento otterremmo quanto rappresentato nella Figura 1.12, che non è nulla di speciale, ma che ci permette di consolidare un punto di partenza.

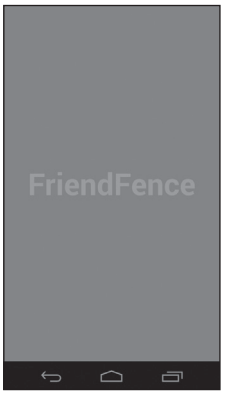

Figura 1.12 La visualizzazione della SplashActivity come punto di partenza.

Non ci resta che fare in modo che un utente non possa avviare la nostra applicazione se non con la versione corretta dei Google Play services. Si tratta di un procedimento ormai standard, per cui le stesse API ci mettono a disposizione dei metodi di utilità per raggiungere il nostro scopo in modo semplice.

> **NOTA**
>
> Nel nostro caso eseguiamo questo controllo all'avvio dell'applicazione, in quanto i Google Play services sono necessari da subito. Nel caso si trattasse di funzionalità secondarie è possibile implementare la logica di controllo in un altro punto dell'applicazione.

Il codice della nostra `SplashActivity` prevede sostanzialmente la gestione del controllo sulla compatibilità (presenza o versione) dei Google Play services e, in caso di successo, il passaggio automatico alla `Activity` principale, che abbiamo chiamato `MainActivity` e che vedremo come creare utilizzando un *Wizard* di Android Studio.

```
public class SplashActivity extends Activity {
    /**
    * The Tag for the Log
    */
    private static final String TAG_LOG = SplashActivity.class.getName();

    /**
    * The delay to wait before going to the MainActivity
    */
    private static final long SPLASH_DELAY = 2000;

    /**
    * The Request code to use for the Dialog management
    */
    private static final int GPS_REQUEST_CODE = 1;

    /**
    * The Handler to manage the delay message
    */
```

```
private final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        if (!isFinishing()) {
            final Intent mainIntent =
                    new Intent(SplashActivity.this, MainActivity.class);
            startActivity(mainIntent);
            finish();
        }
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_splash);
}

@Override
protected void onResume() {
    super.onResume();
    // We check if the GooglePlayServices are available
    int resultCode =
              GooglePlayServicesUtil.isGooglePlayServicesAvailable(this);
    if (ConnectionResult.SUCCESS != resultCode) {
        // In this case the Google Play services are not installed or
        // in the wrong version so we have to launch the Play Store
        // for the installation
        GooglePlayServicesUtil.showErrorDialogFragment(resultCode,
                this, GPS_REQUEST_CODE, new DialogInterface.OnCancelListener() {
            @Override
            public void onCancel(DialogInterface dialog) {
                // In this case we close the app
                Toast.makeText(SplashActivity.this, R.string.gps_mandatory,
                        Toast.LENGTH_LONG).show();
                finish();
            }
        });
    } else {
        // Here we implement the logic for the MainActivity
        mHandler.sendEmptyMessageDelayed(0, SPLASH_DELAY);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (GPS_REQUEST_CODE == requestCode &&
            Activity.RESULT_CANCELED == resultCode) {
        Log.d(TAG_LOG, "Dialog closed from the Play Market");
```

```
            Toast.makeText(SplashActivity.this, R.string.gps_mandatory,
                    Toast.LENGTH_LONG).show();
        }
    }
}
```

Il primo controllo è molto semplice e utilizza il seguente metodo statico della classe `GooglePlayServicesUtil`:

```
public static int isGooglePlayServicesAvailable (Context context)
```

Questo restituisce un valore intero che indica la presenza o meno dei Google Play services nella versione corretta. Si tratta di un valore corrispondente a una delle seguenti costanti esplicative della classe `ConnectionResult`:

```
SUCCESS
SERVICE_MISSING
SERVICE_VERSION_UPDATE_REQUIRED
SERVICE_DISABLED
SERVICE_INVALID
DATE_INVALID
```

Nel nostro caso controlliamo se il codice restituito è diverso da `SUCCESS`. Fortunatamente non è necessario gestire ogni singolo caso, in quanto la stessa classe di utilità mette a disposizione il seguente metodo:

```
public static boolean showErrorDialogFragment (int errorCode, Activity activity,
        int requestCode, DialogInterface.OnCancelListener cancelListener)
```

Questo permette di creare una `DialogFragment` con un messaggio corrispondente al problema da risolvere e soprattutto con la soluzione del problema. I parametri di questo metodo meritano un piccolo approfondimento. Il parametro `errorCode` è il valore ottenuto in precedenza, corrispondente a una delle precedenti costanti. Il parametro `activity` è l'`Activity` che ospiterà il `Fragment`, che nel nostro caso è la `SplashActivity`. Il parametro `requestCode` è il valore che verrà utilizzato in corrispondenza della chiamata del metodo `onActivityResult()` che nel nostro caso ha un'implementazione che prevede la semplice visualizzazione, attraverso un `Toast`, di un messaggio che indichi come i Google Play services siano necessari all'esecuzione dell'applicazione, per poi chiuderla attraverso la chiamata del metodo `finish()`. Il quarto e ultimo parametro è il riferimento all'implementazione dell'interfaccia `DialogInterface.OnCancelListener`, che permette di gestire il caso in cui l'utente prema `Back` dopo la visualizzazione della `Dialog`.

NOTA

Il metodo `showErrorDialogFragment()` prevede anche un *overload* senza la necessità di impostare un valore per il parametro `cancelListener`.

È importante comprendere come la cancellazione della `Dialog` comporti la chiamata del particolare `cancelListener`, ma non la chiamata del metodo `onActivityResult()`, il quale

viene invece richiamato nel caso in cui l'operazione venisse annullata direttamente dalla schermata del *Play Market* in fase di installazione dei Google Play services.
Il secondo aspetto della nostra `SplashActivity` riguarda il passaggio alla `MainActivity` nel caso in cui il test sui Google Play services sia andato bene e siano passati 2 secondi. Come possiamo notare, questo viene realizzato attraverso un `Handler` che ha, come unica accortezza, quella di controllare attraverso il metodo `isFinishing()`, se l'`Activity` è stata chiusa.

NOTA

Vista la semplicità della nostra *Splash* abbiamo deciso di non implementare l'*Handler* con il consueto accorgimento della classe interna statica, al fine di eliminare possibili *memory leak*.

Prima di passare a un semplice test sul funzionamento di quanto creato ci ricordiamo di creare una `MainActivity` temporanea.

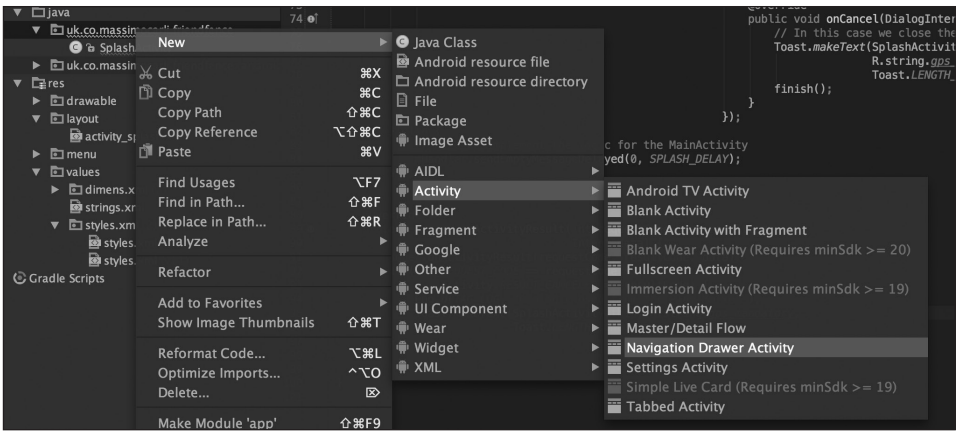

Figura 1.13 Creazione di una Activity di tipo Navigation Drawer.

Abbiamo deciso di utilizzare un `Navigation Drawer` per la selezione delle diverse opzioni, per cui utilizziamo il Wizard che Android Studio ci mette a disposizione. Per mettere un minimo di ordine abbiamo creato un *package activity* e quindi selezionato l'opzione indicata nella Figura 1.13, che ci porta alla finestra rappresentata nella Figura 1.14.
La presenza della classe `MainActivity` permetterà la compilazione della nostra `SplashActivity`. Eseguendo l'applicazione otterremmo la visualizzazione della `Splash` della Figura 1.12 e quanto rappresentato nella Figura 1.15 che andremo a personalizzare nei prossimi capitoli. Quanto visualizzato nella Figura 1.15 potrà essere diverso da quanto visualizzato dal lettore, a seconda della particolare versione di Android utilizzata o del particolare dispositivo.
Quello descritto è il funzionamento nel caso in cui tutto vada bene ovvero che nel device sia disponibile l'ultima versione di Google Play services. Il prossimo paragrafo ci permetterà di verificare cosa succede nel caso in cui i Google Play services non fossero invece disponibili o siano di versione diversa da quella richiesta.

