Chabert Audric
Pujol Jonathan

Tutorial Android/Phonegap

Nous avons choisi réalisé une application mobile pour tester les différences entre le développement Android Natif et le framework cross-platform Phonegap.

Pour démontrer les différences entre ces deux solutions, nous avons choisi de développer une application permettant de :

- Voir les images géolocalisées stockées sur son téléphone, dans une galerie
- Pour une image sélectionnée, trouver les photos les plus proches
- Enfin, afficher les coordonnées de la photo sur une carte du type Google Maps

L'interface se présente comme ceci :

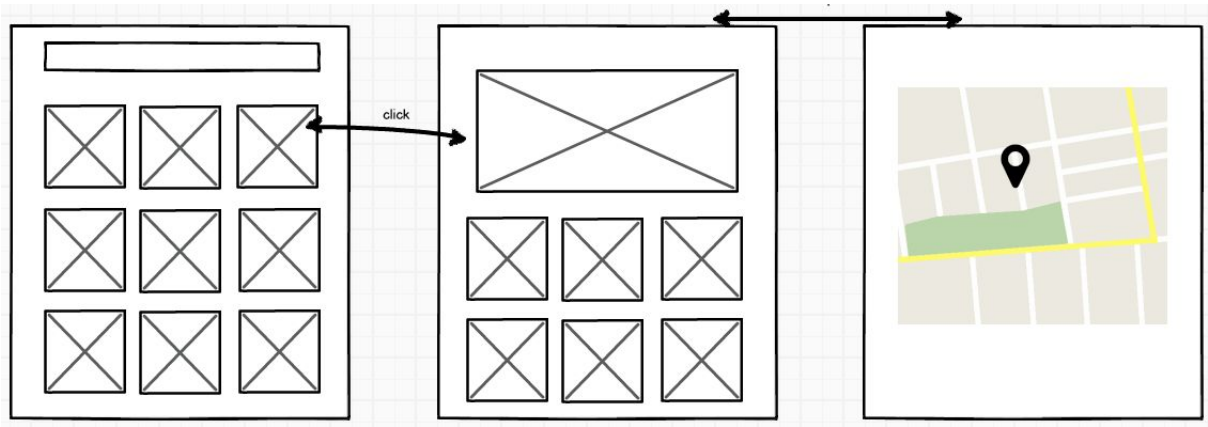


Figure 1 mockup interface

Voici un récapitulatif du fonctionnement de l'application :

La page d'accueil contiendra la liste de toutes les images géolocalisées trouvées par l'application. En cliquant sur l'une d'elle on change de page pour afficher l'image en plus grand, accompagné des photographies les plus proches d'elle. Enfin on peut accéder à une autre page contenant la position des images sur la carte google maps.

Partie Phonegap

L'environnement de travail est constitué de phonegap et de ses plugins, auquels se rajoutent d'autres librairies javascript. Noter que dans les exemples nous avons utilisé cordova, technologie encapsulée dans phonegap.

Pour créer et lancer un projet phonegap

Phonegap a comme pré-requis l'installation de nodejs, qui permet d'utiliser le gestionnaire de paquets npm. Aprè l'installation de ce dernier, faire dans une console classique (comparé à la console node.js) les instructions suivantes :

```
C:\> npm install -g phonegap
```

```
$ phonegap create my-app
$ cd my-app
$ phonegap run android
```

La commande *phonegap run android* lance l'éxécution de l'application sur un des appareil connecté au pc, il faut donc soit avoir un émulateur soit avoir un téléphone android. Nous avons utilisé pour nos tests un téléphone récent ainsi qu'une tablette. Nous avons aussi utilisé l'émulateur Genymotion https://www.genymotion.com/ , qui permet d'avoir une application fluide, plus proche d'un vrai appareil que d'un émulateur classique.

Il est possible de rencontrer une erreur lors du lancement de l'application, j'ai noté ci-dessous les détails et la solution pour ce problème.

problème : cordova run android ne trouve pas l'android virtual device de genymotion, il apparaît que le service adb est toujours en cours d'éxécution.

Erreur : ADB server didn't ACK, failed to start daemon.

Solution

1ère étape

Dans la console :

adb kill-server

adb start-server

Source :

http://stackoverflow.com/questions/5703550/my-eclipse-adb-server-didnt-ack-failed-to-start-daemon

https://developer.appcelerator.com/question/159658/cant-connect-genymotion-over-adb

2ème étape

Cordova run android –device

Source
:http://stackoverflow.com/questions/19767810/running-phonegap-on-device-no-device-found

Installation et utilisation d'un plugin Google Maps

Pour utiliser les fonctionnalités d'une map affichant les coordonnées GPS, nous avons utilisé un plugin appellé cordova-plugin-googlemaps, disponible à l'adresse : https://github.com/mapsplugin/cordova-plugin-googlemaps

Les étapes d'installation peuvent varier selon la méthode d'installation et la plateforme hôte. Sur windows et sans utiliser crosswalk, les étapes sont les suivantes :

- S'assurer d'avoir correctement installé et ajouté dans le path :
  - Java avec JAVA_HOME dans les variables d'environnement
  - Le SDK Android (avec ou sans IDE) mis à jour sur les paquets *Platform-tools* et *Build-tools*. On rajoutera dans le path les chemins aux dossiers *Platform-tools* et *Build-tools*
  - Ant
- Créer un projet Phonegap ou Cordova, en prenant soin d'ajouter la plateform android, si ce n'est pas déjà fait, avec la commande *cordova platform add android.* Noter que la syntaxe des commandes Phonegap et Cordova sont assez similaires
- Créer et utiliser les certificats :
  - Avec *keytool* :

```
C:\Program Files (x86)\Java\[your_jdk_version]\bin> keytool -list -v
-keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey
-storepass android -keypass android
```

  - Sur le site des API Google https://code.google.com/apis/console/
- Installer le plugin proprement dit, par exemple avec npm

```
C:\test\HelloMap> cordova plugin add cordova-plugin-googlemaps --variable
API_KEY_FOR_ANDROID="YOUR_API_KEY_IS_HERE"
```

- Utiliser l'exemple fourni, le fichier index.html. Ce fichier est le point de départ d'une application Phonegap
- Lancer le code avec les commandes citées précédemment

Ci dessous le code qui sert à ajouter un marqueur sur la map google :

```
map.addMarker({
        'position' : result.latNlng,
        'title' : "Hello Google Maps! PhoneGap-GoogleMaps-Plugin is easy!",
        'snippet' : 'Tap here, then next',
        'styles': {
                'text-align':'center',
                'font-style':'italic',
                'font-weight':'bold',
                'color':'red'
        },
        animation: plugin.google.maps.Animation.DROP
```

```
        }, onMarkerAdded);

function onMarkerAdded(marker){
        marker.showInfoWindow();
        marker.addEventListener(plugin.google.maps.event.INFO_CLICK,
onInfoWndClicked);
function onInfoWndClicked(marker){
        marker.hideInfoWindow();
}
```

Le résultat :

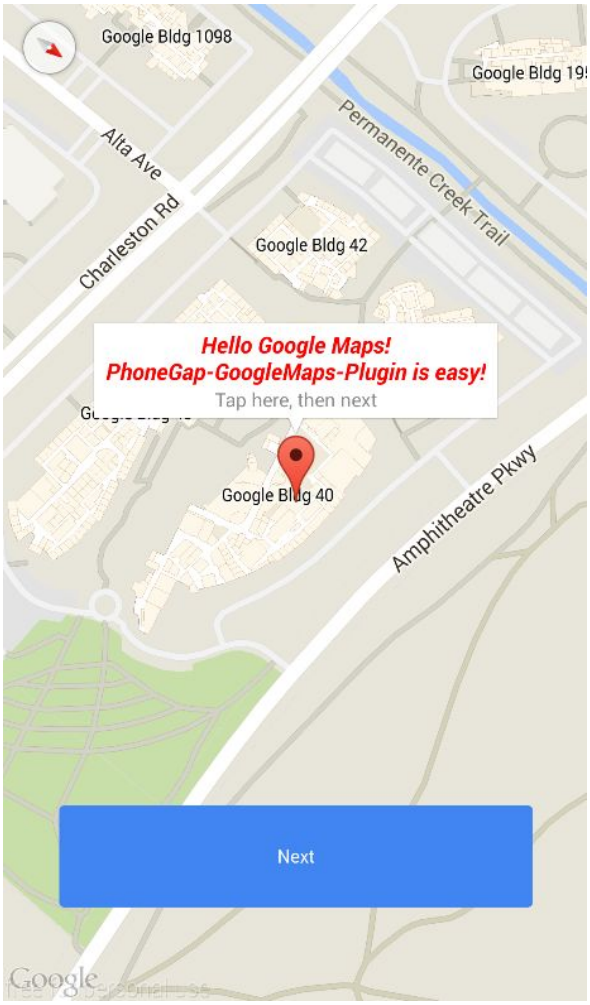


Figure 2 exemple d'affichage d'un marqueur sur google maps plugin

Installation et utilisation de photoswipe

Ce projet permet de réaliser simplement des galleries d'images.

La notice d'installation est disponible ici http://photoswipe.com/documentation/getting-started.html , on l'installe comme ceci :

- