Launcher: funzioni native e pubblicazione negli store

Indice generale

Launcher: funzioni native e pubblicazione negli store

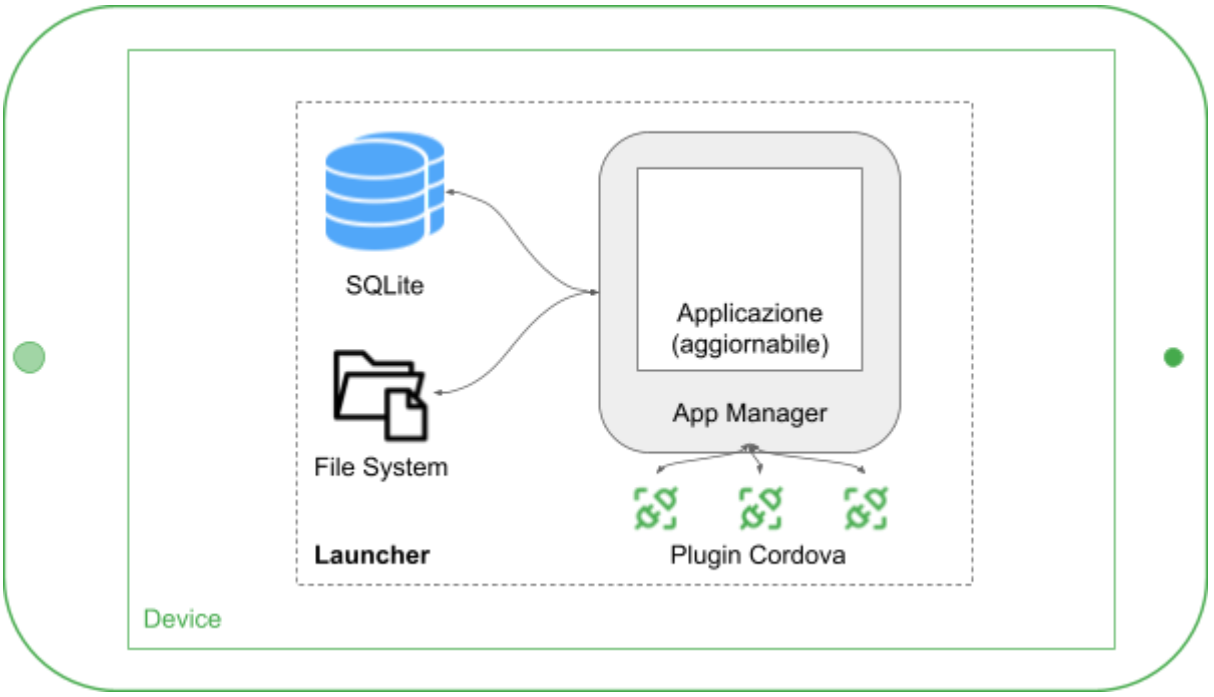
Integra le funzioni native nelle tue app. Pubblica sugli store. Aggiorna l’app in real time.

Introduzione

Nei capitoli precedenti abbiamo visto come sviluppare tutte le componenti dell’applicazione anche in modalità offline, cioè completamente funzionante in un dispositivo o un browser. Ora vedremo come fare il passaggio successivo: inserire l’applicazione sviluppata in un container nativo per utilizzare le varie parti del dispositivo e poter poi inviare l’applicazione completa ad App Store e Google Play. Cominciamo con l’analisi dell’architettura del container nativo di Instant Developer Cloud: il *Launcher*.

Il Launcher

Un *Launcher* è un'applicazione nativa compilabile per dispositivi iOS e Android ed è basato sull’architettura Apache Cordova per la creazione di applicazioni mobile ibride.



Quando l’applicazione viene compilata in versione per launcher, essa integra un device virtuale compatibile con quello del contenitore per applicazioni web: lo stesso codice applicativo viene eseguito con la medesima semantica, fatto salvo il diverso ambiente operativo su cui esso avviene, cioè una *webview* del dispositivo invece che un processo *node.js*.

Questo isomorfismo si esplicita in tre ambiti principali: accesso al database, uso del file system e accesso alle funzioni native del dispositivo.

Quando l'applicazione è in esecuzione nel cloud, le query sui database relazionali vengono eseguite rispetto al database Postgres. In questo caso invece le stesse query vengono eseguite sul database SQLite del dispositivo e il codice SQL viene automaticamente tradotto da Instant Developer Cloud, se la query è stata inserita in modo *strutturato*.

Nel launcher è inoltre disponibile un file system compatibile con quello del cloud; infine l'applicazione può accedere ai vari plugin Cordova tramite delle classi JavaScript di interfaccia. Nelle librerie standard sono inclusi i plugin più usati, ma è possibile aggiungere i propri plugin e renderli disponibili all'applicazione sviluppando la relativa interfaccia.

L'applicazione vera e propria viene compilata dalla console attraverso l'unione di una build e di un launcher, che viene configurato tramite la console di Instant Developer Cloud. A questo punto, sempre tramite la console, è possibile generare il progetto Apache Cordova complessivo per effettuare dei test, compilare l'applicazione in formato nativo ed infine inviare l'applicazione ad App Store e Google Play.

L'applicazione è aggiornabile in tempo reale: dopo la prima installazione è possibile sostituire il codice applicativo senza dover nuovamente sottoporre l'applicazione agli store. Questa funzionalità viene gestita sempre tramite la console e permette di distribuire fix e piccoli miglioramenti che non richiedono cambiamenti alla struttura del launcher (plugin nativi, splash screen, dati di configurazione).

Tipi di Launcher

Instant Developer Cloud mette a disposizione cinque tipi di launcher, due per i test e tre per la produzione. Vediamoli in dettaglio:

InstaLauncher

È un launcher già compilato e disponibile negli app store che permette di testare le proprie applicazioni direttamente dall'IDE. Contiene tutti i plugin standard già pre-configurati e quindi non richiede alcuna preparazione per essere usato, se non l'installazione su un proprio dispositivo.

Per Android, l'installazione avviene tramite Google Play, cercando il nome InstaLauncher. Per iOS, l'installazione avviene tramite TestFlight in quanto Apple non consente la pubblicazione su App Store di applicazioni che hanno lo scopo di testare altre applicazioni. Si deve quindi procedere installando l'applicazione TestFlight da App Store e poi scansionando con la fotocamera dell'iPhone il seguente QR Code:

Developer Launcher

Un Developer Launcher è un launcher di test che funziona esattamente come InstaLauncher ma permette di aggiungere plugin Cordova personalizzati e testare quindi l'applicazione dall'IDE potendo interagire anche con i nuovi plugin.

L'uso del Developer Launcher è necessario solo in caso di integrazione di ulteriori plugin Cordova; in altri casi si consiglia l'utilizzo di InstaLauncher per il test delle applicazioni.

Nota bene: il Developer Launcher non consente la pubblicazione delle applicazioni negli App Store. Può essere utilizzato solo direttamente nel proprio dispositivo, installandolo direttamente o per mezzo dei canali di test delle applicazioni.

Launcher Standard

Un Launcher Standard è un launcher di produzione, cioè pensato per pubblicare le proprie applicazioni su App Store e Google Play.

Launcher Premium

Un Launcher Premium aggiunge alle caratteristiche del launcher standard la possibilità di aggiornare l'applicazione in tempo reale, senza doverla nuovamente pubblicare a meno di dover modificare la parte nativa, cioè la configurazione dei plugin.

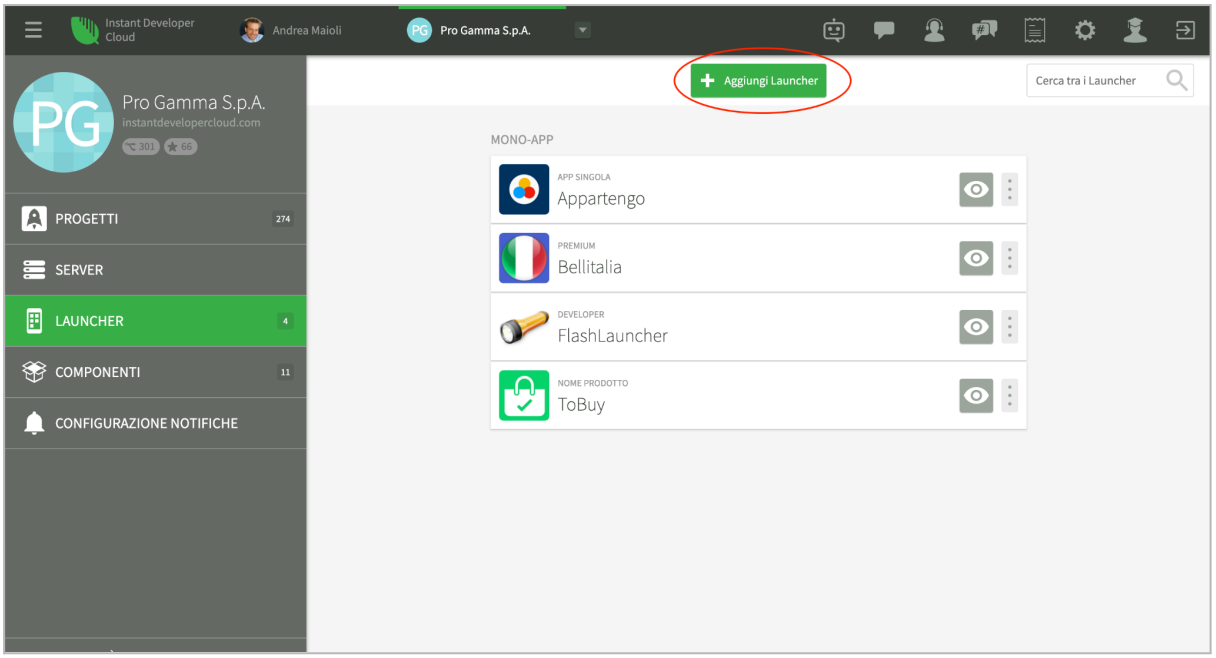
Launcher Premium Vincolato

Un Launcher Premium Vincolato è un launcher premium adatto in modo specifico per i casi in cui la stessa build del progetto deve essere rilasciata più volte su App Store e/o Google Play. Come esempio di utilizzo possiamo immaginare un'applicazione che permette di prenotare servizi per centri estetici. Se tale applicazione deve essere utilizzata da centri estetici diversi dovrà essere rilasciata su App Store e Google Play più volte, una per ogni centro estetico (con personalizzazioni quali il marchio, la palette dei colori, i dati di contatto o il calendario di prenotazione, per esempio). Queste ulteriori pubblicazioni richiedono ulteriori launcher, tuttavia la build del progetto sarà la stessa per tutti i launcher perché la logica dell'applicazione sarà sempre la stessa.

L'acquisto della licenza per un launcher premium vincolato è più vantaggioso rispetto a quello delle licenze per i diversi launcher premium che sarebbero necessari nei casi sopracitati. Si noti, tuttavia, che il launcher premium vincolato non consente di installare una build diversa da quella del launcher master da cui deriva.

Creazione di un launcher

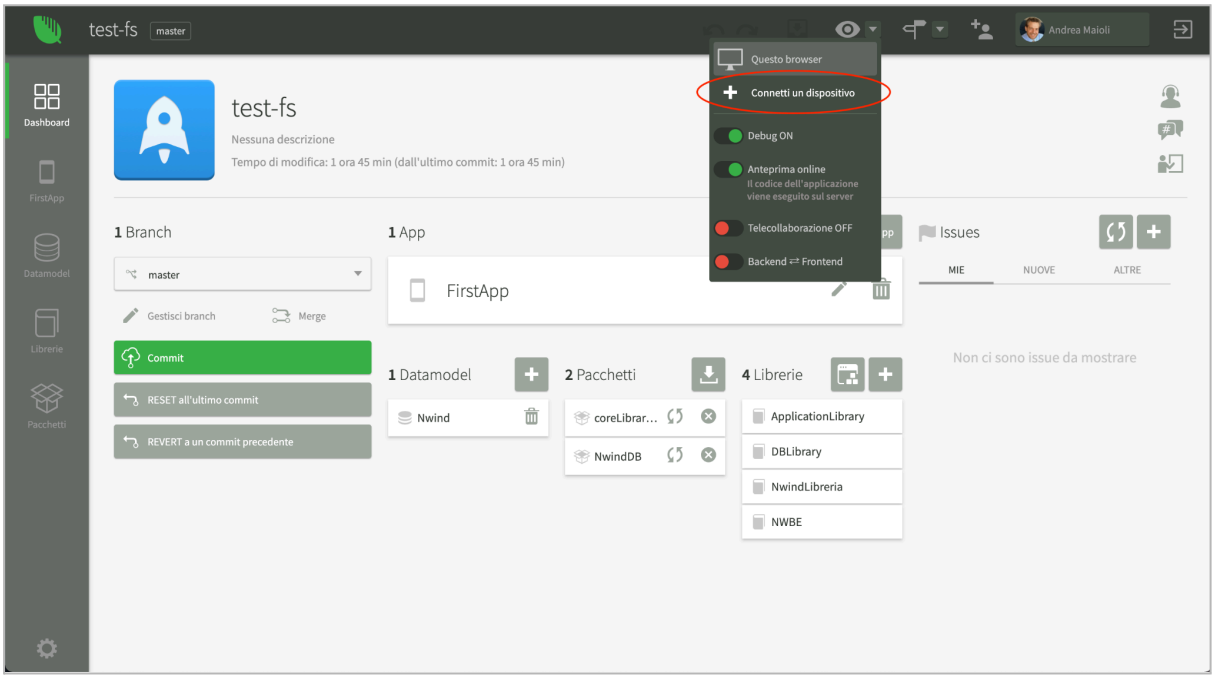
A parte InstaLauncher, già presente negli app store, tutti gli altri tipi di launcher devono essere creati e configurati tramite il pulsante *Aggiungi Launcher* dalla pagina *Launcher* della console, come mostrato nell'immagine seguente.
L'operazione di creazione di un Launcher viene sempre eseguita come acquisto di un oggetto nella console, anche se nel caso di Developer Launcher non ci sarà nessun addebito. Anche in caso di Developer Launcher è quindi richiesta la configurazione del metodo di pagamento per l'organizzazione a cui si appartiene.



Anteprima dell'applicazione in un launcher

Prima di installare un'applicazione in un launcher è necessario testarla negli ambienti finali iOS e Android per verificarne il corretto funzionamento. A questo scopo è possibile eseguire l'applicazione in anteprima in un launcher di test oltre che in una finestra browser. Per ottenere questo risultato è necessario:

1) Installare InstaLauncher o un launcher di test in un proprio dispositivo.
2) Utilizzare l'opzione *Connetti un dispositivo* nel menu di anteprima, come mostrato nell'immagine qui sotto.
3) Cliccare il pulsante *Connect via QR Code* nel launcher ed inquadrare il QR Code che appare nell'IDE.



I plugin nativi

In questo paragrafo vediamo come utilizzare i plugin che permettono di accedere alle funzioni native dei dispositivi. Le descrizioni e gli esempi riportati sono da intendere per applicazioni in funzione in un launcher e non in un browser, anche se quello del dispositivo.

Se si utilizzano i plugin in ambiente browser, essi non generano eccezioni, ma possono restituire dati vuoti o attivare una versione simulata del plugin corrispondente. In questo modo è possibile testare le applicazioni anche con un browser senza dover incorrere in errori o eccezioni.

Per ogni plugin viene riportato il relativo repository GitHub in modo da poter approfondire il funzionamento leggendo la documentazione originale e il codice sorgente.

È possibile testare molti dei plugin nativi tramite il seguente progetto di esempio: Plugins Design Patterns. Dopo aver aperto il progetto cliccando sul link, si consiglia di connettere un proprio dispositivo alla sessione IDE, come illustrato nell'immagine alla pagina precedente.

Barcode Scanner

Il plugin *barcode scanner* permette di aprire la fotocamera del dispositivo e di leggere un QR code o un barcode inquadrandolo con il dispositivo. Un esempio di codice è il seguente:

```
let bc = yield app.device.barcodeScanner.scan();
```

L'esecuzione del flusso di codice JavaScript rimane sospesa fino al termine della scansione a causa dell'uso di *yield* che sincronizza le operazioni. Nel caso l'operazione venga annullata senza leggere nessun codice, viene restituita una stringa vuota.

Se utilizzato in ambiente browser, il plugin apre la fotocamera del dispositivo e tenta di simulare la funzionalità nativa corrispondente. Non è garantita la compatibilità con la versione nativa.

Questo plugin viene implementato tramite: phonegap-plugin-barcodescanner.

Camera

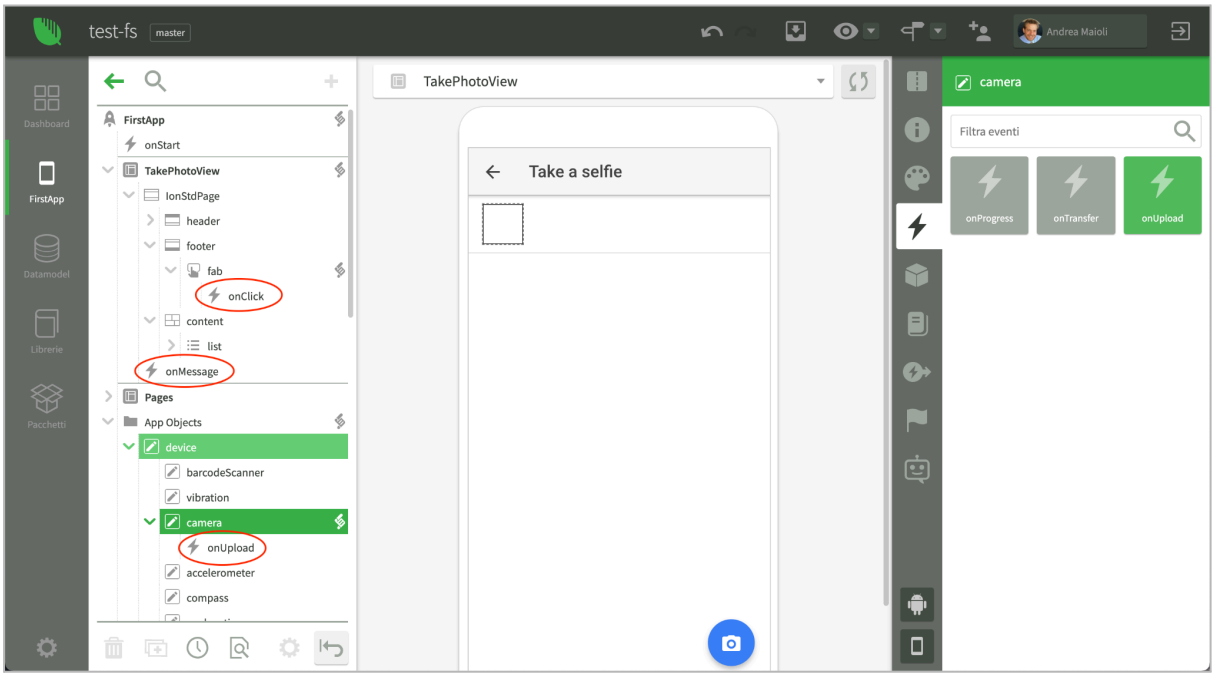
Il plugin *camera* permette di scattare una foto con la fotocamera del dispositivo, oppure di scegliere una foto dalla galleria. La foto selezionata può essere memorizzata come file locale oppure automaticamente trasferita al server.

Oltre alla selezione di foto, questo plugin permette di effettuare uno screenshot del dispositivo e caricarlo sul server; consente infine di inserire un'immagine nella galleria del dispositivo.

Il plugin camera viene implementato tramite: cordova-plugin-camera e cordova-screenshot.

Scattare una foto o selezionare elementi dalla galleria

Vediamo la struttura base per permettere all'utente di scattare una foto o selezionare un'immagine dalla galleria.



Nell'esempio mostrato, è presente un pulsante "fotocamera" che nell'evento *onClick* attiva la relativa funzione. Il codice dell'evento è il seguente:

```
$fab.onClick = function (event)
{
  app.device.camera.getPicture({
    sourceType : "camera", targetWidth : 640, targetHeight : 640
  });
};
```

Il metodo *getPicture* presenta varie opzioni, consultabili nella documentazione in linea. Le più importanti sono:

- La proprietà *sourceType* che permette di scegliere se scattare una foto (*camera*) o leggere dalla galleria (*photolibrary*). Il valore di default è *camera*.
- Le proprietà *targetWidth* e *targetHeight* che impostano la dimensione massima dei lati dell'immagine. Si consiglia di impostare un valore per limitare la dimensione del file immagine che verrà prodotto.

Dopo la chiamata al metodo *getPicture* nel dispositivo si apre la fotocamera o il picker di selezione dalla galleria. Si noti che la chiamata non viene sincronizzata, quindi il codice successivo verrà eseguito subito, prima ancora che la foto venga scattata o selezionata.

Se l'utente cancella l'operazione, non accade più nulla. Se invece scatta la foto o ne seleziona una, il framework notifica eventi diversi in funzione della modalità di esecuzione dell'applicazione.

Se infatti la sessione applicativa è *online*, cioè in esecuzione nel server nel cloud, allora il dispositivo esegue automaticamente un comando di upload della foto al server e poi notifica all'applicazione il metodo *app.device.camera.onUpload,* passando come parametro un oggetto *File* che contiene l'immagine.