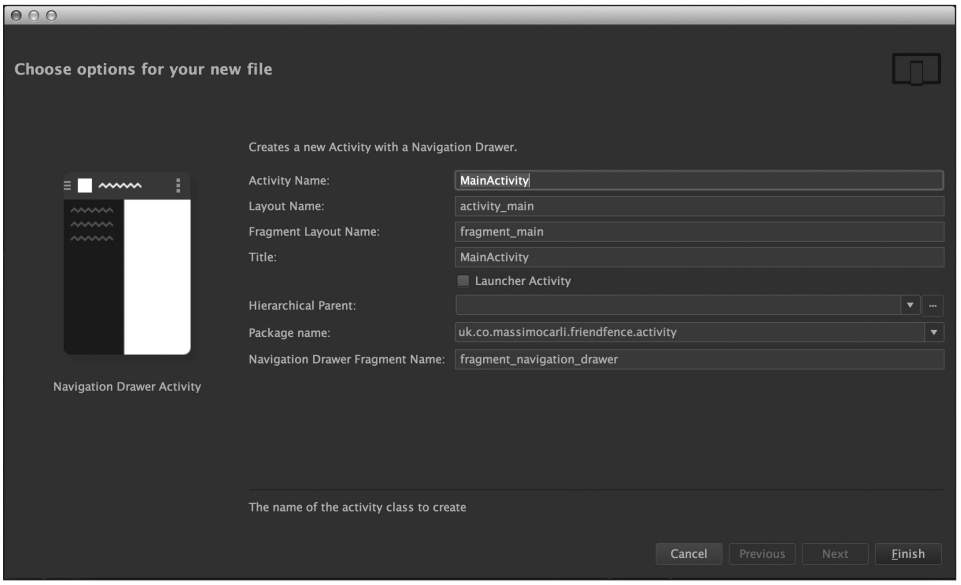

Figura 1.14 Configurazione della MainActivity.

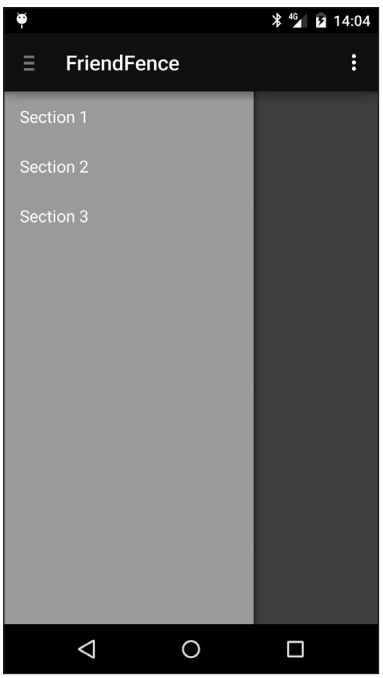

Figura 1.15 La MainActivity in esecuzione.

Aggiornamento dei Google Play services

In questo capitolo non abbiamo scritto moltissimo codice, ma abbiamo implementato qualcosa di necessario in tutte le applicazioni che si basano sui Google Play services. Per completezza è interessante osservare che cosa succederebbe qualora eseguissimo la nostra applicazione in un dispositivo non ancora dotato di Google Play services.

NOTA

Il lettore non interessato a questo tipo di test può saltare al prossimo capitolo, impiegando quanto realizzato in precedenza.

La maggior parte dei dispositivi (approvati da Google) dovrebbe disporre della ultima versione dei Google Play services. In caso contrario, la nostra applicazione dovrebbe chiedere all'utente di procedere alla loro installazione, o aggiornamento, dal *Play Market*.

NOTA

Nella descrizione che segue abbiamo utilizzato un Nexus 5. In altri dispositivi questo meccanismo potrebbe essere leggermente diverso e potrebbe comportare la perdita di alcune informazioni. Lasciamo quindi al lettore la piena responsabilità su quanto segue.

Cerchiamo di simulare questo processo andando nella schermata rappresentata nella Figura 1.2 e quindi selezionando il pulsante `Disable`. A questo punto si ha la visualizzazione del messaggio rappresentato nella Figura 1.16: i dati memorizzati fino a quel momento verranno cancellati.

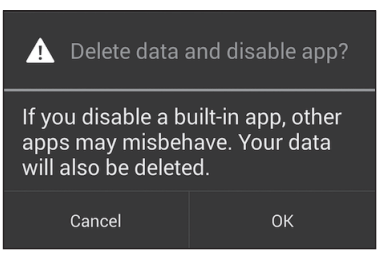

Figura 1.16 La disabilitazione dei GPs comporta la perdita dei dati corrispondenti.

Selezioniamo il pulsante `OK` e otteniamo una finestra di dialogo (Figura 1.17) nella quale ci viene chiesto di ripristinare l'applicazione nella sua versione originale, ovvero quella installata inizialmente nel dispositivo.
Anche in questo caso selezioniamo il pulsante `OK`: un messaggio ci segnala che l'operazione che stiamo per effettuare necessita dei diritti di *Administrator* del device (Figura 1.18).
Se siamo sicuri della nostra scelta selezioniamo il pulsante *Manage device administrators* il quale ci conduce alla schermata rappresentata nella Figura 1.19 che contiene una *checkbox* che deselezioniamo arrivando alla schermata rappresentata nella Figura 1.20.

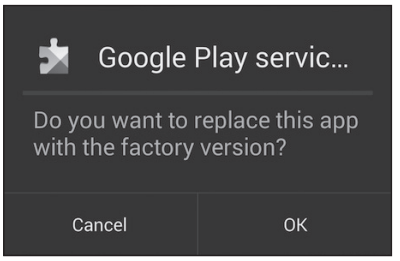

Figura 1.17 Ripristiniamo la versione iniziale.

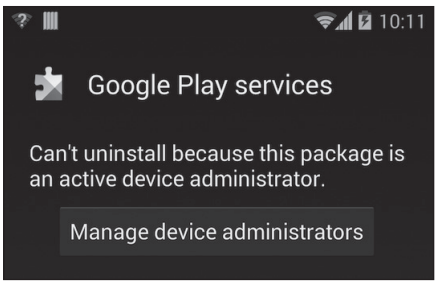

Figura 1.18 La rimozione necessita dei diritti di Administrator.

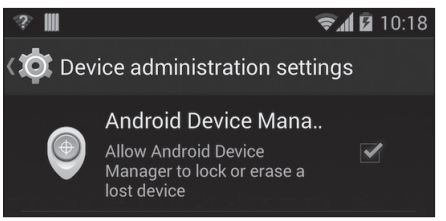

Figura 1.19 Otteniamo il diritto di cancellare la nostra applicazione.

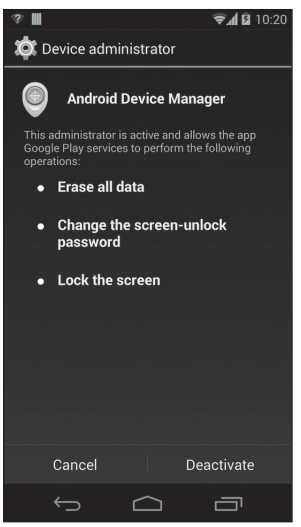

Figura 1.20 In questa schermata possiamo disattivare il ruolo di Administrator.

Selezionando il pulsante `Deactivate` in basso a destra torniamo alla schermata rappresentata nella Figura 1.2, ma questa volta il pulsante `Uninstall updates` è abilitato, come possiamo vedere nella Figura 1.21.

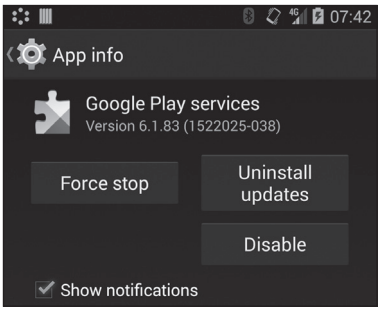

Figura 1.21 Ora il pulsante Uninstall updates è abilitato.

Non ci resta che selezionare il pulsante `Uninstall updates` e dare conferma nella successiva finestra di dialogo, per procedere alla disinstallazione, seguita da una serie di notifiche di "lamentela" da parte delle altre applicazioni, tra cui, nel nostro caso, *Hangout* e Google+ (Figura 1.22).

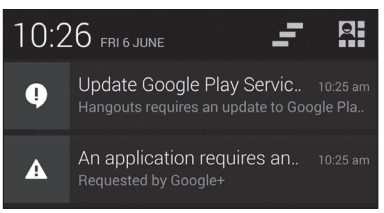

Figura 1.22 Notifiche delle applicazioni che reclamano la presenza dell'ultima versione di GPs.

Selezionando una di queste notifiche sarà possibile andare al *Play Store* e ottenere quello che invece noi vogliamo testare attraverso la nostra applicazione, che avviamo ottenendo il messaggio rappresentato nella Figura 1.23.

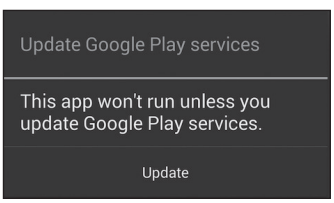

Figura 1.23 La nostra applicazione, giustamente, richiede l'ultima versione di GPs.

A questo punto l'utente potrebbe premere il pulsante `Back`, in quanto non intendere procedere all'aggiornamento. Questa è la parte del caso d'uso gestito dalla nostra implementazione dell'interfaccia `DialogInterface.OnCancelListener` la quale termina l'applicazione visualizzazione un messaggio attraverso un *Toast* (Figura 1.24). In questo caso il metodo di `callback onActivityResult()` non viene richiamato.

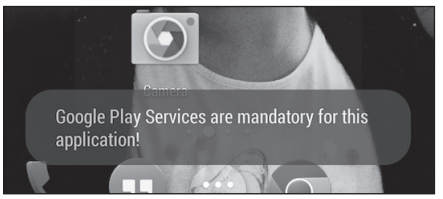

Figura 1.24 Se l'utente rifiuta l'aggiornamento dobbiamo uscire dall'applicazione.

Se invece selezioniamo il pulsante `Update` veniamo condotti al *Play Market*, dove potremo procedere all'aggiornamento. Qualora, una volta giunti al *Play Market*, decidessimo di rifiutare l'aggiornamento scegliendo ancora il pulsante `Back`, cadremmo nel caso corrispondente alla cancellazione della operazione.Verrà richiamato il metodo `onActivityResult()` con un valore di `resultCode` di cancellazione, ovvero `Activity.RESULT_CANCELED`. Nel nostro caso il risultato è lo stesso del caso precedente e quindi la visualizzazione di un *Toast* con il messaggio rappresentato nella Figura 1.24. Questo metodo permetterebbe anche la gestione di altri tipi di problemi che si potrebbero verificare di ritorno dalla particolare `Activity` di destinazione della *Dialog,* che non è detto sia necessariamente quella del *Play Store*.
Procedendo invece all'aggiornamento dei Google Play services e tornando alla nostra applicazione noteremo come il funzionamento sia quello sperato, ovvero la visualizzazione della `MainActivity` dopo il tempo di attesa impostato nella `SplashActivity`.
Al termine consigliamo il lettore di riattivare i vincoli legati al ruolo di *Administrator* del dispositivo, attraverso la corrispondente opzione che nel Nexus 5 è nei `Settings` alla voce `Security` e quindi `Device Administrators`.

Conclusioni

Nella prima parte di questo capitolo abbiamo voluto giustificare la realizzazione dei Google Play services come un modo per diminuire la forte dipendenza della piattaforma dai diversi produttori di dispositivi. Attraverso un'accorta separazione tra quello che è fortemente legato all'hardware e quello che è invece gestibile a livello software è stato possibile creare un meccanismo di aggiornamento della piattaforma più affidabile e quindi in grado di soddisfare maggiormente i diversi utenti oltre che Google stessa.
Dopo aver dato una panoramica su quelli che saranno i servizi offerti dai Google Play services abbiamo iniziato lo sviluppo dell'applicazione `FriendFence`, occupandoci di quello che ogni applicazione di questo tipo dovrà fare ovvero verificare la presenza dei Google

Play services nella corretta versione, gestendone eventualmente l'aggiornamento attraverso l'accesso al *Play Market*. Abbiamo quindi concluso attraverso la simulazione di quello che accadrebbe nel caso in cui l'utente non disponesse dei Google Play services necessari alla nostra applicazione. Quello ottenuto alla fine di questo capitolo è lo scheletro di un'applicazione, con alcune voci di menu che andremo a modificare e riempire di volta in volta nei prossimi capitoli.