Cordova

Rev 2.4 del 05/05/2021

Hybrid APP

Allo stato attuale possiamo classificare le applicazioni per il mondo mobile in base alle tecnologie utilizzate in tre categorie:

- **app native** sono le app scritte e compilate per una specifica piattaforma utilizzando uno dei linguaggi di programmazione supportati dal particolare sistema operativo
- **web app**, sono pagine Web ottimizzate per dispositivi mobili (ad esempio tramite Bootstrap o jQuery Mobile) sfruttando le tecnologie Web, (HTML5, CSS3, JavaScript).
- **app ibride** sono le app che cercano di sfruttare il meglio delle due categorie precedenti: Sono scritte con tecnologie Web ma vengono eseguite localmente all'interno di un'applicazione nativa

Ogni tipologia ha aspetti positivi e negativi e la scelta deve essere fatto valutando attentamente le diverse condizioni ed il contesto in cui si intende operare.

- **un'applicazione nativa** ha dalla sua parte una maggiore velocità di esecuzione e una maggiore integrazione con la piattaforma, ma è **dipendente dalla piattaforma**. Le applicazioni native sono scritte usando linguaggi di programmazione completamente diversi:
  - una app **Android** è scritta con **Java,**
  - una app per **iOS** è scritta con **Swift** (rondine -> leggerezza) oppure **Objective-C** (più vecchio, sul quale si basa Swift),
  - una app per **windows Phone** è scritta mediante il **NET** framework
- Una **Web app** invece risulta **indipendente dalla piattaforma** ma richiede una connessione Internet attiva e non é in grado di accedere al file system o ad altre risorse hardware del dispositivo
- La **app ibrida** coniuga i vantaggi delle web app con quelle delle app native:
  - consentente di sviluppare l'applicazione utilizzando esclusivamente le tecnologie Web
  - le applicazioni vengono eseguite in 'locale' senza necessità di connessione Internet costante
  - hanno accesso alle risorse locali del dispositivo. Le funzioni di interazione con l'hw vengono wrappate all'interno di apposite funzioni java script.
  - possono essere pubblicate sullo store esattamente come se fossero app native
  - hanno il notevole pregio di poter "girare" su diversi SO con lo stesso identico codice.

  Anche in questo caso ci sono però degli svantaggi da considerare:
  - L'interfaccia grafica potrebbe risultare poco omogeneo rispetto a quella nativa della piattaforma
  - una minore efficienza nel rendering grafico;
  - la potenziale lentezza di esecuzione nell'accesso alle risorse locali

Strumenti per la creazione di app ibride

Cordova

Progetto creato da una azienda canadese, Nitobi Software, e presentato nel **2009** come **PhoneGap** ad un evento Apple di presentazione e riconoscimento di nuovi progetti. Il nome Phone Gap stava ad indicare il Gap esistente tra il web ed il telefono, tra le web app e le app native. Successivamente il progetto è stato esteso con la possibilità di creare packages installabili non solo su iOS ma anche su Android, Windows Phone, Blackberry, WebOs.

Ad ottobre 2011 Nitobi Software viene venduta ad Abobe e Phone Gap entra a far parte della **Apache Software Foundation** con il nome di **Apache Cordova**, in onore di Cordova Street in Vancouver dove avevano sede gli uffici della Nitobi.

Architettura e Funzionamento

L'architettura di Apache Cordova è strutturata mediante un "contenitore" che esegue localmente la ns applicazione. In pratica **cordova agisce come un web server che "serve" al browser le risorse memorizzate all'interno della cartella www della app.** Il browser viene aperto a schermo intero ed inizialmente visualizza il file **index.html** con tutte le sue dipendenze (CSS e JavaScript).

Oltre a questo, è possibile aggiungere dei **plugin che consentono di wrappare tutte le funzionalità native del dispositivo**, quali ad esempio fotocamera, accelerometro, bussola, GPS.
I plugin aggiungono un livello di astrazione tale per cui le funzionalità native diventano fruibili tramite semplici funzioni JavaScript. Sono sostanzialmente un insieme di API che, al momento della creazione della app, andranno a wrappare al loro interno le API del SO scelto. Queste API **verso il basso** parlano il linguaggio nativo mentre **verso l'alto** parlano JavaScript. C'è un plug-in che wrappa ogni sensore del telefono: fotocamera, accelerometro, gps, ma anche file system, multitouch, etc

Il file **cordova.js** definisce una interfaccia astratta tra la web application ed eventuali plugin di accesso all'hardware del dispositivo. Definisce ad esempio l'oggetto **navigator** di accesso ai plugin e la gestione dell'evento **deviceReady.** Deve essere manualmente linkato al progetto in tutte le applicazioni in cui fa uso di un qualche plugin.

Per sviluppare l'applicazione e compilarla per Android si può usare indifferentemente un PC Linux, Windows o Mac, questo perché **l'Android SDK** è disponibile su tutte queste piattaforme.
Viceversa una app IOS può essere creata solo in ambiente Mac.

Utilizzo di Cordova

- a linea di comando da terminale
- in modo integrato all'interno di un ambiente di sviluppo (ad es Eclispe o IntelliJ Idea), nel qual caso occorre scaricare da cordova.apache.org/#download una serie di files da integrare nel sistema.

PhoneGap

Rappresenta oggi un'implementazione commerciale di cordova. Utilizza tutte le API di cordova più altre API particolari che consentono di usufruire di servizi aggiuntivi. Cordova è un po' come il motore –webkit per Chrome e Safari, mentre Phone Gap ne è l'implementazione commerciale.

Sencha Touch

Rappresenta la derivazione per device mobili del famoso e solidissimo **ExtJS**, con cui si possono creare delle vere applicazioni web per device mobili, con un'interfaccia che pochissimo ha da invidiare alle applicazioni native e un engine JavaScript che, attraverso la sua modalità MVC e alcune soluzioni semplicemente geniali, rende lo sviluppo veloce e molto efficace. I due "difetti" di questo framework sono: 1) una curva di apprendimento molto ripida: la prima volta che si vede un'applicazione Sencha Touch sembra trattarsi di un linguaggio del tutto nuovo, tutt'altro che JavaScript. Appena si prende la mano si capisce invece quanto il framework sia potente ma il primo approccio può essere scoraggiante.

2) le applicazioni che si creano funzionano solo su browser WebKit, e dunque solo su Safari o Chrome, e dunque solo su iPhone o Android.

XDK

Framework creato da Intel e a sua volta basato su Cordova e quindi molto simile. La differenza principale consiste nel fatto che XDK mette a disposizione un comodo IDE grafico per l'esecuzione dei vari step.

I passi necessari per la creazione e il build di un progetto

Cordova può essere installato come semplice package estensivo rispetto a nodejs.
Occorre poi installare l'**SDK** relativo alla piattaforma mobile desiderate.

```
npm install -g cordova
```

Per vedere la versione corrente: **`cordova -v`**

Se non si utilizza l'opzione –g occorre modificare la variabile d'ambiente path oppure lanciare da terminale il seguente comando: `SET PATH=%PATH%;E:\npm\node_modules\cordova\bin\`

1) Creazione di un nuovo progetto

Comando ***`cordova create`***:

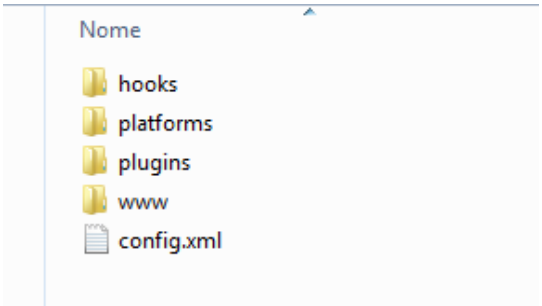
`cordova create <cartella_nuovo_progetto> <nome_package> <nome_app>`

`cordova create ese00_Demo edu.vallauri.demo Demo`

Il **package** serve all'interno per identificare **univocamente** la app. del dispositivo.
App differenti devono assolutamente avere nomi di packages differenti.
Come package si utilizza di solito il **reverse-domain**, cioè il nome di dominio scritto a partire destra.

Il **nome_app** rappresenta il nome identificativo della app, che identificherà la app sullo schermo dello smartphone (insieme all'icona)

Il comando **`create`** crea una nuova **alberatura** a partire dalla cartella avente il nome assegnato.



Il file **`config.xml`** contiene una descrizione della app che stiamo creando (che verrà poi convertita nel Manifest della app). Attenzione che i commenti dentro config.xml sembrano creare problemi.

`<name>Dialog</name>`
Indica il nome vero e proprio della app, che identificherà la app sulla videata home del dispositivo e all'interno degli stores.

`<content src="index.html" />`
Indica la pagina di avvio dell'applicazione.

<plugin name="cordova-plugin-whitelist" version="1" />
Indica un plug-in per la gestione delle restrizioni CORS indispensabile per la realizzazione del progetto. Questo plugin verrà scaricato automaticamente insieme alla platform.

L'impostazione **<access origin="*" />** Indica la cosiddetta **Network Request Whitelist**, cioè l'elenco dei domini esterni ai quali la app potrà inviare richieste Ajax.

- Per default una pagina web non può inviare richieste a domini diversi rispetto a quello da cui ha scaricato la pagina (nel nostro caso dal server locale), a meno che il server di destinazione non specifichi, tramite appositi permessi CORS, i domini abilitati ad inviare richieste (eventualmente qualunque dominio).
- Nel caso delle app, **anche la app deve contenere una white list di domini accessibili**. Ad esempio, al posto di * si potrebbero impostare le seguenti voci:

```
<access origin="http://google.com" />
<access origin="https://google.com" />
<access origin="http://maps.google.com" />
<access origin="http://*.google.com" />
```

Gli <**allow intent**> indicano invece la cosiddetta **Intent Whitelist**, cioè **l'elenco delle URI di cui l'applicazione può richiedere esecuzione al Sistema Opertivo del telefono.** Dalla documentazione ufficiale: *Controls which URLs the app is allowed to ask the system to open.*

Ad esempio inviare una richiesta al SO per l'apertura nel browser di un link esterno.
Gli <**allow intent**> indicano in pratica le restrizioni di accesso ai vari servizi / sensori disponibili sul device (accesso ad internet via http o via https, invio di sms, invio di chiamate, invio di mail, etc.). Queste restrizioni verranno poi riportate all'interno dell'Android Manifest.

E' bene inserire solo ciò che effettivamente serve, in modo da evitare che la app una volta installata, richieda tutti i permessi possibili immaginabili.

```
<allow-intent href="http://*/*" />
<allow-intent href="https://*/*" />
<allow-intent href="tel:*" />
<allow-intent href="sms:*" />
<allow-intent href="mailto:*" />
<allow-intent href="geo:*" />
```

Nota: sembrerebbe che all'interno del Manifest, al momento del build, vengano comunque aggiunti soltanto quei permessi di cui l'applicazione fa effettivamente uso,

Analisi del contenuto della cartella www

La cartella www contiene una applicazione di esempio costituita da un file index.html con relativi css e js. Questa applicazione potrà essere sostituita con i files dell'applicazione da buildare.

I meta tags

Il file **index.html** contiene alcuni meta tag importanti per funzionamento di cordova

```
<meta http-equiv="Content-Security-Policy" content=" default-src *;">
<meta name="format-detection" content="telephone=no">
<meta name="msapplication-tap-highlight" content="no">
<meta name="viewport" content="initial-scale=1, width=device-width,
                                                viewport-fit=cover">
<meta name="color-scheme" content="light dark">
```

Il **primo** meta tag è relativo alle **content-security-policy (CSP),** uno standard definito dal W3C per regolamentare ciò che una pagina HTML può fare. Le policy definite nel progetto Demo sono in realtà abbastanza stringenti e non consentono alcune cose. `default-src *;` lascia passare tutto.

Il **secondo meta tag** disabilita l'individuazione automatica dei numeri di telefono: se attiva questa funzionalità trasformerà automaticamente in link cliccabili tutti i numeri che identificherà come numeri telefonici, link che consentono l'apertura della app di invio delle telefonate.
Allo steso modo potrebbero essere disabilitati gli indirizzi mail e le url:

```
‹meta name="format-detection" content="telephone=no,date=no,address=no,email=no,url=no"/>
```

Il **terzo meta tag** riguarda la grafica del tap (click).
Evita che il testo venga evidenziato quando si esegue un tap

Il **quarto meta tag** riguarda il ViewPort, cioè le modalità di apertura della pagina sul device (livello di zoom, larghezza della finestra, etc.).

Analisi della Content Security Policy

Le content security policy definiscono delle limitazioni riguardo alle **destinazioni** a cui l'applicazione può inviare le **richieste.** Per default, se non vengono inserite viene lasciato passare tutto.

Il link **https://developers.google.com/web/fundamentals/security/csp** spiega molto in dettagli tutte le possibili configurazioni. Le security policy impostate per default da cordova sono :

```
<meta http-equiv="Content-Security-Policy" content="
    default-src 'self' data: gap: https://ssl.gstatic.com 'unsafe-eval';
    style-src 'self' 'unsafe-inline';
    img-src 'self' data: content:;
    media-src *;
">
```

Le security policy sono costituite da un elenco di voci ciascuna terminata da un punto e virgola e scritta nel formato

```
" chiave1 'valore1'; chiave2 'valore2'; chiave3 'valore3'; "
```

default-src indica le policy che devono essere applicate di base ad ogni tipo di risorsa
script-src indica a chi si possono richiedere i file js
style-src indica a chi si possono richiedere i file css
font-src indica a chi si possono richiedere i file relativi ai font
img-src indica a chi si possono richiedere le immagini
media-src indica a chi si possono richiedere i media files in genere.

Le singole policy specifiche, se non vengono impostate, ereditano automaticamente da **default-src.** Se invece vengono impostate "sostituiscono" default nell'ambito indicato per cui occorre ripetere dentro le singole policy le eventuali impostazioni utilizzate all'interno di **default-src**

Analizzando ad esempio la prima riga relativa a **default-src,** c'è scritto che l'applicazione :

- **'self' :** può richiedere risorse (file js, css, img) al server locale
  Già solo questo impedisce all'applicazione di richiedere link esterni ad eventuali CDN.
- **data: :** Può richiedere qualunque risorsa di tipo data, cioè ad es un file codificato in base64
- **gap :** Può richiedere qualunque cordova internal APIs
- **https://ssl.gstatic.com** : può inviare richieste **https** soltanto al dominio indicato che è un dominio a cui Android necessita di poter accedere per gestire le comunicazioni in background. **https://** consente qualunque richiesta https verso qualunque destinazione

'unsafe-eval' consente l'utilizzo della funzione js eval() ritenuta unsafe (converte una stringa in un oggetto) e per default disabilitata

'unsafe-inline' consente l'utilizzo di script js in modo inline all'interno della pagina html (per default disabilitato). Se non si fornisce questo consenso, non sarà possibile :

- inserire del codice all'interno del tag **<script>** ad inizio pagina
- utilizzare chiamate java script all'interno degli eventi.

***** significa che la risorsa può essere richiesta a qualunque URL con qualunque protocollo

Esempio di unsafe-inline

Il seguente codice

```
<script>
   function doAmazingThings() {
      alert('YOU AM AMAZING!');
   }
</script>
<button onclick='doAmazingThings();'>Am I amazing?</button>
```

dovrà essere riscritto nel modo seguente:

```
<!--  html -->
<script src='amazing.js'></script>
<button id='amazing'>Am I amazing?</button>
```

```
// js
function doAmazingThings() {
   alert('YOU AM AMAZING!');
}
document.addEventListener('DOMContentLoaded', function () {
   document.getElementById('amazing').addEventListener('click', doAmazingThings)
});
```

Esempi di Security Policy

Di seguito una vecchia security policy molto permissiva usta in passato e poi quella usata abitualmente:

```
<meta http-equiv="Content-Security-Policy" content="
    default-src * gap: ws: https://ssl.gstatic.com;
    style-src   * 'unsafe-inline' 'self' data: blob:;
    script-src  * 'unsafe-inline' 'unsafe-eval' data: blob:;
    img-src     * data: 'unsafe-inline' 'self';
">
```

```
<meta http-equiv="Content-Security-Policy"  content="
      default-src *;
      script-src * 'unsafe-inline';
      style-src * 'unsafe-inline';
      img-src *  data:  content:;  "
>
```

Contenuto dell'applicazione demo

index.html

```
<body>
    <div class="app">
        <h1>Apache Cordova</h1>
        <div id="deviceready" class="blink">
            <p class="event listening">Connecting to Device</p>
            <p class="event received">Device is Ready</p>
        </div>
    </div>
</body>
```

index.js

E' stato molto ridotto rispetto alle prime versioni.

L'evento **deviceready** è un evento di cordova che si genera al termine del caricamento di cordova e di tutti i plugin utilizzati. Il **false** finale significa che la funzione di callback deve essere eseguita in modo asincrono (default). **true** significa invece esecuzione sincrona in corrispondenza del bind.

Questo evento viene utilizzato ad esempio per gestire i sensori del telefono, che potranno essere utilizzati soltanto DOPO che tutti i pug sono stati agganciati. Se l'applicazione non fa uso di plugin, questo evento può anche non essere gestito.

```
document.addEventListener('deviceready', onDeviceReady, false);

function onDeviceReady() {
    console.log('Running cordova-' + cordova.platformId + '@'
                                   + cordova.version);
    document.getElementById('deviceready').classList.add('ready');
}
```

index.css

```
.event.listening {      // elementi che implementano sia la classe event sia la classe listening
    background-color:#333333;
    display:block;
}

.event.received {       // elementi che implementano sia la classe event sia la classe received
    background-color:#4B946A;
    display:none;
}

#deviceready.ready .event.listening { display: none; }
#deviceready.ready .event.received { display: block; }
```

2) Creazione della android platform

Il comando ***cordova platform add*** (da eseguire dalla root del progetto corrente) specifica la piattaforma di destinazione per la quale si desidera creare l' APK. Modifica di conseguenza il file config.xml ed aggiunge diverse informazioni all'alberatura corrente.
In particolare nella cartella ***platforms*** viene aggiunta una nuova cartella preposta a contenere il progetto nativo per la piattaforma indicata.