Vediamo il codice dell'evento, che, come si può notare in figura, può essere implementato aggiungendo l'evento *onUpload* all'oggetto *device/camera* di solito contenuto nella cartella *App Objects* del progetto.

```
app.device.camera.onUpload = function (picture)
{
  App.Pages.postMessage(app, {picture : picture, bc : true});
};
```

Questo esempio fa uso del metodo *postMessage* del Page Controller IonicUI per passare alle videate aperte il parametro *picture*, che è l'oggetto *File* relativo alla foto caricata dal dispositivo.

A questo punto non resta che implementare l'evento *onMessage* nella videata giusta per ottenere il risultato desiderato. Vediamo un esempio:

```
App.TakePhotoView.prototype.onMessage = function (message)
{
  if (message.picture) {
    let f = message.picture; // type:File
    $image.src = yield f.getPublicUrl();
  }
};
```

Se il messaggio contiene la proprietà *picture*, cioè il file caricato dal dispositivo, esso viene convertito al tipo *File* tramite il suggerimento *// type:File*. A questo punto per visualizzare l'immagine è sufficiente impostare la proprietà *src* dell'elemento image usando la *publicUrl* della foto.
Per testare questo esempio è sufficiente lanciare l'applicazione in anteprima nel dispositivo o nel browser, poi cliccare il pulsante "fotocamera" ed infine scattare una foto.

Funzionamento con sessioni locali

Il caso più frequente di utilizzo del plugin fotocamera si ha con applicazioni installate nel dispositivo, quindi con sessioni in modalità locale, chiamata anche offline.

Vediamo ora come ottenere il caricamento della foto su un server nel cloud. A tal fine occorre predisporre un'applicazione installata in un server nel cloud che implementi una Web API che accetti e gestisca l'invio di file da un dispositivo, come illustrato nel paragrafo *Esporre Web API* del libro: 06-WebAPI.

A questo punto, nella chiamata al metodo *getPicture* occorre specificare anche il parametro *upload*, come si vede nell'esempio seguente:

```
$fab.onClick = function (event)
{
  app.device.camera.getPicture({
    sourceType : "camera", targetWidth : 640, targetHeight : 640,
    upload: {
      url:"https://<server nel cloud>/appname"
      cmd: ""
    }
  });
};
```

La proprietà *url* contiene l'endpoint della WebAPI che accetta il caricamento del file, mentre *cmd* contiene i parametri query string necessari alla WebAPI. Si noti che *mode=rest* viene aggiunto automaticamente e non è quindi necessario specificarlo.

Quando il dispositivo è in funzione in modalità locale e viene specificato il parametro upload, oltre all'evento *onUpload* verrà notificato anche l'evento *app.device.camera.onTransfer*. Nel codice di gestione di questo evento è possibile leggere la risposta della WebAPI del server, che può contenere il nome del file effettivo relativo al server nel cloud.

Questo è un esempio di parametro *picture* passato all'evento *onUpload* quando l'applicazione funziona in locale:

```
onUpload -> {
  _class: "File",
  path: "uploaded/cdv_photo_1634720231.jpg",
  type: "permanent",
  publicUrl:
     "inde://localhost/_app_file_/var/mobile/Containers/Data/Application/61
     A59B5A-90B2-4489-B626-592110C41F2A/Library/NoCloud/fs/ideapp/uploaded/
     cdv_photo_1634720231.jpg"
}
```

Subito dopo viene notificato anche l'evento *onTransfer* che certifica che la foto è stata caricata nel server. La risposta della WebAPI può essere usata per sapere qual è il nome del file nel server e quindi la sua *publicUrl*.

```
onTransfer -> {
  id: "",
  fileName: "cdv_photo_1634720231.jpg",
  byteSent: 23136,
  responseCode: 200,
  responseText: "uploaded/08d770f3-6eaf-442c-a450-62cd56f9844a.jpg"
}
```

I dati restituiti dal server dipendono dall'implementazione scelta per la WebAPI. Si consiglia sempre di richiedere un token di autenticazione e di restituire la *publicUrl* della foto in modo che l'applicazione locale possa memorizzarla.

Questi dati di solito vengono passati alle videate come abbiamo visto nel caso precedente. Si noti che se la foto viene scattata mentre non c'è connessione ad internet, l'evento *onUpload* permette di accedere al file della foto, mentre l'evento *onTransfer* segnalerà un errore.

Se la foto da caricare ha un dimensione rilevante è possibile implementare anche l'evento *app.device.camera.onProgress* che viene notificato più volte in modo da poter monitorare l'andamento del caricamento della foto.

Un modo semplice per sfruttare questo evento consiste nell'usare il metodo *app.popup* con *type:"loading"* sia nell'evento *onProgress* per segnalare la percentuale a cui il caricamento è arrivato, sia nell'evento *onTransfer* per chiudere il popup di attesa.

Nota bene: quando l'applicazione è in modalità online, il parametro *upload* non deve essere specificato.

Manipolare le fotografie caricate

Una volta che le fotografie sono state caricate nel server, esse possono essere manipolate tramite la libreria GM (GraphicsMagick) che è possibile importare nel proprio progetto tramite il package GM. È possibile approfondire l'utilizzo di questa libreria nel progetto di esempio: GraphicsMagick.

Si segnala inoltre che è possibile estrarre informazioni semantiche dalle immagini caricate tramite le API di visione, come ad esempio quelle di Google: Cloud Vision API. Queste API prevedono solitamente un costo legato all'utilizzo.

Catturare uno screenshot dell'applicazione.

Se si desidera catturare uno screenshot dell'applicazione è sufficiente utilizzare il metodo *app.device.camera.getScreenshot* al posto di *getPicture*. Il meccanismo di caricamento è uguale al precedente.

Si noti che la cattura dello screenshot avviene solo nelle applicazioni in esecuzione in modalità locale in un dispositivo.

Salvare una foto nella galleria del dispositivo

Se si desidera aggiungere alla galleria del dispositivo una foto presente localmente o tramite una URL qualunque, è possibile usare il metodo *app.device.camera.saveImageToGallery*, con un codice come il seguente

```
try {
  yield app.device.camera.saveImageToGallery("<URL dell'immagine>");
}
catch (ex) {
  ...
}
```

Se la foto non può essere scaricata o non esiste come file locale, oppure non è nel formato giusto o infine l'utente non concede il permesso di modificare le foto, verrà generata un'eccezione.

L'operazione di salvataggio nella galleria non ha effetto quando l'applicazione è utilizzata tramite un browser.

Compass e Accelerometer

Leggere la direzione in cui è puntato il dispositivo

Il plugin *Compass* permette di conoscere la direzione del dispositivo tramite il sensore bussola integrato. Il metodo di utilizzo consigliato consiste nell'attivare la rilevazione periodica tramite il metodo:

```
app.device.compass.watchHeading(1000);
```

Dove il numero intero passato come parametro indica l'intervallo di rilevamento in millisecondi. Ad ogni rilevamento viene notificato all'applicazione l'evento *app.device.compass.onHeading* come si vede nell'evento seguente:

```
app.device.compass.onHeading = function (heading)
{
  console.log(heading);
};
```

Un esempio di valore restituito è il seguente, dove si può notare che l'accuratezza della misurazione migliora man mano che il dispositivo viene ruotato.

```
{magneticHeading: 43.107398986816406, trueHeading: 46.83002471923828,
headingAccuracy: 13.023140907287598, timestamp: 1634886588812.607}
```

Per interrompere la rilevazione è necessario chiamare il metodo:

```
app.device.compass.clearWatch();
```

Infine si può notare che è disponibile il metodo *app.device.compass.getcurrentHeading* che restituisce i dati di puntamento istantanei.

Leggere l'orientamento del dispositivo

In maniera analoga a *Compass*, il plugin *Accelerometer* permette di leggere i dati del dispositivo leggendo il sensore accelerometro integrato.

Si può quindi attivare o interrompere il rilevamento dei dati tramite i metodi *app.device.accelerometer.watchAcceleration* e *app.device.accelerometer.clearWatch*. I dati rilevati vengono restituiti tramite l'evento seguente:

```
app.device.accelerometer.onAcceleration = function (acceleration)
{
  console.log(acceleration);
};
```

Un esempio di dati ritornati, con dispositivo appoggiato su una scrivania, è il seguente:
`{x: -0.052690429687500005, y: 0.15672409057617187, z: 9.77976287841797, timestamp: 1634887429640.519}`

Anche per il plugin *Accelerometer* è disponibile il metodo *app.device.accelerometer.getCurrentAcceleration* che restituisce i dati istantanei.

Questi plugin sono implementati tramite: cordova-plugin-device-orientation e cordova-plugin-device-motion.

Custom URL Scheme

Il plugin *Custom URL Scheme* permette di definire un prefisso di URL che, se chiamato da una pagina web visualizzata nel dispositivo, permette di attivare direttamente l'applicazione passando anche dei parametri. Questo plugin è utilizzabile solo nei dispositivi e viene implementato tramite: cordova-plugin-custom url scheme.

L'attivazione del plugin e la definizione del prefisso avvengono nella pagina *Plugin* della configurazione del launcher nella console di Instant Developer Cloud.

L'attivazione dell'applicazione tramite custom URL causa la notifica dell'evento *app.onCommand*, in cui il parametro *request* rappresenta i dati passati nella URL.

Nell'esempio in figura, per attivare l'applicazione passando dei parametri è possibile aprire nel dispositivo un link con URL:

```
appa://Appa?eventid=12345&userid=mrossi
```

Dove *Appa* è il nome dell'applicazione all'interno del progetto Instant Developer Cloud. A questo punto il seguente evento *onCommand*:

```
App.Session.prototype.onCommand = function (request)
{
  console.log(request);
};
```

emetterà questo log: `{query: {eventid: 12345, userid: "mrossi"}}`.

Contacts

Il plugin *Contacts* permette di leggere l'elenco dei contatti del dispositivo, di aggiungerne di nuovi o di modificare o cancellare quelli attuali.

L'uso del plugin *Contacts* richiede una particolare progettazione della UX dell'applicazione, in quanto la prima volta che si accede alla lista contatti del dispositivo, il sistema operativo richiede una conferma di accesso all'utente. Se l'utente nega il permesso, non sarà più possibile accedere alla lista contatti in quanto la richiesta viene fatta solo la prima volta.

Prima di leggere la lista contatti è quindi necessario sapere se il permesso è già stato ottenuto, ed in caso contrario si consiglia di aprire una specifica videata dell'applicazione che spiega perché è necessario accedere ai contatti e si chiede all'utente se è d'accordo.

Se l'utente accetta, allora si effettua l'accesso alla lista in quanto si suppone che risponderà in modo affermativo anche alla richiesta di conferma del sistema operativo. Se invece l'utente non accetta, non si deve effettuare l'accesso, in modo che alla prossima occasione sarà possibile nuovamente far apparire la videata interna all'app di richiesta consenso.

Al momento il plugin non espone un metodo per determinare se il permesso è già stato dato ed eventualmente per richiedere il permesso. È quindi necessario memorizzare manualmente il fatto che si sia ottenuto o meno il permesso.

Vediamo adesso quali operazioni sono consentite:

- *app.device.contacts.find*: restituisce una lista di contatti.
- *app.device.contacts.read*: legge i dati di dettaglio di un contatto.
- *app.device.contacts.create*: crea un nuovo contatto.
- *app.device.contacts.update*: aggiorna un contatto.
- *app.device.contacts.delete*: cancella un contatto.

Il progetto di esempio Plugins Design Patterns contiene un esempio completo di utilizzo di queste funzioni. Questo plugin viene implementato tramite: cordova-plugin-contacts.

Calendar

Il plugin *Calendar* permette di gestire il calendario del dispositivo, in particolare le seguenti operazioni.

- Leggere la lista dei calendari, aggiungere o cancellare un calendario.
- Leggere gli eventi di un calendario, aggiungere modificare o cancellare eventi.
- Aprire l'applicazione nativa per la gestione di una data.
- Richiedere i permessi di utilizzo.

Come nel caso del plugin *Contacts*, l'accesso ai dati del calendario genera una richiesta di conferma esplicita da parte del sistema operativo. Si consiglia di utilizzare una videata specifica dell'applicazione per chiedere all'utente di accedere ai dati del suo calendario.

Il progetto di esempio: Plugins Design Patterns contiene un esempio completo di utilizzo di queste funzioni. Questo plugin viene implementato tramite: cordova-plugin-calendar.

Facebook e SignInWithApple

I plugin *Facebook* e *SignInWithApple* consentono di gestire la registrazione e l'accesso all'applicazione tramite i relativi servizi. Oltre alla funzione di login, *Facebook* contiene anche la possibilità di interrogare il database di Facebook relativo alle informazioni e alle attività dell'utente che ha fatto il login.

Si ricorda che attivando un'opzione di registrazione o di login tramite un gestore di account come Facebook e Google è obbligatorio implementare anche *SignInWithApple* pena l'esclusione dell'applicazione da App Store. Se invece il login avviene inserendo username e password, *SignInWithApple* non è richiesto.

Questi plugin vengono implementati tramite: cordova-plugin-facebook-connect e cordova-plugin-apple-login.

Facebook Login

Prima di poter utilizzare il plugin *Facebook*, occorre configurare una propria applicazione facebook registrandosi su Facebook for Developers.

A questo punto il plugin *Facebook* potrà essere usato sia quando l'applicazione è in funzione in un browser che nei dispositivi. L'inizializzazione del plugin avviene in questo modo:

- Se l'applicazione è in funzione in un browser, chiamando il metodo *app.device.fecebook.init*, passando il parametro *app_id* che si trova nella configurazione dell'applicazione nel portale Facebook for Developers.
- Se l'applicazione è in funzione in un dispositivo, configurando i parametri *app_id* e *app_name* nella sezione plugin del launcher nella console.

Dopo l'inizializzazione è possibile utilizzare gli altri metodi del plugin, fra i quali:

- *app.device.facebook.login*: fa apparire la videata per il login dell'utente tramite Facebook. Se l'utente effettua il login, questo metodo restituisce i dati identificativi dell'utente nel sistema Facebook che possono essere memorizzati per identificarlo anche nell'applicazione.
- *app.device.facebook.api:* interroga i dati dell'utente su Facebook. Per poter usare questa API è necessario che l'applicazione richieda particolari permessi a Facebook nella pagina di configurazione del portale Facebook for Developers.

SignInWithApple

Prima di poter utilizzare questo plugin occorre configurare il servizio Sign In With Apple per la propria applicazione come descritto nella documentazione in linea. A questo punto è sufficiente chiamare il metodo *app.device.signInWithApple.request* passando il tipo di informazioni richieste e il *clientID* che si ottiene durante la configurazione del servizio. Se l'utente effettua il login, il metodo restituisce i dati richiesti.

Questo plugin può essere usato in iOS, in Android e anche in ambiente browser. Negli ultimi due contesti, l'utente dovrà effettuare il login al suo account Apple per confermare il login.

Geolocation e BackgroundLocation

I plugin *Geolocation* e *BackgroundLocation* permettono di conoscere la posizione del dispositivo mentre l'app è in esecuzione o anche mentre essa è in background.

Questi plugin vengono implementati tramite: cordova-plugin-geolocation e cordova-plugin-background-geolocation

La posizione del dispositivo è un'informazione estremamente riservata, quindi il sistema operativo protegge gli utenti da abusi. Si consiglia di limitare la localizzazione in background solo agli utilizzi connaturati all'applicazione; per tutte le altre esigenze, deve essere usata la localizzazione in app tramite *Geolocation*.

Prima di richiedere la posizione occorre essere sicuri che l'utente risponda positivamente alla richiesta di permessi del sistema operativo. A tal fine si consiglia di utilizzare una pagina interstiziale che spieghi all'utente perché l'applicazione deve conoscere la posizione del dispositivo e chieda all'utente se è d'accordo o meno a concedere il permesso.

A questo punto, per conoscere la posizione attuale del dispositivo, è possibile chiamare il metodo seguente:

```
let pos = yield app.device.geolocation.getCurrentPosition(options);
```

Se l'utente non ha ancora concesso il permesso, questa chiamata visualizza la videata popup di richiesta permessi. Se l'utente ha negato il permesso, il metodo restituisce *undefined* e il popup di richiesta permessi non appare.

Fra le opzioni è presente la possibilità di specificare un timeout entro il quale arrivare a leggere una posizione sufficientemente accurata, e quella di richiedere una posizione altamente precisa. Si rimanda alla documentazione in linea per la descrizione completa.

Alternativamente è possibile attivare il tracciamento della posizione utilizzando il metodo:

```
app.device.geolocation.watchPosition();
```

Ad ogni rilevamento verrà notificato l'evento *app.device.geolocation.onPosition* che può essere inoltrato alla videata attiva con il codice seguente:

```
app.device.geolocation.onPosition = function (coords)
{
  // send message with position
  App.Pages.postMessage(app, {type : "position", position : coords});
};
```

Per interrompere il tracciamento della posizione è sufficiente chiamare il metodo:

```
app.device.geolocation.clearWatch();
```

Questo plugin è utilizzabile anche nelle applicazioni in funzione in un browser.

Localizzazione in background

L'utilizzo del plugin di localizzazione in background consente di registrare le modifiche alla posizione del dispositivo anche mentre l'applicazione è in background. Se l'utente chiude esplicitamente l'applicazione tramite le funzioni del dispositivo, anche la localizzazione in background verrà interrotta.

L'uso della localizzazione in background viene esplicitamente controllato durante il processo di verifica su app store. Se l'applicazione non ha una ragione specifica per rilevare la posizione in background, l'applicazione verrà rifiutata.

In generale, il ciclo di vita della localizzazione in background richiede i seguenti passaggi:

1) Configurazione delle opzioni del plugin tramite il metodo *app.device.backgroundLocation.configure(options)*.
2) Se l'utente non ha ancora concesso il permesso, dare comunicazione all'utente di voler iniziare la localizzazione e spiegazione dei motivi. Usare il metodo *app.device.backgroundLocation.checkStatus()* per verificare i permessi.
3) Iniziare il tracciamento della posizione tramite il metodo *app.device.backgroundLocation.start().*
4) Mentre l'applicazione è ancora in foreground, il plugin invia la posizione tramite il metodo *app.device.backgroundLocation.onLocation*.
5) Quando l'applicazione entra in background, il plugin continua ad immagazzinare le posizioni in un database interno, ma l'applicazione non ne viene a conoscenza. Tramite le opzioni è tuttavia possibile configurare l'endpoint di una WebAPI a cui il plugin comunica la posizione corrente anche quando è in background, se la rete internet è attiva.
6) Quando l'applicazione torna in foreground, tramite il metodo *app.resume* è possibile leggere il contenuto del database interno al plugin utilizzando il metodo *getLocations*. Le posizioni possono essere cancellate tramite i metodi *deleteLocation* o *deleteAllLocations*.
7) Quando il rilevamento della posizione deve essere interrotto è possibile chiamare il metodo *app.device.backgroundLocation.stop().*

Nei dispositivi Android, la localizzazione può essere interrotta per ottimizzare il consumo di batteria. Per evitare questi casi è possibile usare il metodo *app.device.preferences.ignoreBatteryOptimization* che richiede al sistema di evitare la chiusura dell'applicazione in questo caso. Il sistema evita di farlo se l'utente concede il permesso.

Normalmente, quando l'applicazione è in background, il codice JavaScript che costituisce l'applicazione non è in funzione, quindi l'applicazione non è in grado di ricevere le modifiche alla posizione in tempo reale. Se fosse necessario saperlo, è possibile configurare il plugin per inviare la posizione ad una WebAPI esterna.

Esiste tuttavia un espediente che permette di mantenere il JavaScript funzionante anche quando l'applicazione è in background: è sufficiente infatti inserire un tag video nella pagina e avviare la riproduzione di un file audio completamente silenzioso. In questo modo l'applicazione rimane funzionante anche in background, ma il consumo di batteria del

dispositivo risulta aumentato; inoltre nella videata di blocco schermo viene segnalato che l'applicazione sta riproducendo un file musicale. Non si consiglia di fare affidamento solo su questo espediente per rilevare i dati in tempo reale in quanto è contrario alle politiche di ottimizzazione dei dispositivi e potrebbe essere reso inefficace in future versioni degli stessi.

Questo plugin è utilizzabile solo se l'applicazione è locale ed è installata in un launcher o su InstaLauncher. Se essa infatti viene lanciata in anteprima su InstaLauncher, quando esso entra in background l'anteprima viene interrotta e il tracciamento della posizione viene bloccato.

Il progetto di esempio Plugins Design Patterns contiene un esempio di utilizzo di queste funzioni nella videata *BackgroundLocation*. Si consiglia di leggere la documentazione in linea del plugin per conoscere tutti i metodi e leggere le varie opzioni di configurazione.

Questo plugin non è utilizzabile nelle applicazioni in funzione in un browser.

Media

Il plugin *Media* gestisce la registrazione e la riproduzione di file audio, oltre che la manipolazione dei file audio nel file system locale del dispositivo.

Il progetto di esempio Plugins Design Patterns contiene un esempio di utilizzo di queste funzioni nella videata *MediaView*. Si consiglia di leggere la documentazione in linea del plugin per conoscere tutti i metodi e leggere le varie opzioni di configurazione.

Questo plugin viene implementato tramite: cordova-media-with-compression
Il plugin non è disponibile in ambiente browser.

Gestione dei file audio

In particolare, per la gestione dei file audio nel file system locale si hanno questi metodi:

- *app.device.media.initStorage()*: prepara il file system locale per la gestione dei file audio. Deve essere chiamato prima di utilizzare le funzioni di questo plugin.
- *app.device.media.download()*: scarica un file dalla url passata come parametro e lo memorizza nel file system locale con il nome specificato.
- *app.device.media.upload()*: carica il file con il nome specificato alla url passata come parametro.
- *app.device.media.exists()*: restituisce *true* se il file audio con il nome specificato esiste.
- *app.device.media.size()*: restituisce la dimensione del file audio con il nome specificato.
- *app.device.media.remove()*: cancella il file audio con il nome specificato.
- *app.device.media.url()*: ritorna il percorso completo del file audio specificato nel file system locale del dispositivo.

Registrazione dei file audio

Per la registrazione di file audio sono disponibili i seguenti metodi:

- *app.device.media.startRecord():* inizia la registrazione del file audio specificato. Deve avere come estensione **m4a**.
- *app.device.media.pauseRecord():* mette in pausa la registrazione del file audio.
- *app.device.media.resumeRecord():* continua la registrazione del file audio.
- *app.device.media.stopRecord():* conclude la registrazione del file audio.
- *app.device.media.release()*: libera le risorse relative al file audio; si consiglia di utilizzarlo prima di *startRecord* e dopo *stopRecord*.
- *app.device.media.getRecordLevels():* permette di conoscere il livello di picco e medio della registrazione in corso.

Se si utilizza un dispositivo Android, prima di iniziare una registrazione si deve verificare di avere il permesso di utilizzare il microfono. A tal fine è presente il metodo *app.device.speech.hasPermission(true)*. Vediamo quindi un esempio di codice tratto da Plugins Design Patterns che attiva la registrazione:

```
$recStart.onClick = function (event)
{
  if (app.device.operatingSystem === "android") {
    if (!(yield app.device.speech.hasPermission(true))) {
      var hasPermission = yield app.device.speech.requestPermission(true);
      if (!hasPermission) {
        yield app.popup({type : "alert", title : "Error", message : "To use
                  this example, enable microphone permission",
                  buttons : [t("Close")]});
        return;
      }
    }
  }
  // always free the resource before starting a new recording
  app.device.media.release("test.m4a");
  //
  // start recording
  app.device.media.startRecord("test.m4a");
};
```

Riproduzione dei file audio

Per la riproduzione di file audio sono disponibili i seguenti metodi:

- *app.device.media.play()*: inizia o continua la riproduzione del file audio specificato.
- *app.device.media.pause()*: mette in pausa la riproduzione del file audio specificato.
- *app.device.media.stop()*: termina la riproduzione del file audio specificato.
- *app.device.media.release()*: rilascia le risorse relative al file audio specificato. Da utilizzare dopo lo *stop* o prima di *play*.
- *getCurrentPosition()*: ritorna la posizione attuale di riproduzione all'interno del file audio.
- *getDuration()*: ritorna la lunghezza di riproduzione totale del file audio specificato.
- *seekTo()*: sposta il punto di riproduzione attuale del file audio.
- *setVolume()*: modifica il volume di riproduzione del file audio.

NFC, Beacon, BLE

Questi plugin permettono di comunicare con diversi dispositivi esterni al device, in particolare di tipo NFC, Beacon e BLE.

Per gli scopi introduttivi di questo libro affrontiamo la comunicazione con dispositivi di tipo NFC, rimandando Beacon e BLE agli esempi in linea e alla documentazione dei plugin.

La funzionalità BLE (*Bluetooth Low Energy*, noto anche come *Bluetooth Smart*) è una tecnologia di comunicazione wireless progettata per fornire un consumo energetico ridotto e una maggiore durata della batteria rispetto al *Bluetooth classico*, mantenendo comunque una connettività affidabile.
Quindi occorre che i dispositivi che si vogliono connettere utilizzando il plugin BLE siano compatibili con il *Bluetooth Low Energy* o *Bluetooth Smart* altrimenti non verranno identificati dal plugin.
Se si deve collegare un dispositivo Bluetooth classico, quindi non rilevabile dal plugin BLE, è sempre possibile integrare un plugin Cordova in grado di collegarsi anche con dispositivi non compatibili con il BLE.

Le funzionalità NFC vengono illustrate nell'esempio: nfc-dev-kit. Vediamo ora i passaggi principali per l'utilizzo di questo plugin.

Questi plugin vengono implementati tramite: phonegap-nfc, cordova-plugin-ibeacon e cordova-plugin-bluetoothle

Verifica della funzionalità NFC

È possibile verificare la possibilità di comunicare via NFC tramite il seguente metodo:

```
let ok = yield app.device.nfc.isAvailable();
```

Lettura di tag

La lettura dei tag viene attivata tramite la seguente riga di codice:

```
app.device.nfc.listen("ndef");
```

Mentre per Android è possibile lasciare sempre attivo l'ascolto di un tag, nel caso iOS esso deve essere attivato solamente quando è richiesta esplicitamente la lettura. Chiamando il metodo, infatti, appare sullo schermo l'interfaccia del lettore che invita ad avvicinare il dispositivo al device.

Se viene rilevato un tag mentre la lettura è attiva, viene notificato l'evento *app.device.nfc.onTag*. I dati del tag verranno passati come parametro all'evento.

Occorre tenere presente che nel caso iOS un tag verrà riconosciuto solo se contiene un messaggio di tipo "ndef" e che l'id del tag non è rilevabile. Nel caso Android, invece, il tag è rilevabile anche se non contiene alcun messaggio ed in questo caso l'id del tag è presente come parametro dell'evento.

Se si desidera quindi avere un'applicazione cross-device è necessario leggere l'id dei tag tramite Android e poi scrivere un messaggio ndef che contiene l'id del tag per poterlo rileggere poi anche da iOS.

È possibile interrompere la lettura di tag NFC tramite il metodo *app.device.nfc.unlisten*. Si consiglia di utilizzarlo solo su dispositivi Android per i quali la lettura dei tag può rimanere sempre attiva, cioè non è una operazione singola.

Scrittura di tag

La scrittura dei tag è disponibile solo per dispositivi Android e permette di specificare se cancellare i messaggi precedenti o semplicemente se aggiungere un nuovo messaggio a quelli attuali.

La scrittura di un tag avviene chiamando il metodo *listen* specificando le seguenti proprietà come secondo parametro:

- *write: "stringa":* è il messaggio da scrivere
- *erase: true:* esegue la cancellazione del tag prima di scrivere il messaggio.
- *makeReadOnly: true:* marca il tag come di sola lettura, non saranno possibili ulteriori operazioni di scrittura.

Ad esempio, la seguente riga di codice scrive un guid come messaggio ndef nel tag:

```
let guid = App.Utils.generateGuid36();
app.device.nfc.listen("ndef", { write:"id:" + guid});
```

Dopo aver dato questo comando, la scrittura avviene appena un tag viene avvicinato al device. Dopo aver eseguito la scrittura viene notificato all'applicazione l'evento *app.device.nfc.onTag*, passando come parametro una stringa che contiene l'esito dell'operazione. Si ricorda che, in caso di operazione di lettura, il parametro dell'evento è invece un oggetto.

L'operazione di scrittura avviene solo una volta. Dopo aver scritto il primo tag, la modalità di funzionamento ritorna in lettura. È possibile programmare ulteriori scritture chiamando nuovamente il metodo *listen* con i nuovi dati da scrivere.

Notification e Preferences

I plugin *Notification* e *Preferences* sono utilizzati per l'invio delle notifiche ai dispositivi. Questo processo richiede tre fasi: la raccolta del token di invio dai dispositivi, l'invio delle notifiche lato server ed infine la gestione delle notifiche arrivate nei dispositivi.

Il plugin *Notification* viene implementato tramite: cordova-plugin-push, cordova-plugin-local-notifications e cordova-plugin-dialogs. Il plugin *Preferences* viene implementato tramite: cordova-plugin-app-preferences e cordova-plugin-doze-optimize.

Raccogliere i token

Il primo passo per la gestione delle notifiche è quello di raccogliere i token che identificano l'applicazione su un determinato dispositivo.

Per poter raccogliere i token, è necessario prima ottenere il permesso di notifica dall'utente. Se tale permesso non è ancora stato ottenuto è opportuno presentare all'utente una pagina che spiega perché è importante che l'applicazione possa ricevere notifiche e che chiede all'utente di concedere il permesso. Si ricorda che, se l'utente nega il permesso, non sarà più possibile richiederlo nuovamente, si potrà solo aprire la pagina delle impostazioni dell'applicazione.

Per sapere se il permesso è già stato concesso si può usare il metodo *app.device.notification.hasPermission*. Per ricevere il token di invio notifiche, è possibile utilizzare il metodo *app.device.notification.register*. Usando questo metodo da un dispositivo iOS non si devono specificare parametri, mentre nel caso di dispositivi Android è necessario specificare il parametro *senderID,* recuperabile dalla Console di Firebase tra le impostazioni del progetto che deve essere creato per consentire all'app di ricevere notifiche push.

Se il metodo *register* non restituisce alcun valore, significa che l'utente ha negato il permesso. In questo caso è possibile segnalare il problema all'utente e chiedere se vuole abilitare le notifiche tramite la pagina di impostazioni dell'applicazione, tramite il metodo *app.device.preferences.show*.

Il token raccolto dovrà essere fornito al backend nel cloud, che avrà poi il compito di inviare le notifiche al dispositivo quando necessario. Questo può avvenire tramite sincronizzazione del database locale, oppure usando la DO remota o una Web API.

Nota: nell'evento *app.onStart* si deve verificare subito se l'applicazione ha già ottenuto il permesso di ricevere notifica ed in tal caso occorre chiamare subito il metodo *register*. In questo modo se l'applicazione è stata aperta dal sistema operativo a causa del tocco su una notifica ricevuta, i dati della notifica verranno subito ricevuti dall'applicazione.

Nota: nei dispositivi Android, a differenza di quelli iOS, è sempre possibile inviare notifiche all'applicazione, quindi il metodo *hasPermission* restituisce *true* e il metodo *register* restituisce sempre il token. Tuttavia l'utente ha la possibilità di nascondere tutte le notifiche dell'applicazione ed in questo caso non si avrà alcuna informazione. Anche nel caso Android, è quindi necessario rendere cosciente l'utente dell'importanza delle notifiche e operare quindi come nel caso iOS.

Inviare notifiche ai dispositivi

L'invio delle notifiche ai dispositivi è un'operazione che avviene nei server nel cloud a fronte di fatti importanti per gli utenti dell'applicazione.

Per inviare notifiche a uno o più dispositivi è disponibile il metodo *app.notification.push* che deve essere chiamato solo dall'applicazione in esecuzione in un server cloud. Questo metodo ammette due parametri: il primo è un oggetto che contiene i parametri della notifica,

il secondo è una stringa o un array di stringhe che contiene i token dei dispositivi da notificare. È possibile passare al metodo sia token provenienti da dispositivi Apple che token provenienti da dispositivi Android.

Il metodo restituisce un messaggio che rappresenta lo stato dell'invio o l'eventuale errore avvenuto. L'invio delle notifiche avviene dal server nel cloud dell'applicazione ai backend di Apple (APN) o Google (FCM) che poi effettuano la notifica dei messaggi ai dispositivi. Il risultato del metodo *push* riguarda solo la comunicazione con i due backend, e non è invece possibile determinare se una notifica è effettivamente stata ricevuta da un dispositivo.

Il seguente codice invia una notifica al dispositivo identificato dal token:

```
let token = "apn:12939jdjsdh...";
let msg = {
  title : "Buongiorno",
  body : "Oggi hai tre appuntamenti importanti. Tocca qui per vederli",
  topic: "com.myorganization.newapp",
  payload : "newapp",
  badge : 3
};
let b = yield app.device.notification.push(msg, token);
```

La proprietà *topic* è necessaria per l'invio della notifica a dispositivi iOS e deve essere valorizzata con il bundle dell'applicazione, che si può trovare nella sezione *Config.xml* delle impostazioni del launcher all'interno della console.

Si segnala infine che tutte le proprietà indicate nel messaggio di notifica vengono passate al backend Apple o Google, quindi è possibile ottenere notifiche avanzate contenenti immagini o pulsanti di risposta. Per maggiori informazioni fare riferimento alla documentazione del plugin di notifica: cordova-plugin-push.

Configurazione del server di invio notifiche

Finché l'applicazione viene testata su InstaLauncher non è necessaria alcuna configurazione del server, in quanto le chiavi di comunicazione con i backend di notifica sono già presenti nel framework.

Quando l'applicazione deve essere portata in produzione, e quindi installata su un launcher autonomo, prima di poter inviare notifiche, è necessario fornire al server di notifica le nuove chiavi di comunicazione con APN (Apple) e FCM (Android).

Android

Per permettere ad un'app di inviare notifiche tramite FCM è necessario creare un progetto sulla Console di Firebase e successivamente registrare al suo interno una nuova applicazione Android.

L'app appena registrata contiene due informazioni fondamentali: il file *google-service.json* e la chiave di comunicazione con il servizio FCM.

Il file di configurazione *google-service.json* viene generato durante la fase di registrazione dell'app. Deve essere scaricato e il suo contenuto deve essere inserito nella Console di Instant Developer Cloud, all'interno del campo omonimo nella sezione Plugin del proprio launcher.

La chiave di comunicazione è invece visualizzabile andando nelle impostazioni del progetto Firebase e accedendo poi alla sezione *Cloud Messaging*. Questa chiave sarà quella da usare per valorizzare la proprietà *app.device.notification.gcmKey* nell'applicazione in esecuzione in un server cloud che invierà le notifiche.

iOS

APN richiede tre informazioni per permettere ad un'app di inviare notifiche: un file in formato *.p8* che rappresenta la chiave, l'identificativo univoco della chiave e l'identificativo univoco del team che ha creato l'applicazione iOS.

Tutte queste informazioni sono ottenibili dalla Console sviluppatori Apple.

È possibile creare una chiave andando nella sezione *Certificates, Identifiers & Profiles* e quindi nel menu *Keys*. Durante la procedura di creazione è necessario abilitare il servizio APNs per la chiave. Una volta creata sarà possibile visualizzare il suo identificativo e scaricare il file .p8.

Nota: il file della chiave è scaricabile un'unica volta, quindi è fondamentale conservarlo adeguatamente.

L'identificativo del team è invece sempre visibile nell'angolo in alto a destra della console sviluppatori Apple.

A questo punto possiamo usare queste tre informazioni per valorizzare le proprietà *app.device.notification.teamId*, *app.device.notification.keyId* e *app.device.notification.keyPath* nell'applicazione in esecuzione in un server cloud che invierà le notifiche.

Nota: la proprietà *app.device.notification.keyPath* deve essere valorizzata con il contenuto del file .p8. In alternativa è possibile caricare il file della chiave nel progetto come risorsa e valorizzare la proprietà con *$nomeRisorsa*.

Per evitare di scrivere direttamente nel codice le informazioni necessarie all'applicazione per inviare notifiche push è possibile sfruttare i parametri di run-time dell'applicazione installata (o del server IDE o di produzione che si sta utilizzando), configurabili nella pagina del server sulla console di Instant Developer Cloud.

Creando i parametri *gcmKey*, *apnTeamId*, *apnKey* e *apnKeyId* e assegnando loro il valore corrispondente, queste informazioni saranno rese automaticamente disponibili all'applicazione, senza la necessità di scriverle nel codice.

Nota: il parametro di run-time *apnKey* deve essere valorizzato con il contenuto del file .p8.

Invio di notifiche locali

Oltre alle notifiche provenienti da server nel cloud, è possibile che un'applicazione programmi una notifica locale in modo da ricordare all'utente di effettuare azioni nel futuro. Si ricorda che anche per le notifiche locali è necessario acquisire il permesso.

Per programmare una notifica locale è possibile usare il metodo *app.device.notification.schedule* passando come parametro un oggetto o un array di oggetti che contengono le proprietà delle notifiche da programmare. Il seguente codice programma una notifica per essere mostrata dopo 10 secondi.

```
let atTime = new Date().getTime()+10000;
let notificationObj = {
  id: 1
  title : "Buongiorno",
  text : "Sono già passati 10 secondi dall'ultima volta",
  date : new Date(),
  at : atTime,
  badge : 1,
  payload : "XXX123"
};
app.device.notification.schedule(notificationObj);
```

I seguenti metodi consentono di gestire le notifiche già programmate:

- *app.device.notification.update*: aggiorna una notifica già programmata, identificata tramite il suo *id*.
- *app.device.notification.clear, clearAll*: elimina una o tutte le notifiche già mostrate all'utente.
- *app.device.notification.cancel, cancelAll*: elimina una o tutte le notifiche prima che siano mostrate all'utente.

Gestire le notifiche ricevute

Un'applicazione può ricevere notifiche mentre essa è in esecuzione in primo piano oppure quando essa è in background o non in esecuzione. Se l'applicazione è in esecuzione in primo piano, la notifica viene passata immediatamente all'applicazione e non viene mostrato il toast del sistema operativo. Negli altri casi, invece, la notifica viene passata se l'applicazione viene attivata cliccando sulla notifica. Se infine l'applicazione viene attivata dalla springboard del dispositivo, essa non riceve alcuna notifica, anche se esse sono state ricevute dal dispositivo.

Perché l'applicazione possa ricevere le notifiche da parte del sistema operativo, è necessario chiamare il metodo *register* direttamente nell'evento *app.onStart,* sempre che si sia già ottenuto il permesso di usarle.

Le notifiche vengono passate all'applicazione tramite l'evento *app.device.notification.onClick*, i cui parametri contengono tutti i dati relativi alla notifica.

Nel solo caso di notifiche locali, se l'applicazione è in esecuzione in primo piano le notifiche locali vengono comunicate tramite il metodo *app.device.notification.onTrigger.*

Si segnala infine che è possibile gestire il badge che mostra il numero di notifiche di un'applicazione nella springboard tramite i metodi *app.device.notification.getBadge* e *app.device.notification.setBadge.*

Ricevere notifiche in applicazioni browser

Il sistema di notifiche è disponibile anche per applicazioni browser in esecuzione nel desktop.

Nota: le API con cui i browser permettono di gestire questo tipo di notifiche non sono ancora stabili e disponibili in tutti gli ambienti. Si consiglia quindi di utilizzarle solo quando l'applicazione viene utilizzata tramite Google Chrome desktop. In alternativa si può considerare l'invio di email come sistema di notifica all'utente.

Così come nel caso di applicazioni in esecuzione su un dispositivo, anche per le applicazioni browser la registrazione alla ricezione di notifiche push avviene usando il metodo *app.device.notification.register*.

Il sistema di notifiche push dei browser prevede l'utilizzo di chiavi VAPID (Voluntary Application Server Identity) da usare per la registrazione. Queste chiavi devono essere generate usando il metodo *app.device.notification.generateKeys* e fornite quindi al metodo *app.device.notification.register* come proprietà *keys* del secondo parametro *options*.

Il seguente codice effettua la registrazione di un'applicazione browser alla ricezione di notifiche push.

```
let keys = app.device.notification.generateKeys();
let options = {keys: keys, resubscribe: true};
let browserToken = app.device.notification.register(undefined, options);
```

L'opzione *resubscribe* impostata a true forza la reiscrizione al servizio di notifiche push. È necessaria nel caso in cui si decida di rigenerare le chiavi VAPID ad ogni avvio dell'applicazione.

Mostrare dialog nativi

Il plugin *notification* mette a disposizione alcuni metodi per mostrare popup native del dispositivo. Si consiglia tuttavia di preferire l'uso del metodo *app.popup*. I metodi sono i seguenti: *app.device.notification.alert*, *app.device.notification.confirm*, *app.device.notification.prompt*, ed infine *app.device.notification.beep* che emette un certo numero di beep sonori.

Gestire il consumo energetico e la rete cellulare per Android

Il plugin *Preferences* permette infine di gestire alcune impostazioni speciali per dispositivi Android che servono per evitare che l'applicazione venga chiusa mentre essa è in

background oppure che l'utilizzo dei dati venga limitato. Questo può essere utile, se, ad esempio, l'applicazione sta tracciando la posizione del dispositivo anche mentre è in background.

I metodi per la gestione della batteria sono i seguenti:

- *app.device.preferences.isIgnoringBatteryOptimizations*: restituisce *true* se l'applicazione non è sottoposta a restrizioni mentre essa è in background.
- *app.device.preferences.ignoreBatteryOptimizations*: richiede al sistema di evitare le restrizioni sull'uso in background. A sua volta il sistema richiede un consenso all'utente. Viene restituita una stringa che contiene lo stato del comando.
- *app.device.preferences.displayOptimizationsMenu*: mostra una pagina del sistema operativo per consentire all'utente di modificare le impostazioni di gestione della batteria.

I metodi per la gestione del consumo di dati su rete cellulare sono i seguenti:

- *app.device.preferences.isIgnoringDataSaver:* l'applicazione non ha restrizioni sull'utilizzo della rete cellulare.
- *app.device.preferences.displayDataSaverMenu*: mostra una pagina del sistema operativo per consentire all'utente di modificare le impostazioni di gestione della rete cellulare.

Pdf

Il plugin *Pdf* permette di generare documenti PDF a partire file HTML e può essere usato localmente nei device, oppure nelle applicazioni installate nel cloud. Questo plugin viene implementato tramite: cordova-pdf-generator.

È importante comprendere che la conversione di HTML in PDF avviene da parte di una libreria nativa che è diversa nei vari contesti di utilizzo:

- *Generazione da server in cloud*: viene utilizzata la libreria puppeteer, che, a sua volta, usa un browser chrome headless per eseguire la conversione. Ci si devono aspettare i medesimi risultati della stampa in PDF del file HTML da parte di Google Chrome per desktop.
- *Generazione da dispositivo Android*: viene utilizzato un plugin nativo che, a sua volta, utilizza una webview Android per effettuare la conversione. Ci si devono aspettare i medesimi risultati della stampa in PDF del file HTML da parte di Google Chrome per Android.
- *Generazione da dispositivo iOS*: viene utilizzato un plugin nativo che, a sua volta, utilizza una webview iOS per effettuare la conversione. Ci si devono aspettare i medesimi risultati della stampa in PDF del file HTML da parte di Safari per iOS.

I metodi per generare il PDF sono i seguenti:

- *app.device.pdf.fromData(stringaHTML, opzioni)*
- *app.device.pdf.fromURL(url, opzioni)*

Questi metodi sono disponibili anche in ambiente browser (online), in quanto, in quel caso, la generazione avviene nel cloud. Non sono invece disponibili nelle PWA.

Entrambi i metodi restituiscono un oggetto *file* che punta al PDF generato, oppure *undefined* in caso di errore. Le opzioni permettono di specificare alcuni parametri del PDF come le dimensioni della pagina e l'orientazione.

Se si utilizza la variante che genera il PDF in funzione della stringa passata come parametro, occorre tenere presente che le risorse puntate devono essere presenti nel file system dell'applicazione e il loro percorso deve essere relativo alla cartella *files*. Ad esempio, un tag *img* potrà essere inserito così:

```
<img src='files/folder/image.jpg'>
```

Un esempio di codice che stampa e visualizza un PDF relativo ad una stringa HTML può essere il seguente:

```
let htmlString = "<b>Hello</b> world";
let pdfFile = yield app.device.pdf.fromData(htmlString);
yield app.open(pdfFile.publicUrl);
```

Un esempio d'uso di questo plugin è presente nel progetto di esempio: Plugins Design Patterns.

Conversione di contenuti generati tramite JavaScript

Il plugin per la generazione dei PDF è in grado di convertire documenti HTML che contengono codice JavaScript, come ad esempio i grafici o le mappe di Google. Se quindi è richiesta una formattazione particolare, è possibile inserire anche un proprio codice JavaScript che andrà in esecuzione al momento della conversione in PDF e lavorerà nel DOM descritto dal file HTML, come se esso fosse in esecuzione in una finestra browser dedicata.

Risoluzione dei problemi

Nel caso in cui il plugin generi un PDF diverso da quanto atteso, occorre testare che il medesimo file sia convertito correttamente usando il browser relativo al contesto, e cioè:

1) Google Chrome per la generazione nel cloud.
2) Google Chrome for Android per la generazione su dispositivi Android.
3) Safari for iOS per la generazione su dispositivi Apple.

È possibile anche utilizzare browser simili, come Safari per Mac o Google Chrome desktop nel caso Android, ma ci potrebbero comunque essere delle differenze.

Per ottenere un risultato identico nei vari casi si consiglia di utilizzare le direttive css-print e modificare l'HTML fino ad ottenere un rendering uniforme.

Generazione solo Cloud

Si ricorda che tramite WebAPI o DO Remota è possibile inviare dal device un comando remoto ad un server nel cloud che richiede la generazione del PDF, e poi scaricare il documento lato client per stamparlo o mostrarlo a video. Questa soluzione permette anche

di archiviare il PDF generato per referenza futura o per renderlo disponibile anche dall'applicazione web.

Se questa modalità è possibile, si ha l'ulteriore vantaggio di dover testare la generazione del PDF solo nell'ambiente cloud, che è quello più prevedibile in quanto utilizza lo stesso codice di conversione di Google Chrome desktop.

Status Bar e Keyboard

Il plugin *StatusBar* controlla alcuni aspetti della status bar del dispositivo. Il metodo più importante è *app.device.statusBar.overlaysWebView(true)* che permette all'interfaccia utente del dispositivo di sovrapporsi alla status bar, potendo quindi controllare anche quella parte di schermo.

Questo metodo va utilizzato nell'evento *app.onStart*, in modo che l'interfaccia dell'applicazione sia già pronta quando viene nascosto lo splash screen, cioè pochi istanti dopo che l'evento *onStart* è stato notificato.

Il plugin *Keyboard* aiuta il framework nella gestione della tastiera nativa del dispositivo. Espone anche alcuni metodi importanti come:

- *app.device.keyboard.copy(text)*: inserisce il testo nella clipboard del device,
- *app.device.keyboard.paste()*: restituisce il contenuto della clipboard. L'utilizzo di questo metodo viene segnalato all'utente dal sistema operativo, perché il suo uso indiscriminato viene considerato una violazione della privacy.

Questi metodi sono disponibili anche in ambiente browser.

Questi plugin vengono implementati tramite: cordova-plugin-statusbar e cordova-plugin-ionic-keyboard.

Sqlite

Il plugin *Sqlite* viene utilizzato dal framework per gestire database locali al dispositivo quando non è disponibile l'ambiente WebSQL. Per impostazione predefinita, il plugin viene utilizzato in ambiente iOS, mentre per Android e per le PWA viene usato WebSQL perché garantisce migliori performance.
In ambiente iOS è possibile utilizzare alcuni metodi di questo plugin per effettuare un debug di basso livello dell'utilizzo del database. I metodi sono i seguenti:

- *app.device.sqlite.logStart*: inizia il log del database.
- *app.device.sqlite.logStop*: termina il log del database.
- *app.device.sqlite.logClear*: svuota il log del database.
- *app.device.sqlite.logGet*: recupera il log del database.

Il log viene mantenuto solo in memoria e non deve essere lasciato attivo per lungo tempo per non utilizzare troppa memoria del dispositivo.

Suggerimenti per migliorare le performance di accesso ai dati

Alcuni suggerimenti sono validi per ogni tipo di database:

- Utilizzare sempre indici legati alle foreign key per l'accesso veloce padre-figli. Per impostazione predefinita, le foreign key generano sempre l'indice. Non si consiglia di modificare questa impostazione.
- Utilizzare transazioni esplicite in caso di esecuzione di query o modifiche ai dati multiple. Occorre anche tenere conto anche della gestione delle transazioni da parte dei documenti.
- Preferire l'estrazione di dati correlati all'interno di un'unica query. L'esecuzione di query multiple, anche se più semplici, è sempre peggiore a livello di performance.
- Non effettuare mai query o caricamento di documenti negli eventi *onRowComposition* delle liste. Recuperare i dati necessari all'interno della query principale, anche aggiungendo al documento proprietà derivate da altri documenti.

I seguenti suggerimenti sono validi in particolare nel caso di utilizzo del plugin *Sqlite*:
- Le modifiche di dati multiple devono essere inserite in una transazione esplicita.
- Eseguendo modifiche multiple ai dati (insert, update o delete), esse non devono mai essere intercalate con query di lettura dati.

La ragione di questa impostazione risiede nel fatto che le chiamate dall'ambiente JavaScript a quello nativo richiedono tempo (diversi millisecondi) per il passaggio di dati da un contesto all'altro. Questo tempo non impegna il dispositivo in operazioni complesse, ma deve comunque passare perché il comando venga eseguito.

Il driver nativo del framework di Instant Developer, se è aperta una transazione in modo esplicito, invia le istruzioni di modifica dati al plugin, ma non attende il loro completamento prima di far proseguire l'esecuzione del programma. In questo modo il tempo di attesa del ponte JavaScript-nativo non incide più di tanto, perché tutte le istruzioni vengono accodate contemporaneamente e passano al lato nativo tutte insieme.

Questo accodamento deve essere interrotto in due casi:
- Se la transazione viene confermata (commit) perché in questo caso occorre prima attendere i risultati di tutti gli statement di modifica. Se solo uno di questi dà errore, infatti l'intera transazione verrà annullata invece che confermata e l'istruzione di conferma lancerà un'eccezione.
- Se viene eseguita una query, occorre attendere il risultato di tutte le istruzioni di modifica prima di poter recuperare il risultato della query.

Questo plugin viene implementato tramite: cordova-sqlite-plugin.

SocialSharing

Questo plugin permette di condividere contenuti testuali, link o file con altri utenti o applicazioni tramite il plugin di condivisione nativo del dispositivo.

Il metodo principale per la condivisione è *app.device.socialSharing.shareWithOptions* che ammette un parametro di tipo oggetto in cui è possibile specificare le seguenti proprietà:
- *message*: il messaggio di testo da condividere.
- *subject*: l'oggetto del messaggio, nel caso di condivisione via email.

- *files*: un array di stringhe che contiene i percorsi assoluti dei file da condividere, sia locali che remoti.
- *url*: un link da condividere.

Tramite la videata nativa di condivisione, l'utente potrà inviare i dati ad altre applicazioni di comunicazione, oppure usare comandi del sistema operativo come, copia, stampa eccetera.

È possibile attivare anche la condivisione diretta via email tramite il metodo *shareViaEmail*. Per sapere se la condivisione via email è supportata, si può chiamare il metodo *canShareViaEmail*.

Infine questo plugin contiene anche il metodo *checkAvailability* che consente di sapere se nel dispositivo sono presenti altre app installate. Il parametro da passare a questo metodo è l'URI schema dell'app nel caso iOS e il nome del package dell'app nel caso Android.

Ad esempio, per verificare se sul dispositivo è installata l'applicazione *Twitter*, su iOS bisognerà passare come parametro la stringa "twitter://", mentre su Android bisognerà passare la stringa "com.twitter.android".

Su iOS la lista delle app (e dei relativi servizi aggiuntivi) di cui è possibile conoscere l'installazione è limitata alle seguenti: *Twitter* (twitter://), *Whatsapp* (whatsapp://), *Facebook* (fb://, fbapi://, fbauth2://, fbshareextension://), *Messenger* (fb-messenger://, fb-messenger-api://), *Linkedin* (linkedin://, linkedin-sdk://, linkedin-sdk2://), *Skype* (skype://), *Google Maps* (comgooglemaps://).

Questo plugin viene implementato tramite: cordova-plugin-x-socialsharing e cordova-plugin-appavailability. Un esempio d'uso di questo plugin è presente nel progetto di esempio: Plugins Design Patterns.

Speech

Il plugin *speech* mette a disposizione metodi per consentire al dispositivo di riconoscere un input vocale da parte dell'utente (riconoscimento vocale) e di generare un file audio a partire da un testo (sintesi vocale). Entrambi questi metodi sono disponibili anche in ambiente browser.

Un esempio d'uso di questo plugin è presente nel progetto di esempio: Plugins Design Patterns. Questo plugin viene implementato tramite: cordova-plugin-tts e cordova-plugin-speechrecognition.

Verifiche e permessi

Per sapere se il riconoscimento vocale è disponibile, è possibile chiamare il metodo: *app.device.speech.isRecognitionAvailable*.

Prima di poter utilizzare questa funzionalità è necessario richiedere i permessi all'utente. Per sapere se i permessi sono già stati chiesti è possibile usare il metodo: *app.device.speech.hasPermission*. Questo metodo ammette come primo parametro un

valore booleano che consente di controllare l'accesso solo al microfono (true) oppure all'intero sistema di riconoscimento vocale (false o assente).

Per richiedere il permesso è presente il metodo *app.device.speech.requestPermission* che ammette un parametro con la stessa logica del precedente.

Riconoscimento vocale

Per iniziare l'operazione di riconoscimento è disponibile il metodo *app.device.speech.startListening* che permette di specificare una serie di opzioni che è possibile consultare nella documentazione in linea.

Su dispositivi iOS il metodo ritorna immediatamente, mentre su dispositivi Android e su browser esso ritorna quando l'utente fa una pausa. In questo caso, il metodo restituisce un array con il testo riconosciuto.

Nel caso iOS invece, il riconoscimento rimane attivo fino a che non si chiama il metodo *app.device.speech.stopListening*. A questo punto verrà notificato l'evento *app.device.speech.onSpeechRecognized* passando come parametro l'elenco dei testi riconosciuti con la percentuale di attendibilità.

Su iOS questo evento viene notificato anche durante il riconoscimento, se viene attivata l'opzione *showPartial*. Nel caso browser viene sempre notificato anche durante il riconoscimento. Utilizzando questo evento e un timeout è possibile gestire uno stop temporizzato anche per iOS, chiamando il metodo *stopListening* se non ci sono nuovi risultati parziali entro un certo tempo limite.

Sintesi vocale

Per far sì che il dispositivo o il browser "legga" un determinato testo è possibile usare il metodo *app.device.speech.speak* passando come parametro il testo da leggere o un oggetto che contiene sia il testo che i parametri della voce, descritti nella documentazione in linea.

Se il metodo viene sincronizzato tramite *yield*, come avviene per default, esso non torna fino a quando la frase non viene completamente letta dal sistema. Se si desidera proseguire l'esecuzione prima del termine della frase, è sufficiente rimuovere la parola chiave *yield*. A questo punto, se si desidera, sarà possibile interrompere la lettura prima del termine tramite il metodo *app.device.speech.stopSpeak*.

Touchid

Il plugin *touchid* permette di salvare chiavi di accesso o altri valori in uno storage protetto e di poterli recuperare solo a fronte di una verifica biometrica come l'impronta digitale o il riconoscimento facciale.

È possibile verificare se il dispositivo supporta il riconoscimento biometrico tramite il metodo: *app.device.touchid.isAvailable*.

La gestione delle chiavi di accesso avviene con i seguenti metodi:

- *app.device.touchid.save*: salva una chiave passando il nome e il valore.
- *app.device.touchid.delete*: cancella una chiave per nome.
- *app.device.touchid.has*: verifica che la chiave con il nome dato sia presente.
- *app.device.touchid.verify*: estrae il valore di una chiave dato il suo nome, solo a fronte di una verifica biometrica eseguita con successo.

Questo plugin viene implementato tramite: cordova-plugin-keychain-touch-id. I metodi del plugin funzionano anche nel browser, ma solo con scopi di test: le chiavi rimangono solo in memoria e vengono perse alla chiusura della sessione; inoltre le verifiche biometriche non vengono effettuate e simulano sempre un successo.

Vibration e Haptic

Il plugin *vibration* emette una vibrazione di lunghezza costante per iOS e di lunghezza variabile in funzione del parametro passato per Android. Un esempio di questo è:

```
app.device.vibration.vibrate(1000);
```

Il plugin *haptic* permette di usare una serie di feedback tattili particolari. Il numero dei feedback è piuttosto ampio su iOS e limitato per Android. La documentazione del metodo *feedback* elenca i vari tipi e stili di feedback disponibili. Vediamo un esempio:

```
app.device.haptic.feedback("boom", "weak");
```

Sia il plugin *vibration* che *haptic* non hanno effetto se usati in ambiente browser.

Questi plugin vengono implementati tramite: cordova-plugin-vibration, cordova-plugin-taptic-engine e device-feedback.

OIDC

Il plugin *oidc* consente di integrare un flusso di autenticazione OpenID Connect basato su OAuth 2.0 all'interno dell'applicazione. In particolare permette di effettuare il login tramite un provider OIDC compatibile, di ottenere e rinnovare token di accesso, leggere le informazioni dell'utente autenticato e terminare la sessione.

Questo plugin è disponibile dopo aver importato nel progetto il package *OpenIDConnect* e può essere utilizzato sia per le applicazioni web che per quelle mobile.

Viene implementato tramite: cordova-plugin-oidc-basic.

Configurazione iniziale

Il plugin richiede un identity provider (idP) compatibile con OpenID Connect, come ad esempio Auth0, Keycloak (open source, on-premise), Google, Microsoft Azure AD, Okta, Amazon Cognito, eccetera. All'interno dell'idP sarà necessario creare un'App Registration, cioè bisognerà registrare la nostra applicazione per fare in modo che l'idP possa gestire i flussi di autenticazione che vengono avviati dall'app.

La configurazione iniziale del plugin consiste nell'impostare le proprietà *app.device.oidc.domain, app.device.oidc.clientId* e *app.device.oidc.clientSecret*, che permettono di definire qual è l'idP che gestirà il flusso di autenticazione e qual è il *client* che vuole autenticarsi, in questo caso la nostra app. Ecco un esempio:

```
app.device.oidc.domain = "https://your-oidc-provider.com";
app.device.oidc.clientId = "your-client-id";
app.device.oidc.clientSecret = "your-client-secret";
```

Il valore da impostare per queste proprietà può essere recuperato all'interno dell'App Registration definita sull'idP. Questo codice può essere scritto nell'evento *onStart* dell'applicazione oppure nell'evento *onLoad* della videata dalla quale viene avviato il flusso di autenticazione.

È presente anche la proprietà *app.device.oidc.redirectUrl* che rappresenta l'url di reindirizzamento dopo l'autenticazione. Questa proprietà è precalcolata e, a meno di casi particolari, non è necessario modificarla. Il suo valore viene calcolato in base al tipo di applicazione e all'ambiente in cui viene eseguita:

- nel caso di app offline eseguita su shell nativa è: *indeoidc://callback*
- nel caso di app online e di app offline su browser è: *[serverUrl]/app/client/objects/base/shellEmulator/plugins/oidc/oidcCallback.html*

Questo valore deve essere impostato nell'idP all'interno della configurazione dell'App Registration, in una voce chiamata spesso *Callback URLs* o simile.

Flusso di autenticazione

Un flusso di autenticazione OIDC è composto da 3 step:

1. richiesta di autenticazione con l'idP che risponde aprendo la pagina di login;
2. richiesta dei token di accesso usando l'authorization code e il code verifier ottenuti nel primo step;
3. richiesta dei dati dell'utente usando i token di accesso ottenuti nel secondo step.

I metodi *app.device.oidc.authorize, app.device.oidc.getToken* e *app.device.oidc.getUserInfo* implementano i 3 step del flusso di autenticazione.

Il codice che implementa un flusso di autenticazione completo è quindi il seguente:

```
// 1) First step: require authorization and showing login page
let authorization = yield app.device.oidc.authorize();
//
// 2) Second step: use authorization code to ask for token
let tokenData = yield app.device.oidc.getToken({authorizationCode:
authorization.authorizationCode, codeVerifier:
authorization.request.codeVerifier});
//
// 3) Third step: use token data to request user info
let userInfo = yield app.device.oidc.getUserInfo(tokenData);
```

Infine il metodo *app.device.oidc.endSession* permette di terminare la sessione utente dell'idP, invalidando i token di accesso.

Un esempio d'uso di questo plugin è presente nel progetto di esempio: Plugins Design Patterns.

Test delle applicazioni nei launcher

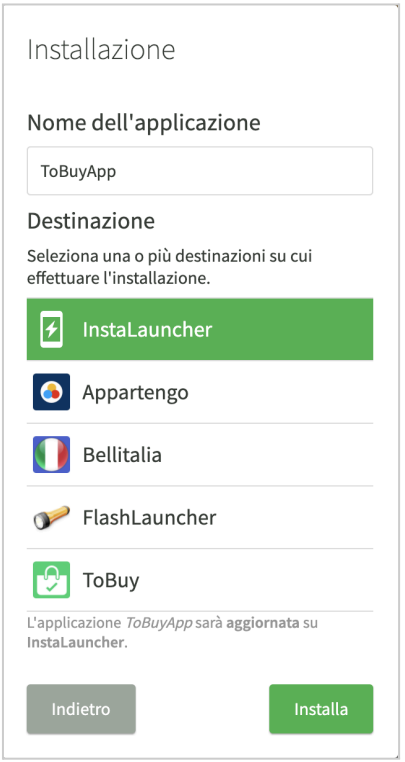
Dopo aver testato l'applicazione in anteprima su InstaLauncher come indicato nel paragrafo Anteprima dell'applicazione in un launcher, è necessario procedere ad un test più approfondito prima di iniziare la distribuzione tramite app store.

A tal fine InstaLauncher permette di installare localmente applicazioni sviluppate con Instant Developer Cloud così da poterle testare come quando saranno in produzione. È inoltre possibile distribuire un'applicazione via email ad un numero limitato di persone, che potranno così eseguire l'alpha test dell'applicazione.

L'installazione delle applicazioni su InstaLauncher avviene tramite la funzione di installazione della console. I passaggi sono i seguenti:

1) Scegliere la voce *Installazioni* del menu di progetto.
2) Cliccare il pulsante *+Installa*. Nel popup delle opzioni di installazione, cliccare sul pulsante *Launcher* e quindi su *Avanti*.
3) Inserire le ulteriori informazioni richieste e cliccare su *Avanti*.

4) Nella lista delle destinazioni, scegliere la voce *InstaLauncher*, come mostrato nell'immagine seguente, poi cliccare su *Installa*.



Al termine della compilazione, sarà possibile ricevere l'applicazione nel proprio smartphone lanciando InstaLauncher ed eseguendo il login con le stesse credenziali della console di Instant Developer Cloud.

Dopo qualche istante l'applicazione apparirà nell'elenco di quelle installate. Se si è già effettuato il login, l'applicazione verrà aggiornata nel momento in cui InstaLauncher viene avviato. Se si desidera forzare l'aggiornamento, usare il pulsante con l'icona di aggiornamento nella videata delle applicazioni installate. Per cancellare un'applicazione installata, eseguire uno swipe verso destra e cliccare il pulsante con l'icona di cancellazione.

Dopo aver lanciato l'installazione su InstaLauncher, è possibile condividerla con un numero limitato di persone accendendo alla pagina *App e Dati* del menu di progetto nella console.

Espandendo la riga della lista corrispondente alla build che si desidera condividere, apparirà il link *Condividi*. A questo punto sarà possibile inserire gli indirizzi email delle persone che devono ricevere l'applicazione.

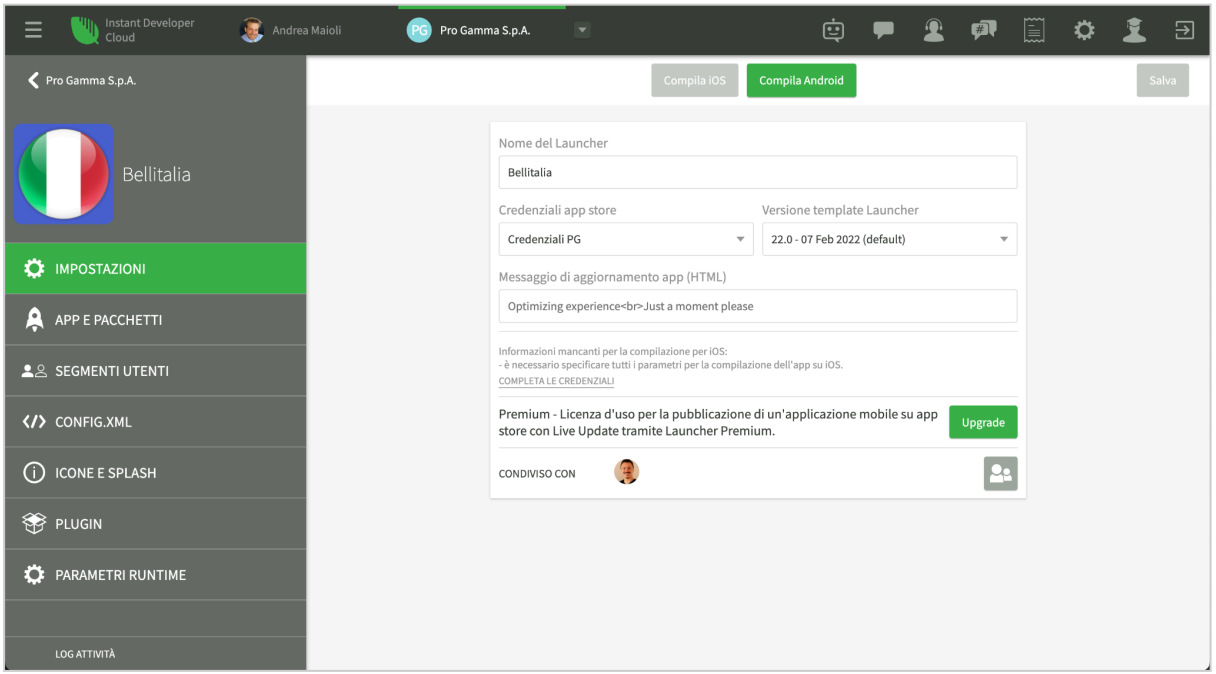
Agli indirizzi email indicati arriverà un'email di notifica della condivisione nella quale è presente un link che, se aperto da dispositivo mobile in cui è già presente InstaLauncher, ne avvia il download e l'installazione.

Pubblicazione sugli store

Al termine dei test l'applicazione è pronta per essere distribuita e pubblicata sugli store di Apple e di Google. Per completare con successo questa operazione occorre creare un launcher standard o premium, configurarlo ed infine ottenere le credenziali e i certificati necessari alla pubblicazione.

Configurazione del launcher

Nell'immagine seguente vediamo il menu di configurazione del launcher. Vediamo ora quali dati devono essere inseriti in ognuna delle sezioni.



Impostazioni

La sezione delle impostazioni è quella mostrata nell'immagine precedente. Devono essere inseriti i seguenti dati:

- Nome del launcher: è il nome con il quale viene mostrato il launcher nella console di Instant Developer.
- Versione template launcher: la versione del launcher che verrà usata per creare i pacchetti per gli store.
- Messaggio di aggiornamento: nel caso di un launcher Premium, è il messaggio che l'utente vedrà al momento dell'aggiornamento dell'applicazione. Se non viene specificato, verrà usato il messaggio standard.

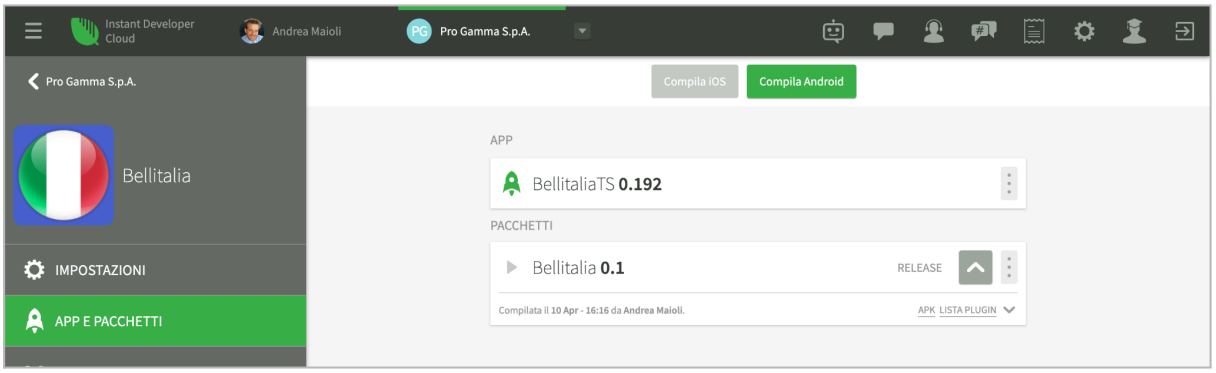
Se la versione del launcher viene lasciata vuota, verrà sempre utilizzata l'ultima versione disponibile, che corrisponde all'ultima versione rilasciata del framework di Instant Developer Cloud.
Si consiglia quindi di selezionare sempre l'ultima versione corrispondente a quella del framework in uso e aggiornarla in modo corrispondente. Si ricorda che App Store e Google

Play costringono gli sviluppatori a rimanere aggiornati all'ultima versione dei loro framework e strumenti di sviluppo, quindi se si continua ad utilizzare un template datato, dopo un certo tempo, che di solito è un anno, non si potrà più inviare l'applicazione agli store.

Nota bene: l'aggiornamento del template consiste nell'utilizzo di un nuovo set di plugin, di una nuova versione di Cordova, di una nuova versione delle librerie Android e iOS, ed infine di una nuova versione degli ambienti di compilazione. Per quanto l'obiettivo di Instant Developer sia quello di minimizzare il numero di breaking change al cambiamento di versione, non è possibile garantire che cambiando il template l'applicazione non modifichi il proprio comportamento o che non si ottengano errori di compilazione o di runtime. Dopo aver cambiato template, si consiglia quindi di leggere attentamente le note di rilascio delle versioni corrispondenti di Instant Developer Cloud e di testare nuovamente l'applicazione prima di distribuirla agli utenti.

App e Pacchetti

Questa sezione indica la build installata nel launcher e i pacchetti inviati agli store. Per installare una build nel launcher occorre eseguire la procedura di installazione su launcher, come indicato nel paragrafo Test delle applicazioni nei launcher.



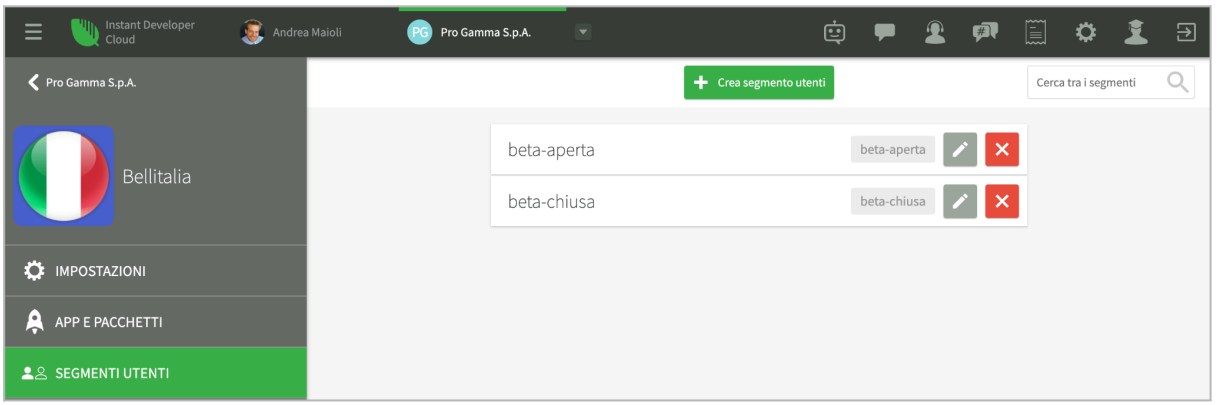
Se il launcher è di tipo Premium, ogni volta che si installa una nuova build nel launcher, essa verrà automaticamente installata in tutti i dispositivi e questo processo è praticamente istantaneo. Se invece il launcher è standard, si dovrà passare da una nuova pubblicazione sugli store e poi aspettare che i dispositivi si aggiornino. In questo caso possono passare giorni o settimane prima di poter aggiornare tutti i dispositivi.

Per compilare ed inviare ad App Store oppure a Google Play, occorre cliccare i pulsanti *Compila iOS* o *Compila Android*. Prima di poter compilare o inviare, tuttavia, occorre completare tutta la procedura di configurazione e di preparazione indicata nelle pagine seguenti.

Nella lista dei pacchetti è possibile vedere quali pacchetti sono stati compilati, scaricare l'APK o la IPA e vedere quali plugin nativi contiene ogni pacchetto. In caso di errori di compilazione o di invio agli store, nella lista appare un link che permette di vedere i dettagli dell'errore.

Segmenti utenti (launcher premium)

Un segmento utenti permette di mettere in produzione build diverse per gruppi di utenti diversi, in caso di launcher premium. In questo modo, ad esempio, è possibile effettuare un beta test di una nuova versione senza influenzare gli utenti della versione attuale dell'app.



Per aggiungere un segmento utenti occorre cliccare il pulsante + *Crea segmento utenti* e scegliere un nome univoco che servirà per identificare il segmento utenti all'interno dell'applicazione.

A questo punto sarà possibile installare nel launcher versioni diverse per segmenti utenti diversi. L'associazione avviene nel momento dell'installazione, come mostrato nell'immagine seguente: al momento della scelta del launcher in cui installare la build, apparirà la lista dei segmenti utenti e sarà possibile associare la build ai vari segmenti.

Occorre infine predisporre un metodo per associare ogni dispositivo ad un determinato segmento. Questo avviene all'interno del codice dell'applicazione valorizzando la proprietà *app.device.userSegment.* Si consiglia di farlo dopo aver identificato l'utente in modo da capire a quali segmenti appartiene. Dopo aver valorizzato questa proprietà, se l'applicazione necessita di essere aggiornata, questo avverrà immediatamente.

Config.xml

Questa sezione contiene i dati che andranno a comporre il file config.xml del progetto Cordova utilizzato per compilare l'applicazione. I campi necessari sono i seguenti:

- Bundle: è l'id univoco dell'applicazione. Si consiglia di crearlo a partire dal nome della società e dell'applicazione usando una notazione DNS inversa. Ad esempio *com.società.applicazione.*
- Nome: è il nome con cui verrà mostrata l'applicazione una volta installata nei dispositivi.

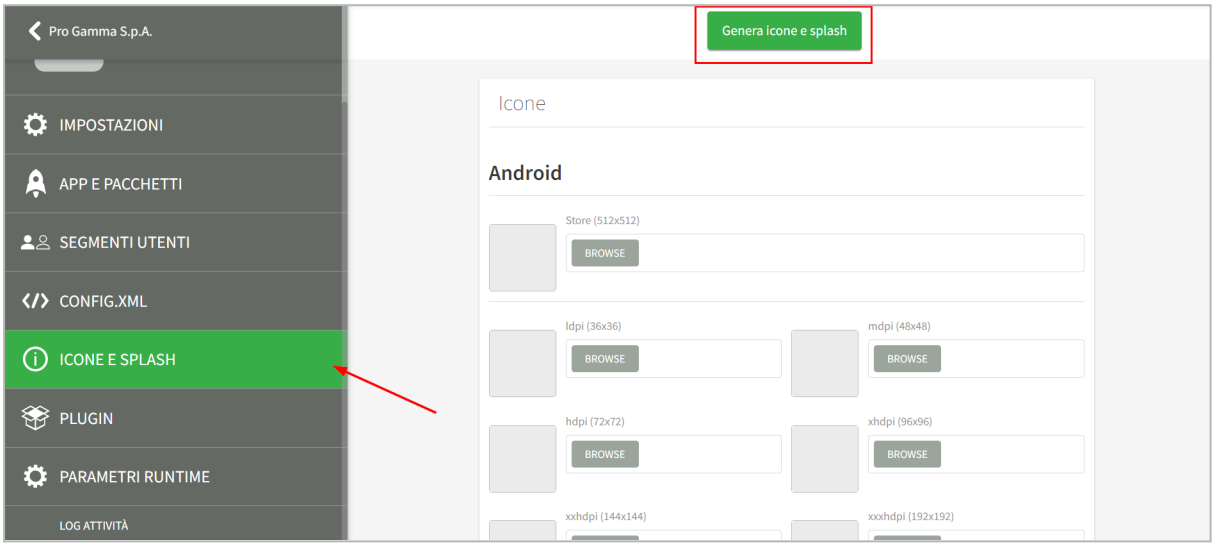
- Versione: è la versione dell'applicazione.
- Dispositivo target iOS: indica il tipo di dispositivi per i quali l'applicazione sarà disponibile.
- Codice versione APK corrente: è la versione dell'applicazione nel Google Play Store. Può essere inizializzato a 1.
- Versione di SDK Android minima: la versione minima di Android che permette l'installazione dell'app. Può essere lasciato vuoto.

I dati delle sezioni Android ed iOS devono essere inseriti come indicati nelle pagine seguenti per poter procedere alla compilazione e all'invio ad app store.

Icone e Splash

La sezione Icone e Splash contiene la lista di tutte le immagini che devono essere incluse nel pacchetto di installazione e che quindi sono necessarie alla pubblicazione. È possibile caricare le varie immagini una alla volta, oppure si può utilizzare la funzione di generazione automatica che parte da una sola copia dell'icona per derivare tutte le altre risorse.

Per iniziare la procedura di generazione immagini è sufficiente cliccare il pulsante *Genera icone e splash.*

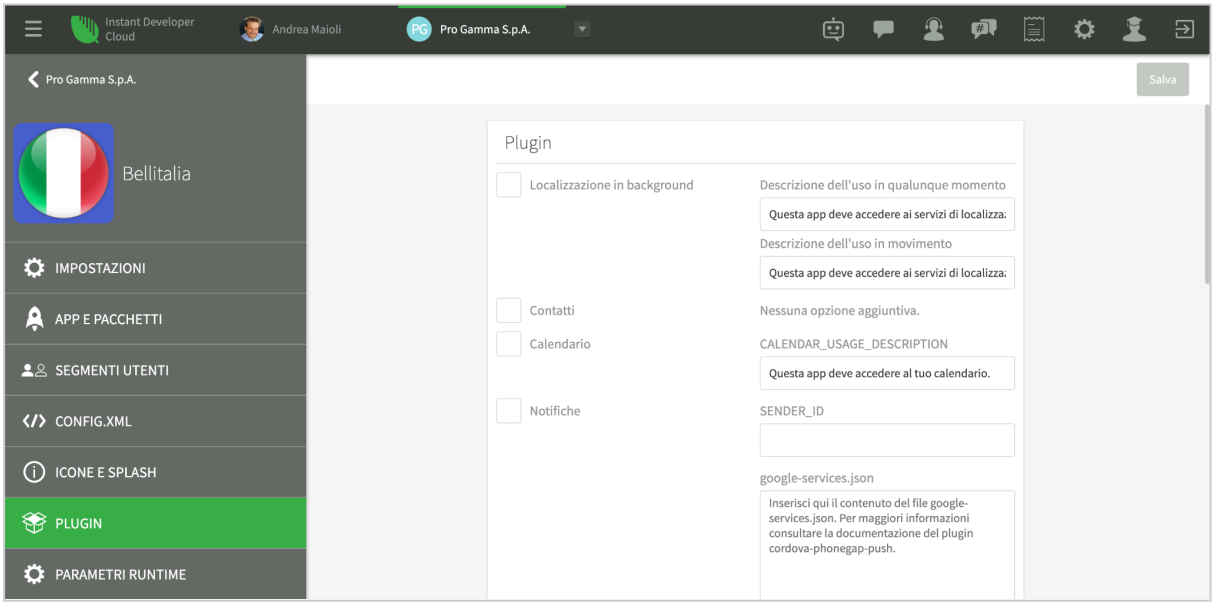


Si aprirà un popup che permetterà di caricare l'icona dell'applicazione e lo splash screen in formato *png*. Facoltativamente, è possibile caricare anche le immagini specifiche per lo splash screen in modalità portrait e per quello in modalità landscape. A questo punto è possibile cliccare il pulsante *Genera icone e splash* per creare le immagini.

Se si dispone solo dell'immagine dell'icona e non di quella dello splash screen, è possibile indicare al sistema di generare in automatico lo splash screen a partire dall'icona.
In questo caso è necessario caricare solo l'immagine relativa all'icona, poi cliccare il pulsante *Imposta Splash*, impostare i parametri relativi ai colori e ai font e infine cliccare *Genera icone e splash*.

Plugin

La sezione Plugin contiene l'elenco dei plugin nativi che possono essere inclusi nell'applicazione e per alcuni di essi permette di configurare le opzioni.



I plugin obbligatori per il funzionamento dell'applicazione vengono sempre inclusi e non sono elencati qui, ma possono essere trovati nella sezione *App e pacchetti*, cliccando sul link *Plugin inclusi* nella lista dei pacchetti.

Si consiglia di deselezionare tutti i plugin che non servono alle funzioni specifiche dell'applicazione per non dover incorrere in problemi di pubblicazione. Occorre tuttavia porre attenzione al fatto che alcuni plugin sono necessari per attivare alcune funzioni applicative: non includendo il plugin SQLite, ad esempio, l'applicazione non potrà scrivere dati nel database locale.

Per i dettagli relativi ai plugin vedi il capitolo: I plugin nativi.

Parametri runtime

La sezione Parametri runtime permette di definire una serie di coppie nome-valore che il launcher riceve direttamente a runtime. L'applicazione può leggere il valore dei parametri tramite la funzione *app.getParameter.* Se un parametro cambia mentre l'app è in esecuzione, viene notificato l'evento app.*onParameterChange*. Occorre tenere presente che l'applicazione riceve il valore aggiornato dei parametri quando viene lanciata oppure quando esce dallo stato di background.

Si noti infine che i parametri di runtime vengono conservati in un file memorizzato nel dispositivo; non si consiglia quindi di usarli per salvare password senza prima criptarle. Si ricorda anche che il plugin TouchID permette di memorizzare informazioni in modo sicuro. Tali informazioni saranno reperibili solo dopo aver verificato la presenza del proprietario del dispositivo.

Configurazioni per gli store

Dopo aver eseguito una prima configurazione del launcher, occorre configurare le schede delle applicazioni negli store in modo da reperire i certificati e le ulteriori informazioni necessarie alla compilazione e pubblicazione delle proprie applicazioni tramite launcher.

Scelta del bundle

Ogni applicazione presente sugli store è rappresentata da un *bundle*, una stringa che identifica in modo univoco l'applicazione. Di solito il bundle è composto usando il nome della società e il nome dell'applicazione in notazione DNS inversa, ad esempio *com.società.applicazione.*

Se si pubblica la propria applicazione sia sull'App Store di Apple che sul Play Store di Google, occorre tenere presente che le app sui due store dovranno avere lo stesso bundle perché la console di Instant Developer Cloud permette di specificarne solo uno.
Occorre inoltre tenere presente che la console di Instant Developer Cloud non accetta bundle che contengono spazi.

Il bundle deve essere impostato nei parametri del launcher e, nel caso di pubblicazione sull'App Store, anche al momento della creazione dell'AppID. Il Play Store di Google, invece, non chiede di specificare esplicitamente il bundle ma lo ottiene in automatico dal pacchetto dell'app caricato sullo store in cui è presente il bundle impostato nel launcher.

Si consiglia quindi di partire dalla pubblicazione su App Store definendo il bundle con le regole indicate sopra; Play Store erediterà lo stesso bundle in fase di pubblicazione.

Preparazione per iOS

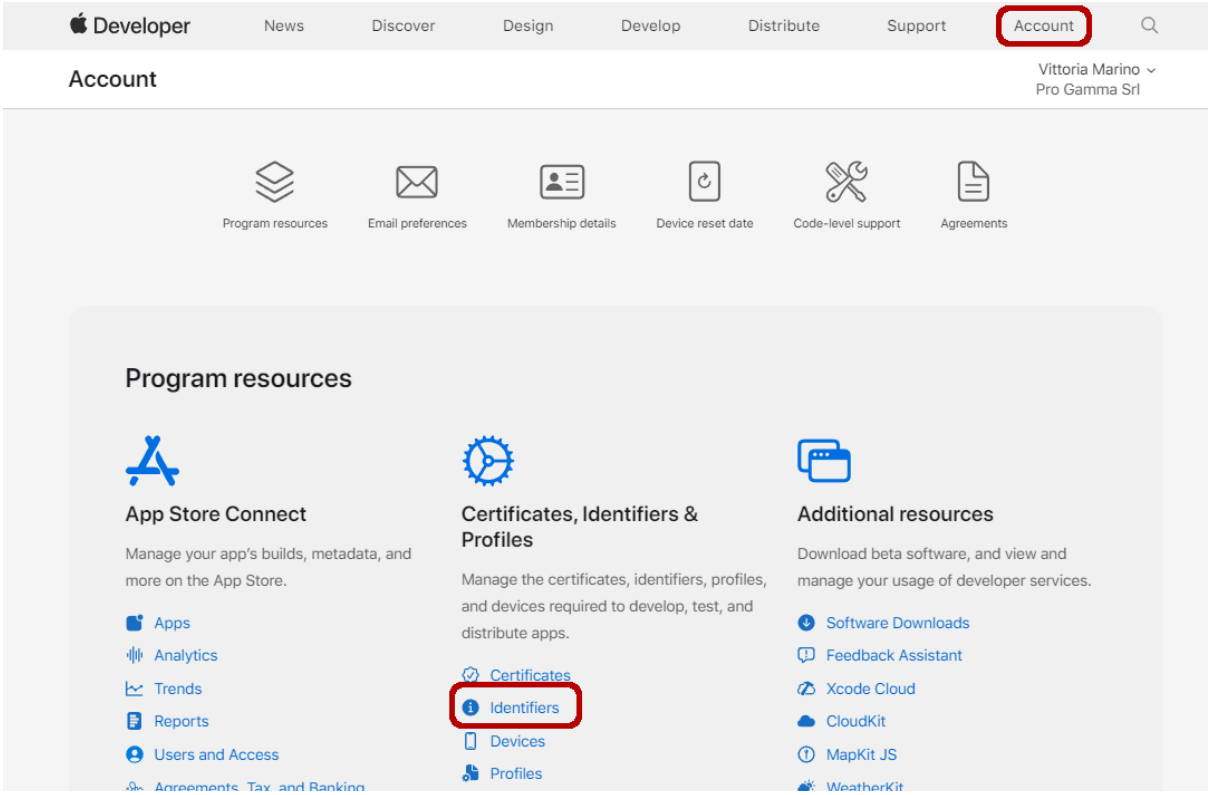
Per ottenere tutti i dati e le credenziali necessarie per la pubblicazione dell'applicazione sull'App Store occorre seguire questi passaggi:

- Creare un AppID
- Creare la scheda dell'app
- Creare i certificati
- Creare una chiave per le app
- Creare il provisioning profile

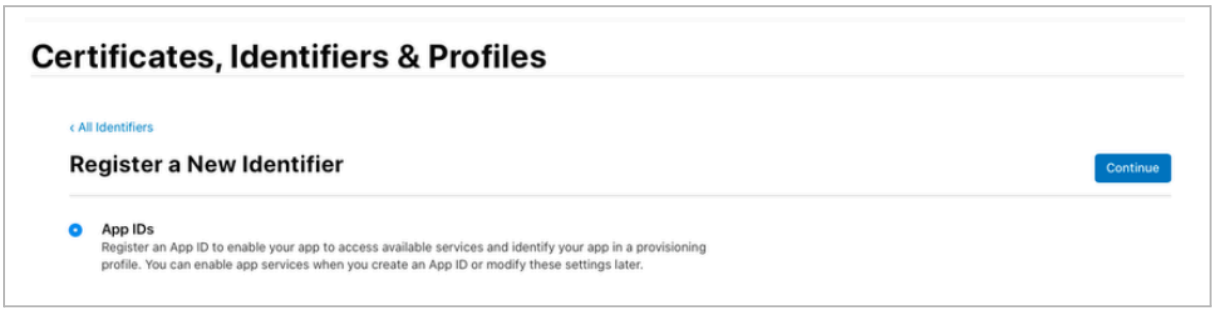
Vediamoli uno a uno.

Creare un AppID

Per ottenere l'identificativo dell'applicazione in App Store, occorre entrare nella [console developer di Apple] e scegliere *Identificatori* dalla lista *Certificati, identificatori e profili*.



Nella testata della lista degli identificatori, cliccare il pulsante "+" per aggiungere un nuovo identificatore, poi selezionare *App ID* e cliccare il pulsante *Continua*.



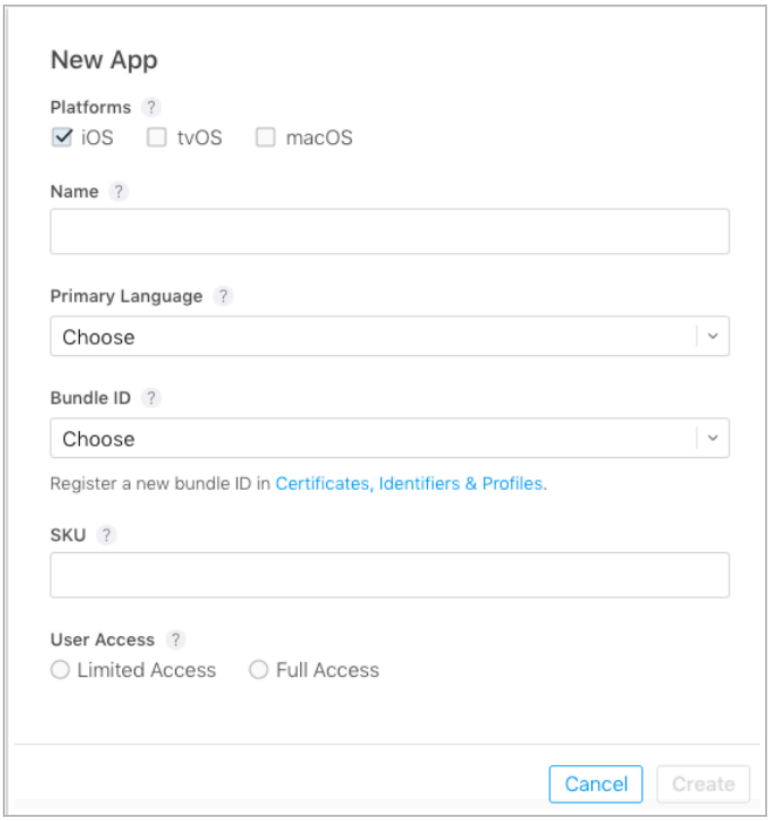
Nella videata successiva occorre scegliere il tipo di applicazione: selezionare *App.*
Nel form successivo occorre inserire il nome dell'app nel campo *Nome*; poi nel campo *Bundle ID* è necessario scegliere l'opzione *Esplicito* ed infine scrivere il bundle dell'applicazione, con le regole esposte nel paragrafo precedente.

Se si desidera inviare notifiche, occorre abilitare il servizio *Push Notification* nell'elenco dei servizi attivati per l'applicazione. Cliccare infine il pulsante *Register* per completare la creazione dell'App ID.

Creare la scheda dell'app

Dopo essere entrati nella console developer di Apple, accedere alla sezione *App Store Connect* e cliccare su *Le mie app*.

Dall'elenco delle applicazioni, cliccare il pulsante "+" in alto a sinistra e selezionare *Nuova app*. Riempire il modulo con le informazioni di base dell'applicazione (nome, lingua, ecc…), selezionare il *Bundle ID* inserito durante la registrazione dell'App ID e poi cliccare su *Crea*.



A questo punto appare la scheda informativa della nuova applicazione nell'app store. Qui occorre fornire i tutti dati richiesti e fare l'upload degli screenshot, che possono essere ottenuti durante la fase di test dell'applicazione su InstaLauncher. Quando tutti i dati sono stati forniti, cliccare il pulsante *Crea* per completare l'operazione.

Per completare questa parte di configurazione occorre accedere alla sezione relativa alle informazioni dell'app, scegliendo la voce informazioni sull'app presente nel menu a sinistra e poi copiare il valore dell'Apple ID.

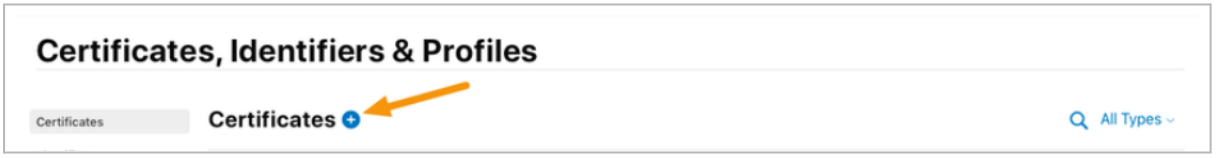
Questo valore dovrà essere impostato durante la configurazione del launcher nel campo App ID che si trova nella sezione iOS della videata che si apre scegliendo la voce di menu *Config.xml*.

Creare i certificati di distribuzione

Per le operazioni necessarie ad ottenere i certificati è possibile utilizzare un sistema open source come openSSL, oppure utilizzare un sistema Mac e seguire questi passaggi:

1. Aprire l'applicazione *Accesso Portachiavi*.
2. Nella barra di sistema presente nella parte alta dello schermo cliccare su *Accesso Portachiavi*, poi scegliere *Assistente certificato* e infine *Richiedi un certificato da un'autorità di certificazione*.
3. Scrivere l'indirizzo email e il nome nei rispettivi campi. Lasciare vuoto il campo *Indirizzo e-mail CA*.
4. Selezionare l'opzione salvata su disco e cliccare il pulsante *Continua*.

A questo punto verrà scaricato il file di richiesta di firma del certificato. Occorre ora spostarsi nella console developer di Apple, cliccare su *Certificati* nella lista *Certificati, identificativi e profili* e, nella videata che si aprirà, cliccare sul pulsante "+" per aggiungere un nuovo certificato.



Scegliere poi l'opzione *iOS Distribution (App Store o Ad Hoc)* e cliccare su *Continua*. Viene ora richiesto di fare l'upload del file di richiesta di firma del certificato creato in precedenza. Cliccare poi su *Continua* per proseguire. È ora possibile scaricare il certificato cliccando sul pulsante *Download*.

Dopo aver scaricato il certificato sul Mac, fare doppio clic sull'icona per installarlo nell'applicazione *Accesso Portachiavi*.

Selezionare ora il certificato dalla lista mostrata dall'applicazione e cliccare su di esso con il pulsante destro del mouse. Scegliere la voce *Esporta <nome del certificato>* dal menu contestuale e poi selezionare il formato *p12*. Verrà richiesta una prima password che verrà legata al certificato, da generare sul momento, e una seconda password che invece è quella usata per accedere all'applicazione *Accesso Portachiavi*. Prendere nota della password del certificato che dovrà essere inserita nella console di Instant Developer Cloud. Dopo aver inserito le password il certificato verrà scaricato.

Se, dopo aver installato il certificato, nell'applicazione *Accesso Portachiavi* compare un messaggio in rosso che avverte che il certificato non è valido, occorre installare il nuovo *IntermediateCertificate*. Per farlo è sufficiente scaricare il certificato da questo indirizzo e fare doppio clic sul file scaricato (maggiori informazioni si trovano qui).

Nota bene: lo stesso certificato di distribuzione può essere utilizzato per compilare e pubblicare più applicazioni; è sufficiente quindi crearne uno solo. Tuttavia, il certificato ha come scadenza un anno, quindi ogni anno occorre ripetere questa operazione.

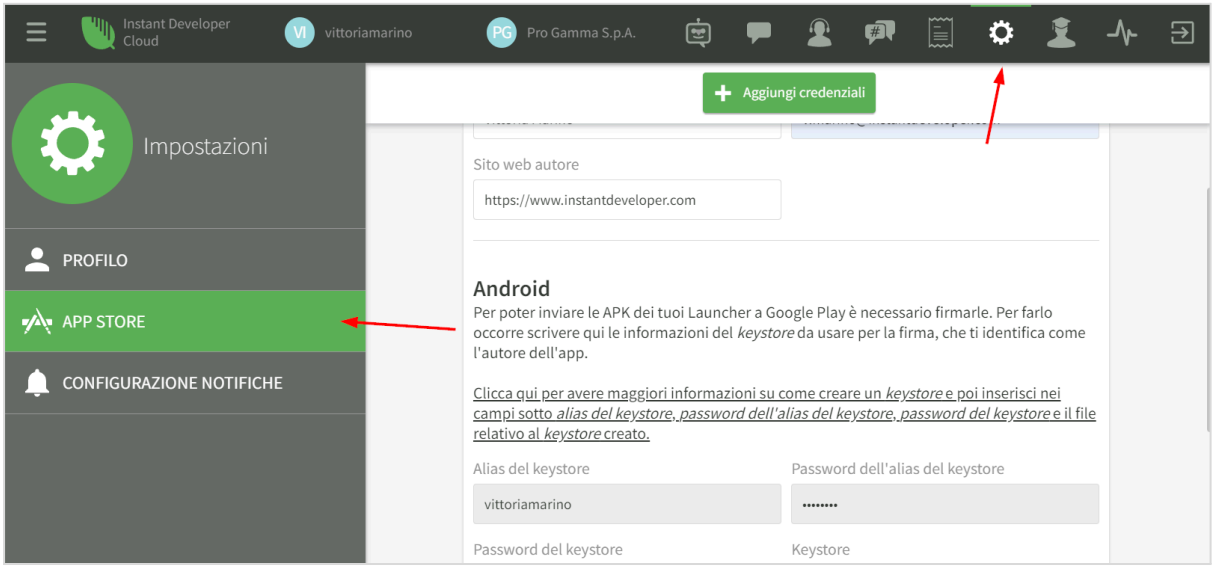
Creare una chiave per la pubblicazione delle app

Se non si dispone già di una chiave per la pubblicazione di app, occorre crearne una nuova per permettere alla console di Instant Developer Cloud di pubblicare l'applicazione.

Per creare una chiave per la pubblicazione di app occorre fare login nella pagina dell'account dell'Apple ID. Nella sezione *Accesso e sicurezza* cliccare sul link *Genera password* sotto *Password specifica per le app*. Dopo aver inserito un nome descrittivo per la password, cliccare su *Crea* per generare la password. A questo punto è possibile copiare la password e salvarla in un posto sicuro.

Impostare le credenziali nella console

Si hanno ora a disposizione tutti i dati da impostare nelle credenziali usate dalla console di Instant Developer Cloud. Le credenziali si trovano nella videata che si apre cliccando il pulsante con il simbolo dell'ingranaggio e poi scegliendo *App Store* dal menu di sinistra.



Se le credenziali non sono ancora state create, occorre cliccare il pulsante + *Aggiungi Credenziali*. Se invece sono già presenti, è sufficiente espanderle cliccando sul pulsante con la freccia verso il basso.

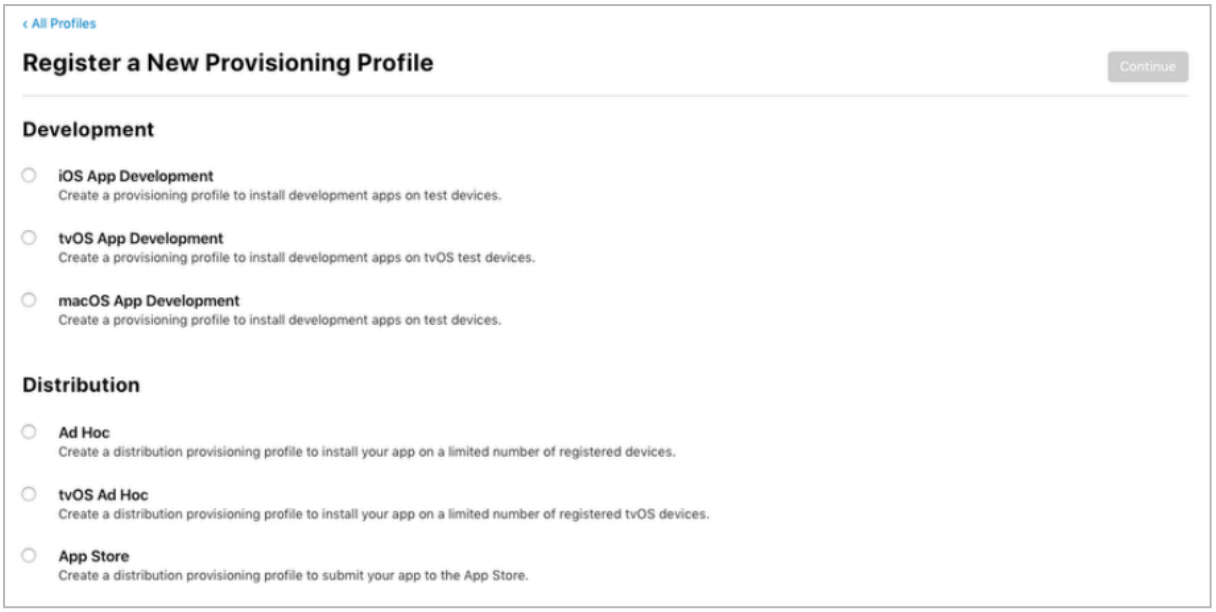
Nella sezione iOS occorre indicare:

- l'email dell'Apple ID con cui si effettua l'accesso alla console Developer Apple.
- la chiave per la pubblicazione di app nel campo password dell'Apple ID.
- la password del certificato di produzione nel campo password del certificato. La password del certificato di produzione è quella generata in fase di esportazione del certificato dall'applicazione *Accesso Portachiavi*.
- Infine, nel quarto campo, occorre caricare il file del certificato in formato *p12* esportato dall'applicazione *Accesso Portachiavi*.

Creazione del distribution provisioning profile

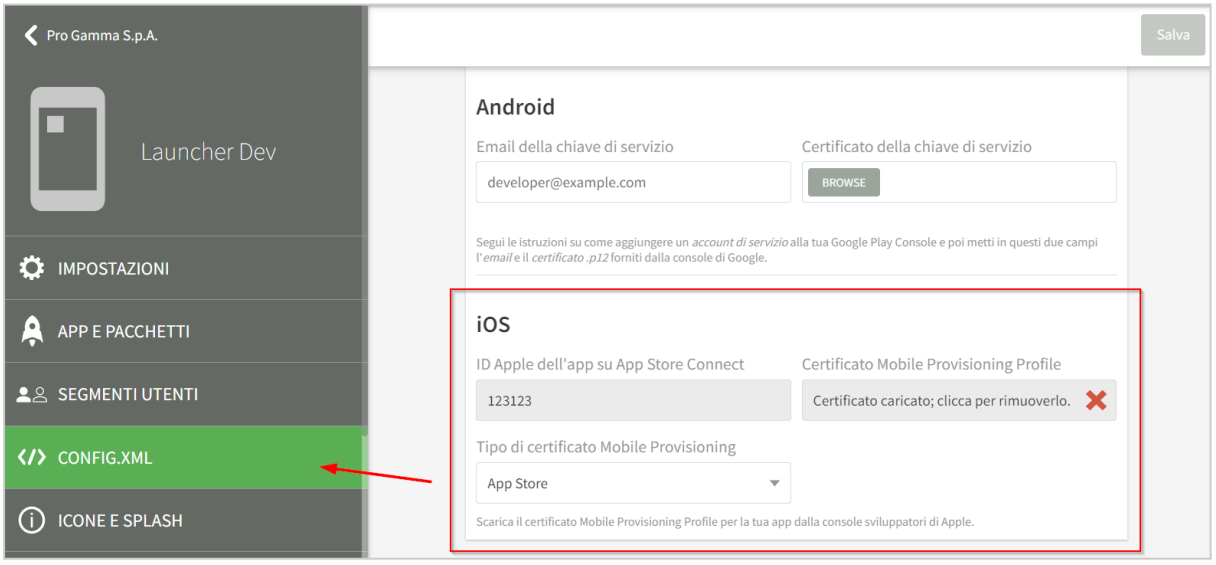
Il distribution provisioning profile contiene informazioni che riguardano il certificato usato per firmare l'applicazione e l'App ID che identifica l'applicazione stessa. Viene usato dall'App Store per assicurarsi che l'applicazione è stata inviata allo store dal legittimo proprietario e per determinare quali autorizzazioni ha l'applicazione (notifiche push, camera, ecc.).

Per ottenerlo occorre fare login nella console developer di Apple e cliccare poi su *Certificati, identificatori e profili*. Scegliere poi *Profili* dal menu a sinistra e cliccare sul pulsante "+" per aggiungere un nuovo profilo. Selezionare l'opzione *App Store* e cliccare il pulsante *Continua*.



Nella videata che segue, scegliere dalla combo box l'App ID associato all'applicazione che si sta pubblicando e cliccare il pulsante *Continua*. Scegliere poi il certificato creato in precedenza, che verrà così incluso nel provisioning profile e cliccare di nuovo *Continua*. Scrivere un nome per il provisioning profile nel relativo campo e cliccare ancora *Continua*. A questo punto è possibile scaricare il provisioning profile cliccando il pulsante *Download*.

Il provisioning profile deve essere impostato nella sezione iOS della videata che si apre dalla voce di menu *Config.xml* nella videata del launcher.



Dopo aver completato la configurazione del launcher, sarà possibile inviare l'app all'App Store cliccando il pulsante *Compila iOS*, presente nella parte alta della videata che si apre dalla voce di menu *Impostazioni*, e poi cliccando *Compila e invia*.

Preparazione per Android

Configurazione Play Console

Per prima cosa, è necessario creare un account sviluppatore sulla Play Console o fare login usando un account esistente. Occorre poi aggiungere una nuova app nella Play Console e configurare la sua scheda.

Creazione del keystore

Il keystore è un repository di certificati di sicurezza e chiavi private usate per firmare l'applicazione. È possibile creare il keystore e la chiave per firmare l'applicazione da Android Studio seguendo questa guida, oppure è possibile crearli da riga di comando con la seguente istruzione:

```
keytool -genkey -v -keystore <name>.jks -keyalg RSA -keysize 2048 -validity 10000 -alias <alias-name>
```

Il comando *keytool* si trova nella cartella *bin* della directory della jdk. Se la jdk non è ancora stata installata, prima di proseguire occorre installarla.

Il campo *<name>* deve essere sostituito con un nome descrittivo che identifica il keystore. Il campo *<alias-name>* deve essere sostituito con un nome descrittivo che identifica la chiave che si sta aggiungendo al keystore.

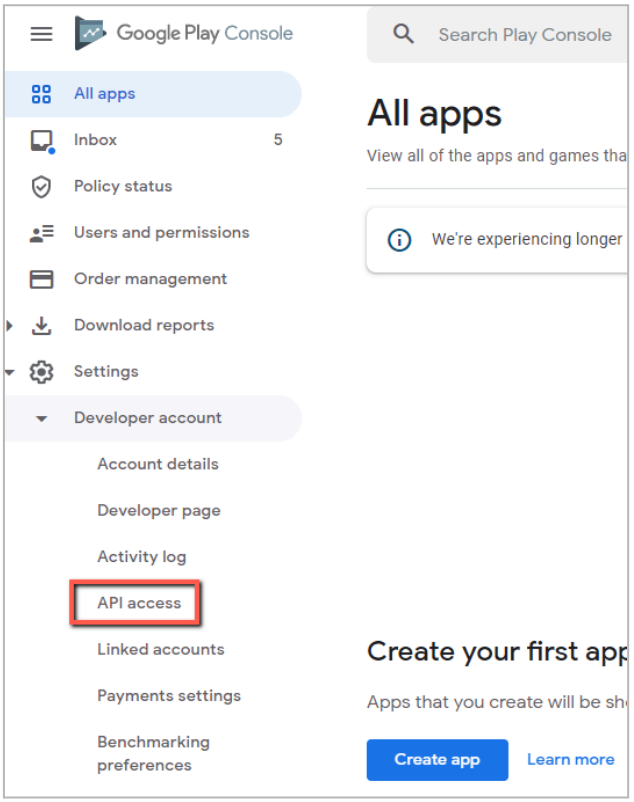
Dopo aver eseguito il comando, verranno richieste una password per il keystore e una password per l'alias del keystore e verrà generato un file *<name>.jks* nella stessa directory in cui è stato eseguito il comando. È importante conservare in un luogo sicuro e non perdere mai né il keystore generato né le relative password.

Creazione account di servizio

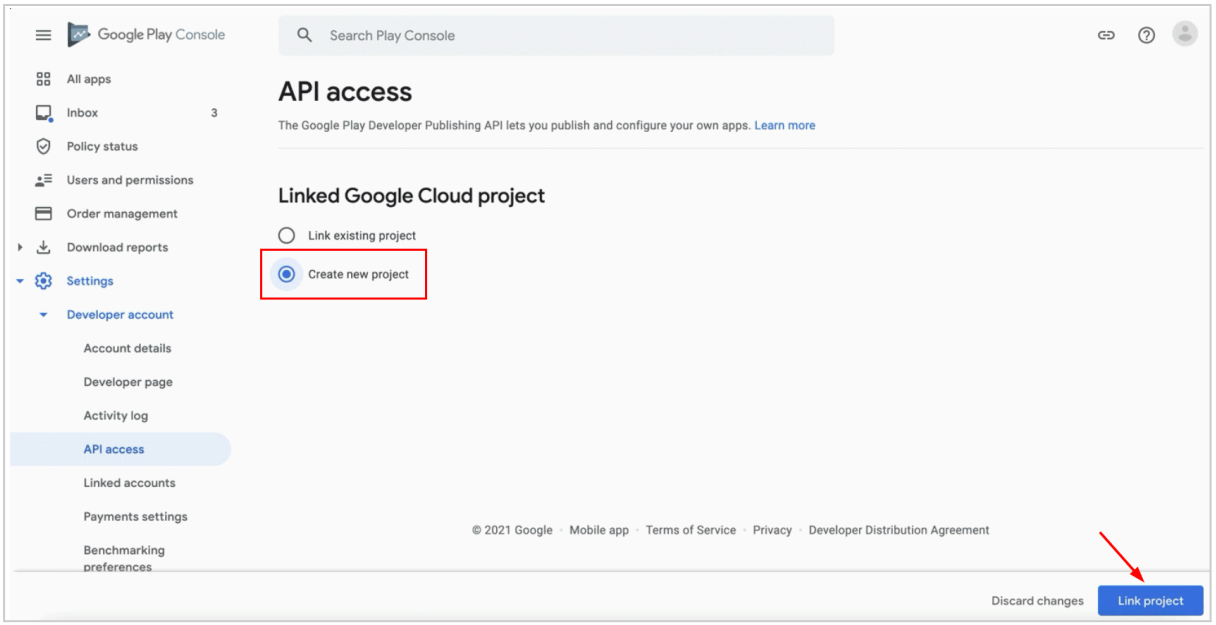
L'account di servizio permette alla console di Instant Developer Cloud di pubblicare l'applicazione sullo store.

Se si possiede già un account di servizio, è sufficiente aggiungere l'applicazione all'account già esistente. Si accede all'account di servizio dal menu *Utenti e autorizzazioni* della Play Console.

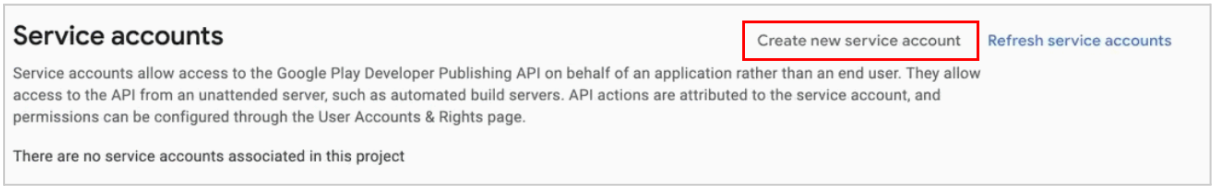
Se invece occorre creare un nuovo account di servizio, per prima cosa è necessario collegare l'applicazione ad un progetto creato nella Google Developer Console. Dalla Play Console occorre cliccare sul menu *Settings* e poi su *API access*.



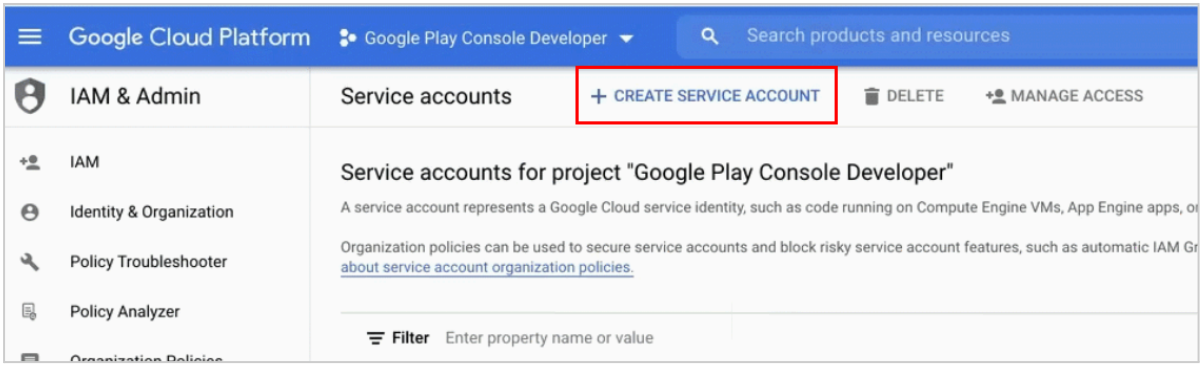
cliccare poi il pulsante *Create new project* e successivamente *Link project*, accettando i termini di servizio quando viene richiesto.



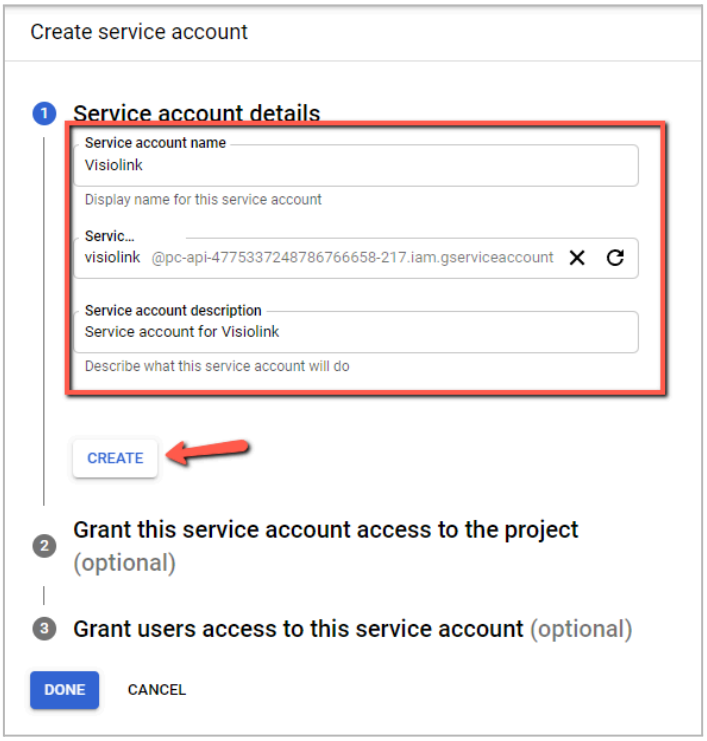
Andare infine in fondo alla pagina e cliccare il pulsante *Create new service account.*



Verrà ora mostrato un popup con le istruzioni sulle prossime operazioni da seguire. Nel popup è presente un link che porta alla *Google Cloud Platform.* Cliccandolo si aprirà una nuova tab in cui occorre cliccare su *+ Create service account.*



Nella videata che si aprirà occorre scrivere un nome per il service account. Il campo *Service account description* è opzionale mentre nel campo *Service account ID* occorre lasciare il valore preimpostato. Cliccare poi il pulsante *CREATE*.



Al passo successivo occorre aggiungere il ruolo *Owner* e cliccare *Done* e finalmente la creazione del service account è completa.

Affinché l'account di servizio sia correttamente configurato, occorre verificare che le Google Play Android Developer API siano abilitate per il progetto appena creato. Per verificarlo occorre:

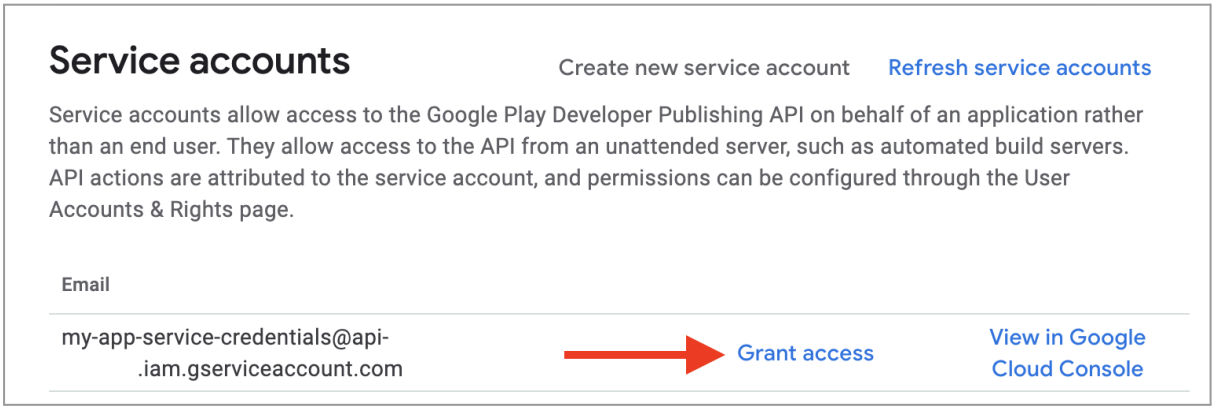
1. scegliere *API Manager* dal menu di sinistra della Google Cloud Console.
2. cliccare *Enable API.*
3. cercare *Google Play Android Developer API.*
4. cliccare sulle API e poi su *Enable.*

Occorre ora scaricare il certificato associato all'account di servizio in formato *p12*. Per ottenerlo seguire questi passaggi:

1. Aprire la pagina che mostra la lista degli account di servizio, cliccando la voce di menu *account di servizio* nel menu di sinistra della Google Cloud Console.
2. Aprire il menu dell'account di servizio cliccando l'icona con i tre puntini a destra
3. Scegliere *Gestisci chiavi.*
4. Cliccare il pulsante *Aggiungi chiave.*
5. Scegliere *Crea nuova chiave* e, nella videata successiva, scegliere il formato *p12*.

A questo punto verrà scaricato il certificato dell'account di servizio in formato *p12*. Prendere nota dell'email dell'account di servizio che viene generata automaticamente perché andrà impostata nel launcher insieme al certificato appena scaricato.

Ora occorre tornare sulla Play Console e cliccare su *Concedi l'accesso* sulla riga relativa al service account appena creato.



Creazione credenziali

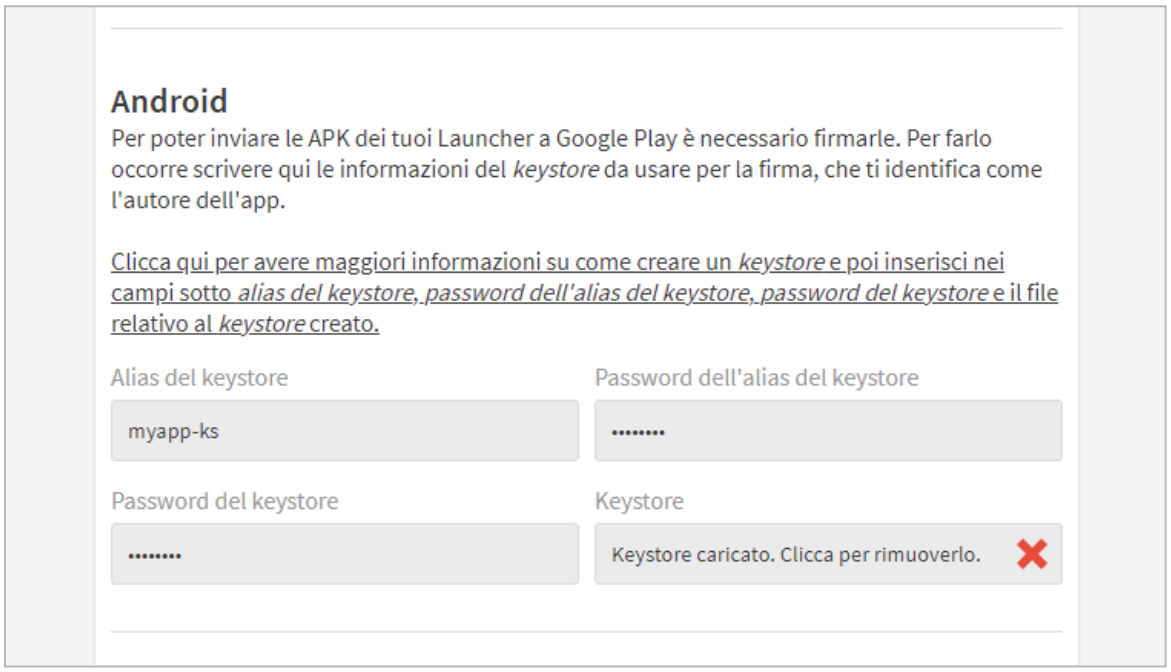
A questo punto si hanno tutti i dati necessari per poter impostare le credenziali per la pubblicazione nella console di Instant Developer Cloud e per preparare il launcher alla pubblicazione.

Se non sono ancora state create delle credenziali per gli store o se si vogliono utilizzare delle credenziali diverse da quelle che si hanno già, occorre aggiungerne delle nuove.

Per creare delle nuove credenziali occorre:

1. cliccare sull'icona dell'ingranaggio nella toolbar in alto;
2. scegliere *App Store* dal menu di sinistra;
3. scegliere *Aggiungi Credenziali* per creare delle nuove credenziali.

Nella sezione *Android* occorre compilare i campi relativi all'alias del keystore, alla password dell'alias del keystore e alla password del keystore usando le credenziali appena create. Occorre infine caricare il file .jks del keystore nel campo *Keystore*.



Scegliere infine le credenziali appena create nella combo della proprietà *Credenziali app store* presente nel pannello *Impostazioni* del launcher.

Impostazioni del launcher

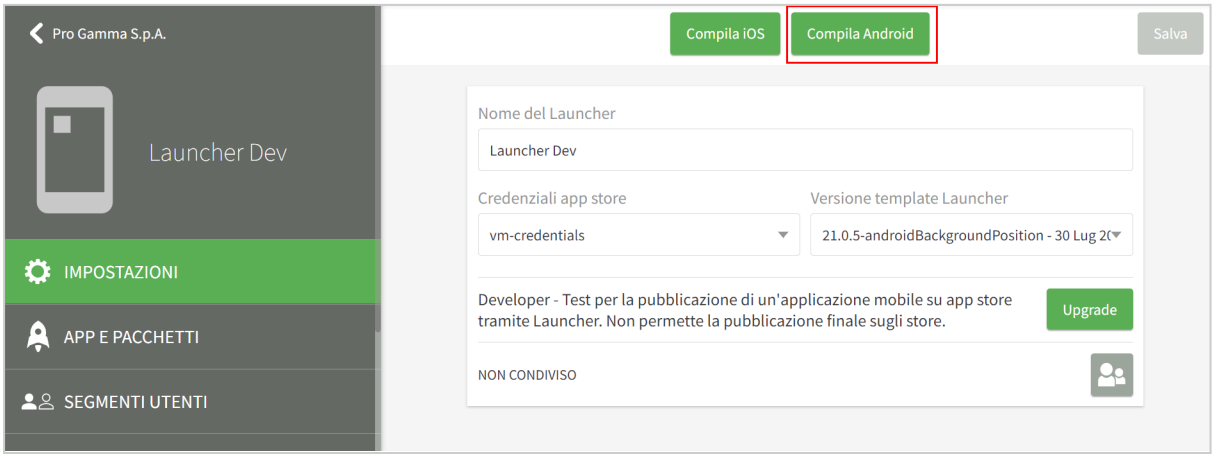
Per quanto riguarda le impostazioni del launcher occorre aprire la console di Instant Developer Cloud, cliccare sul launcher per aprire la videata delle proprietà e scegliere *Config.xml* dal menu di sinistra. Nella sezione *Android* scrivere l'email dell'account di servizio nel rispettivo campo e caricare il certificato.

Ottenere la chiave di caricamento

Per poter pubblicare un'applicazione sul Play Store occorre creare un pacchetto, apk o aab, e firmarlo. Tutte le applicazioni Android vengono firmate dallo store stesso con una chiave privata, ovvero la chiave di firma dell'app che, nel nostro caso, è quella generata in precedenza e conservata nel keystore.

Occorre però anche firmare il pacchetto prima di caricarlo sullo store. La chiave da usare per questa operazione è la chiave di caricamento che sarà resa disponibile nella Play Console dopo un primo caricamento manuale del pacchetto.

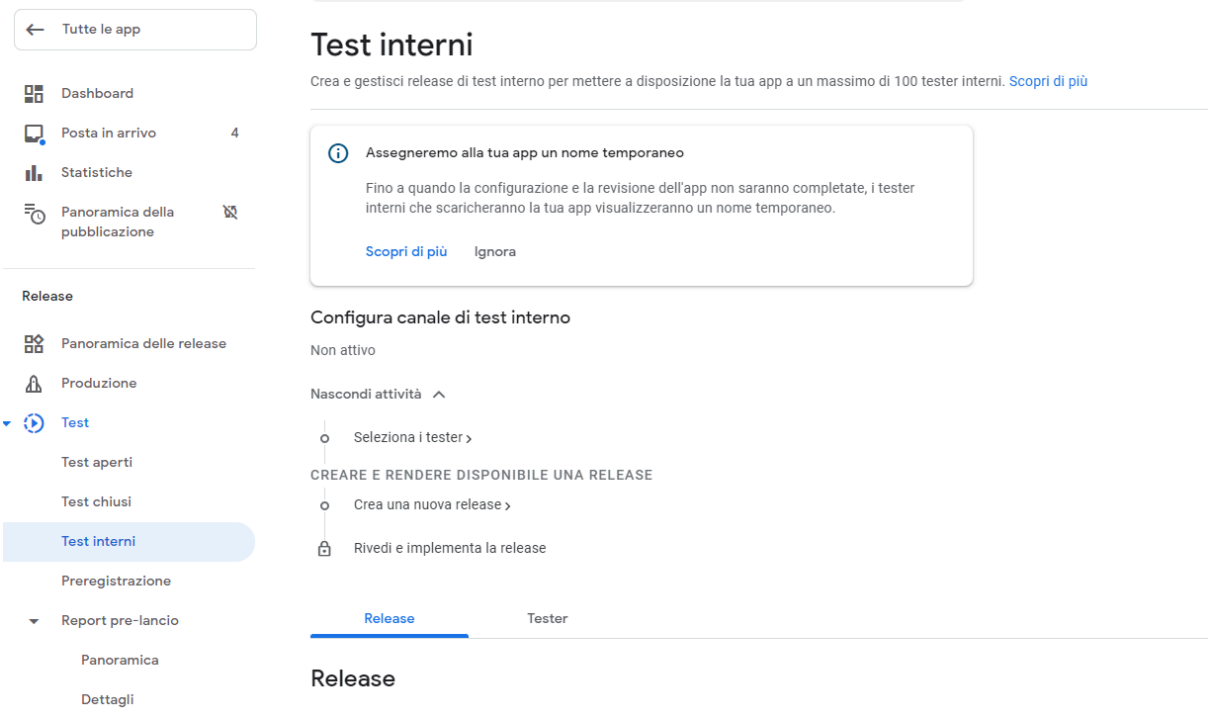
A tal fine, e solo per il primo caricamento, occorre caricare il pacchetto a mano. Per creare il pacchetto, dopo aver completato la configurazione del launcher, occorre cliccare i pulsanti *Compila Android* e, di seguito, *Compila* presenti nelle videate *Impostazioni* o *App e pacchetti* del launcher.



Occorre poi scaricare il pacchetto in formato *aab* dalla sezione *App e pacchetti* del launcher e caricarlo sullo store seguendo questi passaggi:

- Dalla pagina principale della Play Console, selezionare l'applicazione che si sta pubblicando cliccando su di essa.
- Selezionare la voce di menu *Test* > *Test interni* dal menu di sinistra.
- Cliccare il pulsante *Crea nuova release* in alto a destra.

Occorre inoltre completare tutti gli step mostrati nella videata *Test interni* in modo che nello stato dell'app non sia indicato solo *Bozza* ma anche *Test interni*.

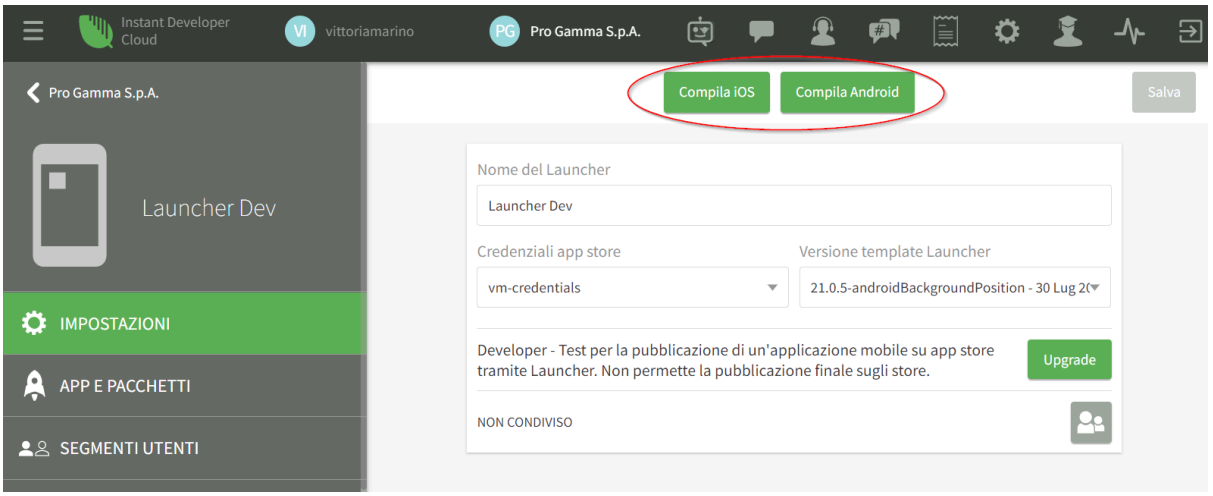


A questo punto sarà possibile caricare il pacchetto nella sezione *App bundle.*

Da questo momento in poi per pubblicare l'applicazione sullo store non sarà più necessario caricare a mano il pacchetto. Sarà possibile pubblicare cliccando il pulsante *Compila Android* nella videata *Impostazioni* o *App e pacchetti* del launcher e poi cliccando il pulsante *Compila e invia*.

Fase di build e di invio

Quando il launcher è configurato è possibile inviare l'applicazione agli store. Per farlo occorre cliccare i pulsanti *Compila iOS* e *Compila Android* e scegliere *Compila e invia*.



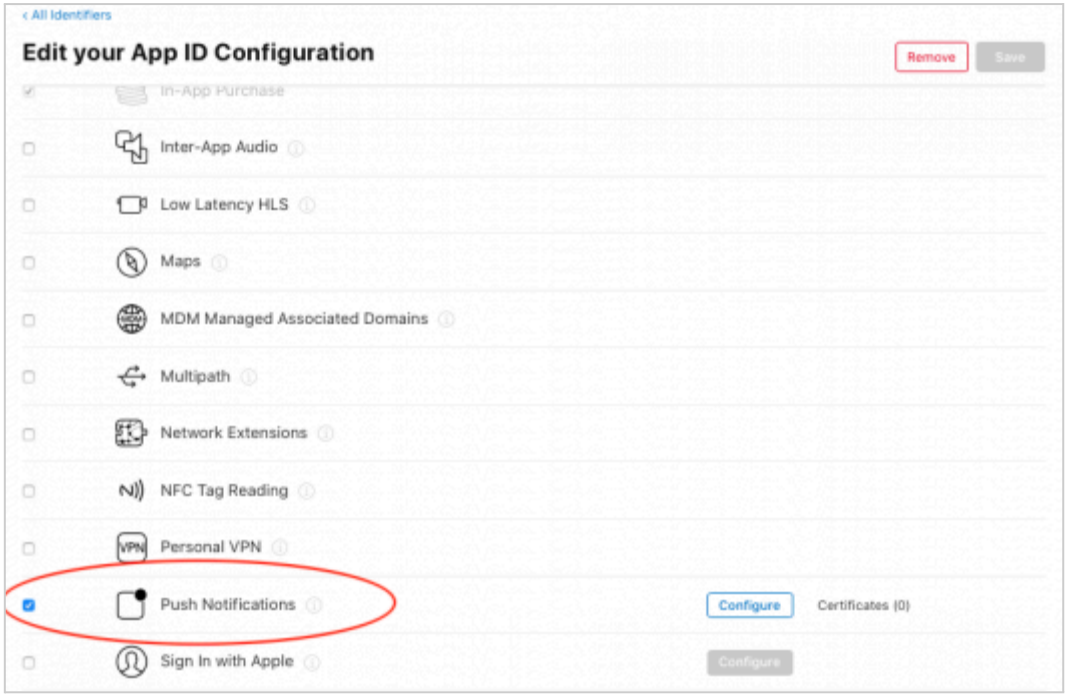
La console di Instant Developer Cloud contatterà a questo punto il proprio build server, ovvero una macchina Mac nel cloud che si occuperà di compilare il pacchetto (apk, ipa, aab).
Se l'operazione va a buon fine l'applicazione verrà inviata allo store. In caso contrario verrà mostrato l'errore che ha interrotto la build o l'invio allo store.

Correzione degli errori iOS

In questo paragrafo vengono elencati una serie di errori comuni in cui è possibile incorrere durante le operazioni di build e invio di un'applicazione iOS. Per ognuna di queste verrà illustrata la strada migliore per affrontarle.

The mobile provisioning profile used is not entitled for Push notifications
Questo errore si può verificare se nell'applicazione vengono usate le notifiche ma la configurazione dell'App ID utilizzato non è corretta. L'App ID in questione è quello che è stato scelto dalla combo durante la creazione del provisioning profile e l'errore indica che l'App ID non ha i permessi necessari per poter usare le notifiche.

Per risolvere questo errore occorre collegarsi alla console developer di Apple, scegliere *Identificatori* e poi l'App ID corretto. A questo punto è possibile selezionare *Push Notification* nella lista delle capability dell'App ID e poi cliccare *Configura* per associare i certificati per le notifiche.



Dopo aver apportato le modifiche e salvato, occorre scaricare nuovamente il provisioning profile e sostituirlo nella sezione *Config.xml* del launcher.

The mobile provisioning profile used does not have a matching bundle ID

Questo errore indica che il bundle impostato nel campo *Bundle* della sezione *Config.xml* del launcher non corrisponde a quello impostato al momento della creazione dell'App ID. L'App ID coinvolto è quello usato per creare il provisioning profile impostato nel launcher, sempre nella sezione *Config.xml*.
Occorre collegarsi alla console developer di Apple e, nella sezione *Identificatori*, verificare il bundle e riportarlo correttamente nelle impostazioni del launcher.

You may be using two-factor authentication for your AppleID. You need to specify an app-specific password as Apple ID password

Questo errore indica che la password indicata nel campo *Password Apple ID* delle credenziali della console di Instant Developer Cloud non è la password specifica per le app. Occorre creare la password specifica per le app e impostarla nelle credenziali per gli App Store della console di Instant Developer Cloud.

Correzione degli errori Android

In questo paragrafo vengono elencati una serie di errori comuni in cui è possibile incorrere durante le operazioni di build e invio di un'applicazione Android. Per ognuno di questi verrà illustrata la strada migliore per risolverli.

APK Version code is lower than the published one

Questo errore indica che la versione corrente dell'applicazione è inferiore alla versione presente sullo store, anche se essa non è ancora stata pubblicata e si trova in uno dei canali di test. Per risolvere questo errore occorre aumentare il numero di versione presente nel campo *Codice versione APK corrente* nella sezione *Config.xml* del launcher.

The apk has permissions that require a privacy policy

Questo errore indica che nell'applicazione è stato incluso un plugin che richiede la presenza di una policy per la privacy, ad esempio camera, localizzazione, registrazione audio, ecc…
In generale la privacy policy è richiesta dallo store in tutti i casi in cui l'applicazione può accedere o raccogliere informazioni personali dell'utente.
Per risolvere l'errore occorre fornire la privacy policy come indicato a questo link.
Al di là dei requisiti dello store, occorre sottolineare che, secondo quanto stabilito dal GDPR, è obbligatorio fornire informazioni sul trattamento dei dati.

No matching client found in google-services.json. You may have entered the wrong bundle (package name)

Questo errore si verifica nel caso in cui siano state attivate le notifiche in un'applicazione per Android, come descritto nel relativo paragrafo, ma il file *google-service.json* usato per impostare il campo *google-services.json* nella sezione *Plugin* del launcher non è stato creato usando lo stesso bundle impostato nella sezione *Config.xml* del launcher.
Occorre collegarsi alla console di Firebase, verificare qual è il bundle usato ed eventualmente modificare il campo *Bundle* nella sezione *Config.xml* del launcher oppure creare un nuovo *google-service.json* con il bundle corretto.

Pubblicazione

Dopo l'invio agli store l'applicazione non è ancora in fase di distribuzione. Per fare in modo che venga distribuita occorre prima promuovere la build tramite le relative funzioni nella console degli store. Per quanto riguarda il Play Store, l'applicazione si trova nella sezione *Test Chiusi* e per renderla pubblica sullo store è possibile seguire questa guida di Google a partire dal passaggio 2.

Per quanto riguarda l'App Store, l'applicazione si trova nella sezione dedicata a *Test Flight*. Per renderla pubblica sullo store è possibile seguire questa guida di Apple.

Debug del launcher dall'ambiente nativo

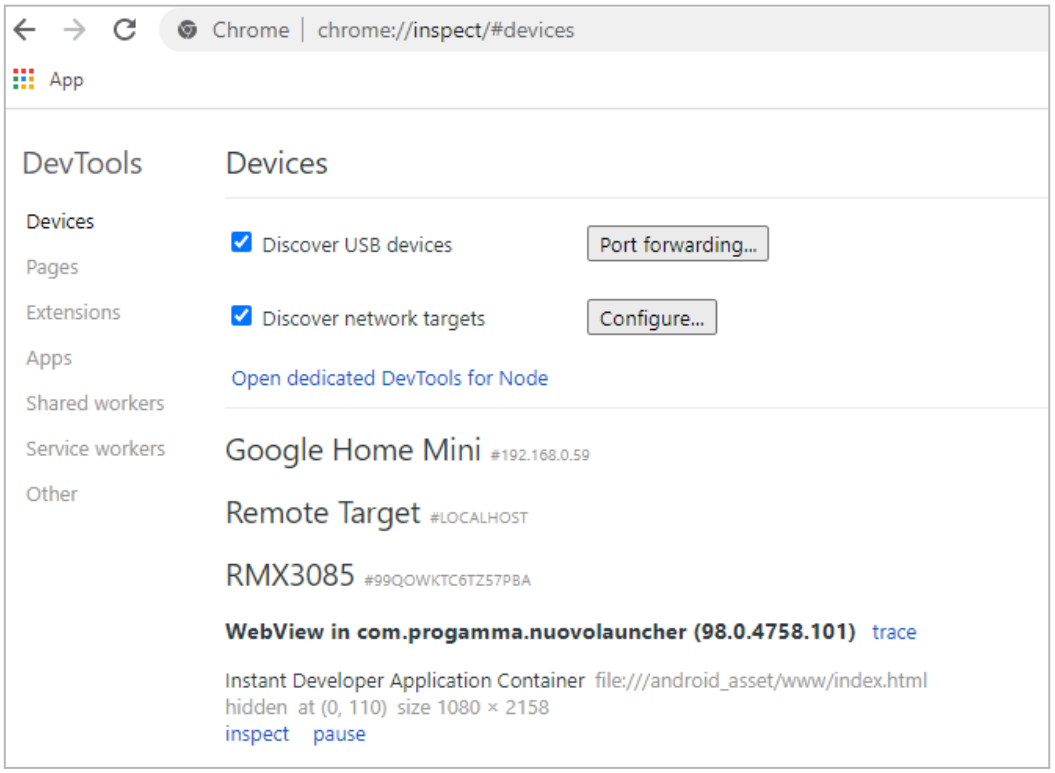
In alcuni casi può rendersi necessario eseguire il debug del launcher nell'ambiente di programmazione nativo (Android Studio o XCode). In tal caso è possibile scaricare il progetto Cordova già pronto per essere aperto nell'ambiente di programmazione desiderato.

A tal fine è sufficiente entrare nella sezione *App e Pacchetti* del launcher, poi cliccare il pulsante *Compila Android (o Compila iOS)* ed infine scegliere *Crea Progetto*. Dopo aver eseguito la compilazione, il link per scaricare il progetto diventa disponibile nella riga della lista corrispondente al pacchetto appena creato.

Nel caso di launcher Android è disponibile anche una seconda modalità di debug, che non richiede la presenza di Android Studio e dell'intero framework Android nel proprio PC. Infatti è possibile compilare un pacchetto per Android attivando l'opzione *Attiva debug*. In questo caso, dopo la compilazione del pacchetto, è possibile scaricare nel proprio PC l'applicazione in formato APK con le informazioni di debug.

A questo punto è possibile installare manualmente l'APK nel proprio dispositivo Android e poi collegare il dispositivo al PC per eseguire il debug direttamente usando il browser Chrome. I passaggi sono i seguenti:

- Abilitare nel dispositivo le *opzioni sviluppatore* e, al loro interno, attivare *debug USB*.
- scrivere il seguente url nella barra degli indirizzi di Chrome: *chrome://inspect.*
- accettare la richiesta di autorizzazione che potrebbe comparire sul dispositivo.
- eseguire l'applicazione sul dispositivo.
- cliccare il link blu *inspect* mostrato nell'immagine seguente*:* si aprirà una nuova videata di Chrome dalla quale sarà possibile effettuare il debug del codice JavaScript dell'applicazione.



Gestione dell'applicazione

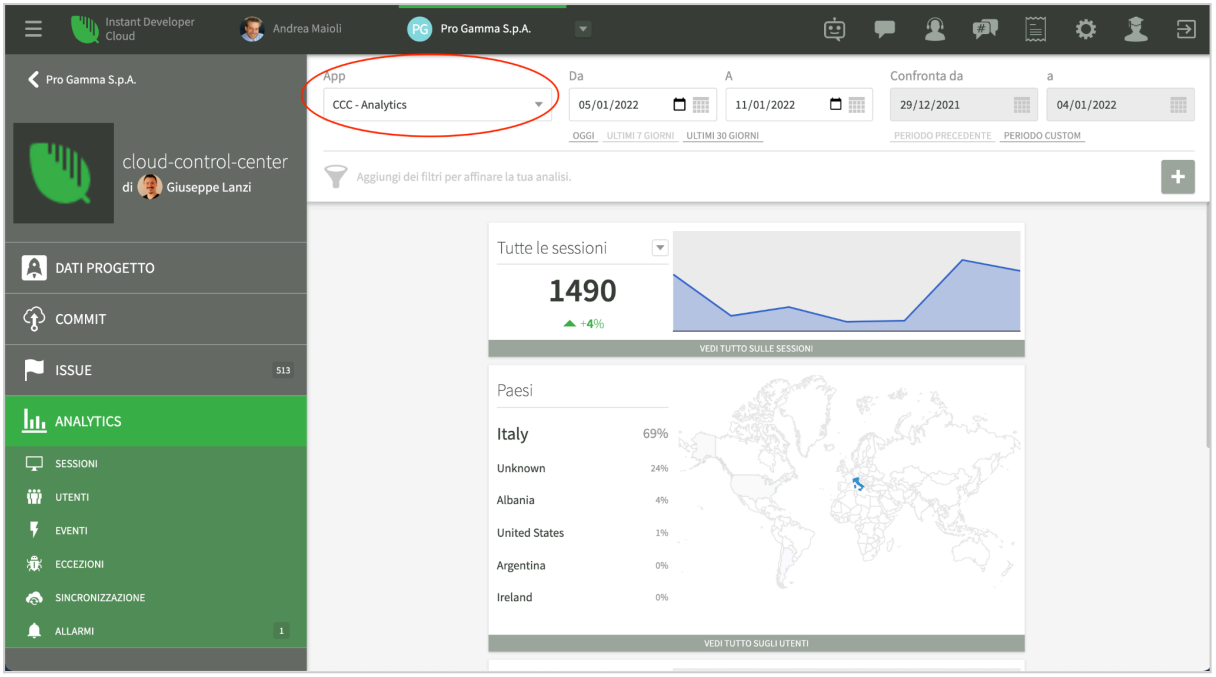
Verifica del corretto funzionamento

Dopo aver pubblicato l'applicazione, è necessario comprendere:

- se gli utenti stanno utilizzando l'applicazione come previsto;
- se si verificano errori all'interno dell'applicazione;
- se i dati vengono sincronizzati correttamente;

- se gli utenti sono contenti di utilizzare l’applicazione.

Per poter raccogliere queste informazioni nella maniera più semplice possibile, si consiglia di attivare il modulo *Analitiche e Feedback*, la cui documentazione è disponibile a questo link..



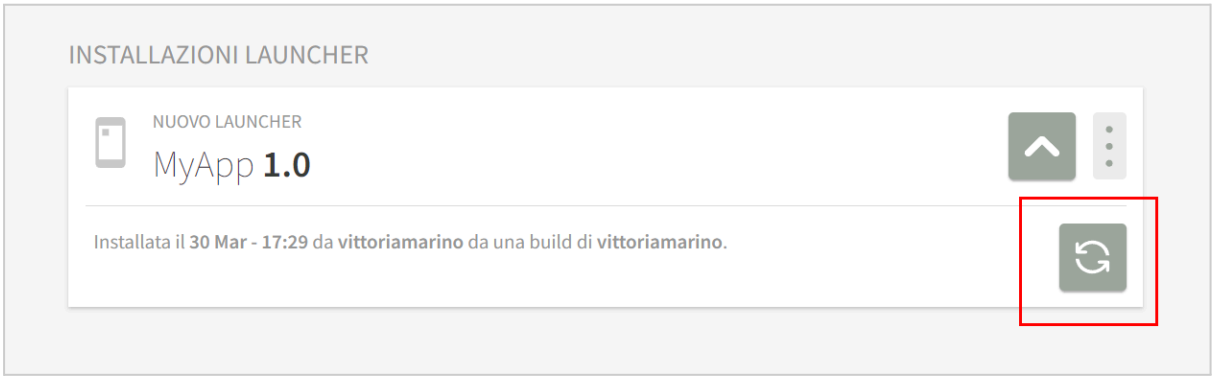
Nota bene: la distribuzione di un'applicazione destinata ad un pubblico generico (b2c) tramite gli store senza aver attivato il servizio di analitiche e feedback è fortemente sconsigliata. Gli utenti di tipo Consumer, infatti, hanno come unica modalità di interazione con gli sviluppatori le recensioni o i commenti sugli store. Anche un semplice bug può causare una serie di recensioni negative che influenzeranno le performance dell’applicazione per lungo tempo.

Se invece l’applicazione è destinata ad un pubblico ristretto (b2b o b2e) è possibile contattare direttamente gli utenti per avere un riscontro del funzionamento. Anche in questo caso, tuttavia, il servizio di analitiche e feedback risulta molto più conveniente in quanto offre un controllo completo.

Si consideri infine che l’utilizzo di un launcher premium completa perfettamente il servizio di analitiche e feedback permettendo la correzione istantanea degli eventuali problemi rilevati dai dispositivi.

Pubblicazione di un aggiornamento

Quando si desidera pubblicare una nuova versione dell'applicazione è innanzitutto necessario compilare ed installare una nuova build nel launcher. Questo può essere fatto tramite le funzione nella sezione *Installazioni* del progetto, come già visto in precedenza.



Se si sta utilizzando un launcher premium, questa operazione è sufficiente per aggiornare le applicazioni nei dispositivi in cui essa è installata. L'aggiornamento avverrà automaticamente al primo avvio dell'applicazione o riattivazione dallo stato di background.

Se invece si sta utilizzando un launcher standard, occorre inviare una nuova versione agli store ed eseguire nuovamente tutte le operazioni di pubblicazione.

Questa operazione diventa necessaria anche per i launcher premium nel caso sia necessario modificare i plugin nativi installati nell'applicazione. In ogni caso, si consiglia di inviare agli store una nuova versione ogni tre mesi per dare agli utenti l'impressione che l'applicazione sia in fase di sviluppo attivo.

Si ricorda la possibilità di utilizzare i Segmenti utenti per effettuare rilasci parziali delle nuove versioni in caso di launcher premium.