Il comando può essere ripetuto più volte specificando platform differenti.
Al momento del build verranno create le PAK relative a tutte le piattaforme aggiunte.

```
cordova platform add android
cordova platform add blackberry
cordova platform remove blackberry
```

Il comando **cordova platform list** visualizza l'elenco delle piattaforme installate e delle piattaforme che cordova è in grado di gestire. Nel caso di Android visualizza anche la versione **minima** richiesta per il build (versione 9 a gennaio 2021).

```
cordova platform list
```

3) Installazione dei plug-in

```
cordova plugin add cordova-plugin-dialogs
```

Quando si scarica un plug-in, questo viene automaticamente scaricato per tutte le platform per cui quel plug-in risulta disponibile. I plug-in vengono scaricati dal repository di **GitHub.**

La cartella **plug-in** funge in pratica da repository per l'applicazione in corso di sviluppo, dal quale attingeranno poi le varie platform. Non tutti i plug-in sono disponibili per tutte le platform.

Nella sottocartella **cordova-plugin-dialogs/src** viene creata una sottocartella per ogni platform per cui il plug-in risulta disponibile.

L'intera cartella **plug-in** può essere copiata da una macchina all'altra senza problemi. (occorre copiare anche il file fetch.json).

Il plug-in cordova-plugin-whitelist

viene scaricato **automaticamente** insieme alla platform. E' quello che si occupa di gestire le chiamate CORS verso server esterni implementando le impostazioni definite all'interno del file config.xml

Altri plag-in comuni:

```
cordova plugin add cordova-plugin-dialogs
cordova plugin add cordova-plugin-vibration
cordova plugin add cordova-plugin-splashscreen
cordova plugin add cordova-plugin-geolocation
cordova plugin add cordova-plugin-device-motion
cordova plugin add cordova-plugin-console
cordova plugin add cordova-plugin-websql (accesso db locale)
```

```
cordova plugin list
cordova plugin remove <PLUGIN_NAME>
```

Nel caso di Android, il file **Notification.java** contiene il codice sorgente java del plug-in.
E' una classe che estende org.apache.cordova.CordovaPlugin
Nel caso di iOS il sorgente è memorizzato in un file Objective-C con estensione .m

Durante il download del plug-in, all'interno della cartella **platforms\android\platform_www\plugins** (per tutte le platform installate) viene creato un file Notification.js (quello esterno alla cartella android) che funge da wrapper per il plag-in vero e proprio.
Quello interno alla cartella android contiene invece le istruzioni per la gestione della pagina.

Se si aggiunge una nuova platform, dopo aver già scaricato i plug-in, al momento della creazione della platform cordova consulta il repository dei plug-in scaricati ed automaticamente crea il wrapper all'interno della platform. Per cui è indifferente scaricare *prima* i plugin e *dopo* creare le platform o viceversa

4) Build

Configurazione dell'ambiente

Per poter eseguire il build, sul PC corrente devono essere preventivamente installate :

- la **JDK** java (java development kit) di cui è espressamente richiesta la **versione 8** (**jdk1.8.***)
- la **SDK** android (software development kit).
- **gradle** che è il compattatore che fisicamente andrà a creare l'APK (la quale è un pacchetto compresso "simile" agli zip).
- Un **emulatore** per poter eseguire rapidamente il test della app (es **Pixel 2**)

Tutti questi strumenti vengono automaticamente installati con **Android Studio**, per cui si consiglia la sua installazione preventiva. A gennaio 2021 cordova utilizza per il build **Android 9**, che deve essere ulteriormente aggiunto all'SDK dopo aver installato Android Studio.

Occorre quindi definire delle variabili d'ambiente che indichino a cordova la posizione delle installazioni precedenti. A tal fine si può utilizzare il seguente file **setEnv.bat** da eseguire ogni volta che si apre un nuovo terminale:

```
SET JAVA_HOME= C:\Program Files\Java_8_261\jdk1.8.0_261
REM SET ANDROID_HOME=F:\Sdk  // deprecate e sostituita dalla successiva
SET ANDROID_SDK_ROOT= C:\Users\myUser\AppData\Local\Android\Sdk

SET PATH=%PATH%; C:\Program Files\Java_8_261\jdk1.8.0_261\bin;
SET PATH=%PATH%; C:\Users\myUser\AppData\Local\Android\Sdk\tools;
SET PATH=%PATH%; C:\Users\myUser\AppData\Local\Android\Sdk\platform-tools;
// path di gradle
SET PATH=%PATH%; C:\Users\myUser\.gradle\wrapper\dists\gradle-6.5-
bin\6nifqtx7604sqp1q6g8wikw7p\gradle-6.5\bin
```

Uscita tramite proxy

Oltre ai precedenti comandi di ambiente, se il traffico di uscita verso Internet è controllato da un **proxy** server occorre settare le seguenti variabili di ambiente che consentano al builder di inviare richieste http e https tramite proxy:

```
set HTTP_PROXY=http://10.0.5.1:3128
set HTTPS_PROXY=http://10.0.5.1:3128
```

Configurazione di npm per l'uscita tramite proxy

Anche npm deve essere configurato in modo da poter "uscire" tramite proxy:

```
call npm config set proxy       http://10.0.5.1:3128
call npm config set https-proxy http://10.0.5.1:3128
```

- La variabile **proxy** indica ad NPM che le chiamate http devono essere inviate non al default gateway ma alla macchina avente l'indirizzo indicato sulla porta 3128.
- La variabile **https-proxy** indica ad NPM che le chiamate https devono essere inviate alla macchina avente l'indirizzo indicato sulla porta 3128
- Il valore da assegnare alle due variabili è una **URL** che **in entrambi i casi deve iniziare con http://** che può essere scritta con o senza virgolette.
- senza CALL il comando viene eseguito, ma NON termina correttamente
- Le impostazioni di npm vengono salvate in un file di configurazione e quindi non è necessario rieseguirle una seconda volta

Configurazione di gradle per l'uscita tramite proxy

Occorre infine istruire gradle affinchè anche lui possa uscire tramite proxy:
A tal fine occorre aggiungere all'interno del file **platforms / android / gradle.properties** le seguenti configurazioni che istruiscono gradle di inoltrare le chiamate verso il proxy:

```
systemProp.http.proxyHost=10.0.5.1
systemProp.http.proxyPort=3128
systemProp.http.nonProxyHosts=localhost

systemProp.https.proxyHost=10.0.5.1
systemProp.https.proxyPort=3128
systemProp.https.nonProxyHosts=localhost
```

Build

```
cordova build android             // versione di debug
cordova build android  --release  // versione di relese
```

Se si omette android viene eseguito il build per tutte le piattaforme installate. Specificando android viene eseguito soltanto in build per android. In corrispondenza del build parte il download di tutte le API necessarie alla realizzazione del progetto. Il download viene fatto dal repository di **maven.org**.

Le opzioni `--stacktrace e --verbose` aggiungono delle informazioni utili in caso di errori.

Attenzione che il BUILD sembrerebbe andare in errore se è attivo l'hotspot mobile di windows

La cartella platforms / android

Dopo il build contiene la tipica alberatura di una applicazione nativa android.
In fase di build l'intero suo contenuto viene compresso all'interno dell'APK finale.
Nelle versione di debug vengono inclusi anche i sorgenti js, nella versione di release ovviamente no.

- **app / src** contiene il sorgente java compreso il file **`manifest`** All'interno di MainActivity.java si può vedere come all'avvio venga inviata in esecuzione nel browser locale la pagina index.html dichiarata all'interno della variabile **`launchUrl`** definita all'inizio all'interno del file config.xml
- **app / src / main / assets / www** contiene tutte le pagine HTML CSS JS che costituiscono l'applicazione. Tra di esse si può vedere l'aggiunta della libreria cordova.js creata dinamicamente al momento del build
- **app / src / main / res / values** contiene il file **`strings.xml`** con il nome dell'applicazione e alcune variabili globali
- **app / build / outputs / apk** contiene il **`file.apk`** che può essere installato sullo smartphone. Il nome del file non ha nessuna importanza. Ciò che conta sono il package-name e la app-name memorizzate all'interno dell'apk medesima.

La cartella hooks

E' una cartella preposta a contenere i cosiddetti 'uncini' cioè script che possono essere eseguiti prima o dopo il build. Gli uncini sono suddivisi in due gruppi: **`before_build`** e **`after_build`**

5) Deploy e test dell'applicazione

Per lanciare l'applicazione utilizzare uno dei seguenti comandi:

```
cordova run android
cordova emulate android
```

La differenza tra i due comandi consiste nel fatto che:

- **`cordova run`** verifica se c'è un telefono connesso ed in tal caso lancia l'esecuzione direttamente sul telefono (*ma non sempre funziona*), altrimenti usa l'emulatore (se è aperto)
- **`cordova emulate`** utilizza sempre solo l'emulatore (se è aperto)

Riguardo all'emulatore, in entrambi i casi occorre avviarlo preventivamente tramite il Device Manager di Android Studio. Con l'emulatore aperto l'applicazione viene deployata ed eseguita.
Se si utilizza l'emulatore (es **Pixel 2**) sembrerebbe necessario un emulatore almeno **API level 28**

Riguardo all'indirizzamento IP, le richieste del client devono essere indirizzate all'indirizzo IP di ascolto del server, che **non può essere 127.0.0.1**, in quanto 127.0.0.1 rappresenta il loopback dell'emulatore o dello smartphone, ma **deve essere** l'ip della macchina, oppure l'ip dell'antennina (purchè il servizio hotspot mobile sia abilitato), cioè 192.168.137.1

Build di una versione di release

Per buildare la versione di release occorre utilizzare delle chiavi di cifratura:

https://stackoverflow.com/questions/55098935/cordova-build-android-debug-mode-vs-release-mode-when-uploading-apk

https://codeburst.io/publish-a-cordova-generated-android-app-to-the-google-play-store-c7ae51cccdd5

6) Debug

Tramite google chrome è possibile debuggare in tempo reale una app buildata in esecuzione sullo smartphone oppure sull'emulatore.

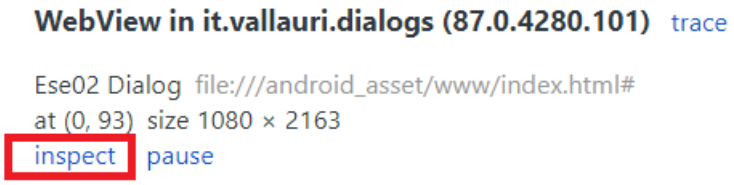
Per quanto riguarda il telefono occorre abilitare la modalità "sviluppatore" e debug USB :

- eseguire 7 tap su impostazioni / Informazioni Telefono / Informazioni software / Build number
- In questo modo al fondo delle impostazioni vengono aggiunte le "Opzioni sviluppatore"
- Nelle "Opzioni sviluppatore" abilitare **debug USB**

Ciò fatto, inserire il seguente comando nella barra di navigazione di chorme:

```
chrome://inspect
```

automaticamente visualizza l'eventuale dispositivo collegato (e/o l'emulatore) e, all'interno del dispositivo, l'elenco delle app attualmente in esecuzione.
Selezionare la app desiderata e cliccare su **inspect**,



All'interno di chrome viene riportata una visualizzazione Read Only della schermata del telefono.
A questo punto è possibile **interagire con l'applicazione direttamente sul telefono** ed eseguire il debugging javascript su chrome.

La cartella che viene utilizzata per il debugging è la cartella **assets/www** contenente i file javascript nella loro versione originale. Infatti l'applicazione finale, prima di essere buildata, viene copiata all'interno della cartella **/platforms/android/app/src/main/assets/www** dove, in fase di build, viene aggiunto anche il file cordova.js. In questa cartella è sufficiente aprire index.html e procedere al debug.

Test tramite il protocollo http

Una simulazione in ambiente http è più realistica in quanto l'applicazione verrà poi eseguita sullo smartphone all'interno di un web server. Il comando

```
cordova serve android [port]
```

attiva un Web server sulla porta predefinita **8000** (il numero di porta può essere configurato passandolo come ultimo parametro).

A questo punto dal browser si può accedere tramite **http://localhost:8000** e poi cliccare su **Android**, oppure lanciare direttamente **http://localhost:8000/android/www**

Note sul Deploy

Dopo aver collegato lo smartphone al PC, con windows 10 è sufficiente trascinare l'apk sullo smartphone ed eseguire l'installazione.

Se si vuole invece gestire una comunicazione tramite un browser (con lo scopo di debuggare la app), occorre installare esplicitamente i cosiddetti ADB driver legati ad ogni singolo modello di smartphone e scaricabili gratuitamente dal sito del costruttore. Dopo aver scaricato i driver i passi da eseguire per scaricare la app sono i seguenti:

```
adb devices –l      (c:\android-sdk\plaform-tools\adb devices –l)
```

Mostra la list degli attached devices.
`adb` è il bridge (porta 5037) tra android e i dispositivi fisici.

```
cordova run android –nobuild –target="id del dispositivo mostrato da adb devices - l"
cordova run android –nobuild –target=QLF7N15912017009
oppure
cordova run android –nobuild –target QLF7N15912017009
```

Se c'è un solo dispositivo connesso il target può essere omesso.

Se si fa il run senza fare -nobuild, viene automaticamente fatto anche il build. Oneroso.

Esempi di applicazioni

Build di applicazioni client/server HTML5/CSS3/JS

Partendo da una applicazione client server esistente, i passi da compiere per incapsularla all'interno di una app sono i seguenti:

client

1. Aggiungere a tutte le pagine **html** i 5 meta tag tipici delle applicazioni cordova, sostituendo la `<meta http-equiv="Content-Security-Policy"` con quella indicata a pag 7
2. Sul client modificare la funzione **inviaRichiestaAjax()** impostando la URL assoluta di ascolto del server:
   ```
   url: "https://192.168.137.1:1338" + url          // oppure
   url: "https://my-crud-server.herokuapp.com" + url
   ```
   Per quanto riguarda **window.location.href** non ci sono problemi in quanto tutte le pagine **html** saranno residenti all'interno della app e fornite direttamente dal web server interno
3. In caso di utilizzo dei **websocket** anche sulla **connect** del client occorre impostare la URL assoluta di ascolto del server:
   ```
   var socket = io.connect("https://192.168.137.1:1338");
   ```

server

1. Il server deve assolutamente essere **https**. La ultime versioni di android per default non consentono più alle APP di inviare richieste http
2. Mettere il server in ascolto sull'indirizzo **IP** della macchina oppure omettere l'indirizzo di ascolto, in modo che sia raggiungibile su tutte le interfacce di rete
   ```
   httpsServer.listen(1338, "192.168.137.1", function() {
   ```
3. Aprire le restrizioni CORS in modo che il server possa accettare richieste da qualunque sorgente
4. Il listener statico ed eventuali listener di pagina ad applicazione buildata non servono più, in quanto le pagine saranno fornite dal web server interno di cordova, mentre al server esterno verranno inviate soltanto le richieste Ajax. La parte statica può comunque rimanere se si desidera che il server rimanga fruibile anche via web
5. Sull'eventuale proxy della connessione Internet del PC che funge da server, impostare il bypass del proxy per gli indirizzi locali 192.168.* oppure 10.*.*.*

Le `alert()` producono un anonimo toast a sfondo nero.
Per migliorare l'aspetto grafico si può utilizzare l'oggetto **navigator.notification** (plugin Dialogs)

Aggiunta di una Icona personalizzata

All'interno della cartella **www/img** copiare il file desiderato che deve essere necessariamente un **file.png** con dimensioni non superiori ai 200px.
Aggiungere all'interno di config.xml (dopo content) la seguente riga:

```
<icon src="www/img/miaIcona.png" />
```

I nomi delle risorse devono iniziare con lettera minuscola.
Questa immagine, al momento del build, verrà copiata nella cartella platform / android e verrà utilizzata come **icona identificativa** dell'applicazione all'interno dello smartphone.

Utilizzo dei plugin

Quando all'interno di una applicazione si fa uso di plugin aggiuntivi, all'interno dei vari file .html, prima del collegamento ai files.js, occorre aggiungere **manualmente** il seguente link:

```
<script src="cordova.js">  </script>
```

Nel caso non si usino plug-in questo link non è necessario.
La libreria cordova.js viene aggiunta automaticamente in fase di build all'interno della cartella **`assets`**, cioè la cartella che conterrà i sorgenti dell'applicazione finale.

Questa libreria rende anche disponibile l'evento **deviceready**, generato dopo l'"aggancio" dei sensori. Attenzione che questo evento non è disponibile in jQuery e pertanto deve necessariamente essere gestito in javascript. Può essere usato in abbinamento o in sostituzione dell'evento document.ready:

```
$(document).ready(function() {
   document.addEventListener('deviceready', function() {
```

Aggiunta di uno SplashScreen

```
cordova plugin add cordova-plugin-splashscreen
```

Lo splash screen è una immagine che può essere visualizzata all'avvio della app in attesa del caricamento della app stessa e di tutti i suoi plug-in. Questa immagine può essere visualizzata soltanto al primo lancio (**default**) oppure anche nei richiami successivi (quando in realtà però la app è già in memoria quindi abbastanza inutile). Per poter utilizzare lo splash screen occorre:

- Inserire nella cartella img l'immagine da utilizzare come splashscreen
- Aggiungere all'interno di config.xml le seguenti righe:

```
<splash src="www/img/splash.png"/>
<preference name="SplashShowOnlyFirstTime" value="true"/>  //default
<preference name="SplashScreenDelay" value="3000"/>
<preference name="AutoHideSplashScreen" value="true"/>
<preference name="FadeSplashScreen" value="true"/>
<preference name="FadeSplashScreenDuration" value="750"/>
```

`SplashScreenDelay` è la durata di visualizzazione dello splash (3 secondi) che al termine verrà automaticamente nascosto. Per disabilitare lo splash screen si può utilizzare la seguente impostazione:

```
<preference name="SplashScreenDelay" value="0"/>
```

Le **ultime due** impostazioni consentono di applicare un effetto di fadeout dello splashscreen in corrispondenza della chiusura dello stesso

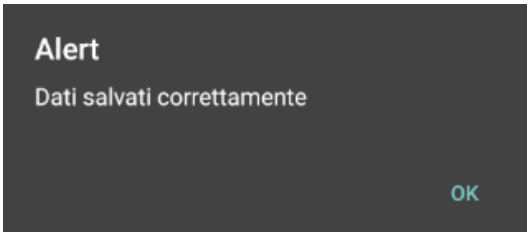
Per quanto riguarda l'immagine, se non si imposta nulla, viene utilizzata l'immagine di default screen.png prememorizzata nella cartella `platforms/android/res`

Immagini multiple
In realtà bisognerebbe fare una cosa più fine aggiungendo una icona per ogni diversa risoluzione, scrivendo 4 righe per i formati portrait e 4 righe per i formati landscape.
Ci sono anche dei siti che, ricevuta una immagine png, generano le immagini corrispondenti nelle varie risoluzioni (splash screen generator)

Finestre di dialogo

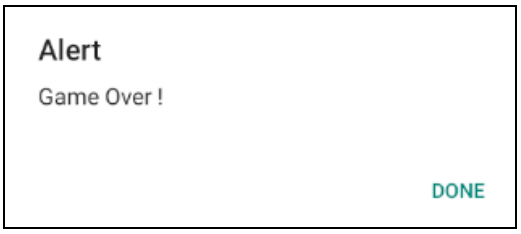
```
cordova plugin add cordova-plugin-dialogs
cordova plugin add cordova-plugin-vibration
```

La classica alert() javascript, buildata nella app senza lacun plugin, produce il seguente risultato



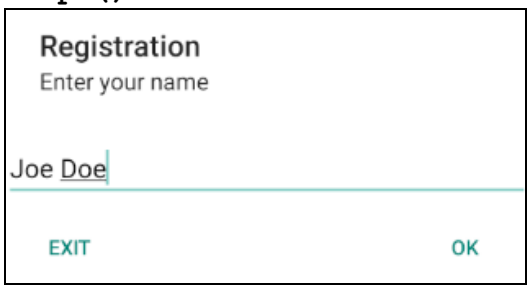
Il plugin **dialogs** rende disponibile all'interno del file js l'oggetto **navigator** che dispone di alcuni metodi per la visualizzazione di alcune finestre di dialogo un pò più ricercate rispetto alla classica alert(). Questi metodi sono:

navigator.notification.alert()



navigator.notification.confirm()

navigator.notification.prompt()



navigator.notification.beep(2); // Numero di beep

navigator.vibrate([500, 500, 500]); // beep, pausa, beep, ...

Codice di accesso ad un sensore

Cordova fornisce un livello di astrazione tale per cui Il codice di accesso ai sensori è praticamente sempre lo stesso indipendentemente dal sensore a cui si desidera accedere ed è basato sui seguenti tre metodi:

.getCurrentPosition(onSuccess, onError, options) restituisce il valore corrente.
In caso di successo viene richiamata la funzione indicata come primo parametro, in caso di errore viene richiamata la funzione indicata come secondo parametro.

.watchPosition(onSuccess, onError, options) Lancia la lettura su un thread separato e legge con continuità fino a quando il thread non viene terminato. La frequenza delle letture dipende dal sensore e non è configurabile dall'utente. Se si desidera una lettura a cadenze regolari l'alternativa migliore è quella di utilizzare getCurrentPosition() all'interno di un setTimeout(). watchPosition restituisce un **watchID** identificativo del thread di lettura .

.clearWatch(watchId) consente di arrestare il thread associato al `watchId` indicato

Il terzo parametro **options** contiene i seguenti tre campi:

```
var options = {
     enableHighAccuracy: true/false
     timeout: 5000,
     maximumAge: 0,
     frequency : 1000     // 1 sec
}
```

enableHighAccuracy I sensori possono restituire le posizioni GPS in modo più approssimativo (7 cifre, Based GPS feature) oppure in modo più accurato (Built-in GPS feature) però con un maggior consumo di risorse (energia). Questa proprietà consente di impostare quale modalità utilizzare. Il default è false e va benissimo nella maggior parte dei casi, a meno che non servano altissimi livelli precisione. C'è però ancora un piccolo problema. Alcuni SO dispongono soltanto di una delle due modalità per cui se la richiesta usa una opzione diversa vanno in errore. La documentazione consiglia di utilizzare false e, in caso di errore, riprovare con true.

timeout (ms) indica il tempo max di attesa della risposta. Se la risposta non arriva entro questo tempo viene richiamata la procedura onError. Il valore di default è infinito, nel qual caso il metodo getCurrentPosition non ritorna nulla.

maximumAge (ms) indica al sensore un tempo max di cache dei valori restituiti nell'ultima lettura effettiva. Il default è 0 e significa che si chiede al sensore di restituire sempre valori effettivi.

frequency imposta il tempo tra una lettura del sensore e l'altra. Nel caso del GPS dalle API v3 in avanti questa opzione non è più supportata. E' invece supportata nel caso dell'accelerometro.

Esempio di accesso al sensore GPS

```
$(document).ready(function() {
 document.addEventListener('deviceready', function() {

    var watchId = null;
    $("#btnAvvia").on("click", startWatch)
    $("#btnArresta").on("click", stopWatch )

    function startWatch(){
       var options={ }
       watchId=navigator.geolocation.watchPosition(onSuccess, onError, options)
       if (watchID != null)
             notifica("Lettura Avviata");
    }
```

```
    function stopWatch(){
      if (watchID != null){
            navigator.geolocation.clearWatch(watchId);
            watchID=null;
            notifica("Lettura Arrestata");
        }
    }

    function onSuccess(position) {  }

    function onError(err) {
        visualizza("Errore: " + err.code + " - " + err.message);
    }

    function notifica(msg){
       navigator.notification.alert(
           msg,
           function () { },
           "Info",        // Titolo finestra
           "Ok"           // pulsante di chiusura
       );
    }

 });
});
```

Geolocation

Occorre includere il seguente plugin:

```
cordova plugin add cordova-plugin-geolocation
```

Prima di avviare la app occorre avere il sensore gps attivo. (come per tutti i sensori).

La funzione **onSuccess** riceve come parametro un oggetto **position** contenente le seguenti informazioni:

coords.latitude	double -> decimal degrees
coords.longitude	double -> decimal degrees
coords.altitude	double or null -> meters (solo per sensori. Nel caso di indirizzo IP è null)
coords.accuracy	double -> meters (es: ± 100.000 metri nel caso dell'indirizzo IP)
coords.altitudeAccuracy	meters -> (es: ± 100 metri)
coords.speed	double or null -> meters/second velocità con cui il dispositivo si sta muovendo
timestamp	long -> timestamp unix espresso in msec a partire dal 1/1/1970

La funzione **onError** riceve come parametro un oggetto **err** contenente le seguenti informazioni:

err.code codice dell'errore
err.message messaggio dell'errore

Possibili codici di errore :

- `PERMISSION_DENIED` (1) if the browser denies you access to their location.
- `POSITION_UNAVAILABLE` (2) if the network is down or the positioning satellites can't be contacted.
- `TIMEOUT` (3) if the network is up but it takes too long to calculate the user's position.

Accelerometro

Occorre includere il seguente plugin:

```
cordova plugin add cordova-plugin-device-motion
```

Al posto dell'oggetto **navigator.geolocation** si utilizza l'oggetto **navigator.accelerometer** che presenta tre metodi del tutto analoghi ai precedenti:

```
.getCurrentAcceleration (onSuccess, onError, options)
.watchAcceleration (onSuccess, onError, options)
.clearWatch(watchId)
```

La funzione **onSuccess** riceve come parametro un oggetto **Acceleration** che indica la forza di gravità che agisce sui tre lati del telefono e contiene le seguenti informazioni

.x	accelerazione lungo l'asse x (da 0 a ±9,81)
.y	accelerazione lungo l'asse y (da 0 a ±9,81)
.z	accelerazione lungo l'asse z (da 0 a ±9,81)
.timestamp	long -> timestamp unix espresso in msec a partire dal 1/1/1970

Un telefono in posizione portrait avrà :

.x = 0
.y = 9,81
.z = 0

Accesso alla fotocamera / photoGallery

https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-camera/

Occorre includere il seguente plugin:

```
cordova plugin add cordova-plugin-camera
```

Il metodo camera.getPicture(onSuccess, onError, cameraOptions)

E' il metodo fondamentale che consente di

- aprire la fotocamera per scattare una fotografia
- recuperare una fotografia all'interno dalla photoGallery del dispositivo.

Occorre preventivamente impostare le seguenti opzioni:

```
let cameraOptions = {
     sourceType: Camera.PictureSourceType.CAMERA
     destinationType: Camera.DestinationType.FILE_URI,
     quality: 50,
}
```

sourceType

- Se **sourceType** è uguale a **Camera.PictureSourceType.CAMERA** (default) Il metodo camera.getPicture() apre l'applicazione fotocamera predefinita del dispositivo e consente all'utente di scattare delle foto. Scattata la foto, l'applicazione della fotocamera si chiude e l'applicazione viene ripristinata richiamando automaticamente il metodo onSuccess a cui viene iniettata l'immagine.

- Se invece **sourceType** è uguale a **Camera.PictureSourceType.PHOTOLIBRARY** o **Camera.PictureSourceType.SAVEDPHOTOALBUM**, viene visualizzata una finestra di dialogo che consente all'utente di selezionare un'immagine esistente.
  Nel caso di Android i valori **SAVEDPHOTOALBUM** e **PHOTOLIBRARY** sono equivalenti.

destinationType

Definisce il formato con cui l'immagine sarà ritornata a onSuccess.

- **Camera.DestinationType.DATA_URL** fa si che l'immagine venga restituita a onSucces in formato base64. onSuccess la potrà utilizzare nel modo seguente:
  `.prop("src", "" + imageData)`
- **Camera.DestinationType.FILE_URI** fa si che venga restituito il path dell'immagine all'interno del file system locale. onSuccess potrà utilizzare l'immagine nel modo seguente:
  `.prop("src", imageData)`

Per evitare problemi di memoria, la documentazione consiglia l'utilizzo di **FILE_URI** (default) invece che **DATA_URL**. C'è però un problema: **FILE_URI** funziona perfettamente con la Camera ma NON funziona in caso di accesso alla photogallery dove restituisce un errore del tipo "Access denied" per problemi CORS: **strict-origin-when-cross-origin**

quality

E' un numero tra 0 e 100 (max quality) che indica la qualità dell'immagine restituita a onSuccess. Si tenga in conto che la risoluzione della fotocamera non è accessibile, per cui il metodo camera.getPicture(), prima di restituire l'immagine a onSuccess, applica una compressione pari al valore indicato. Il default è 50. **Nota:** le foto selezionate dalla galleria del dispositivo non vengono ridotte a una qualità inferiore, anche se viene specificato un parametro di qualità.

targetWidth e targetHeight

Consentono di riscalare l'immagine prima di restituirla a onSuccess

cameraDirection

Consente di scegliere quale fotocamera utilizzare (**BACK** / FRONT)

mediaType

Il default è **PICTURE**. Sono disponibili le opzioni **VIDEO** e **ALLMEDIA**.

Utilizzo onSuccess(imageData)

Una volta acquisita l'immagine, all'interno di onSuccess è possibile

- Renderizzare l'immagine ad esempio all'interno dell'attributo **src** di un tag <img>:
  `.prop("src", "" + imageData)`
  `.prop("src", imageData)`
- Salvare l'immagine in locale, ad esempio nel LocalStorage,
- Pubblicare l'immagine su un server remoto

Il metodo camera.cleanup()

Rimuove i file immagine intermedi che vengono conservati nella memoria temporanea dopo aver chiamato camera.getPicture.
Si applica solo quando il valore di Camera.sourceType è uguale a Camera.PictureSourceType.CAMERA e Camera.destinationType è uguale a Camera.DestinationType.FILE_URI.
Non è supportato nei sistemi Android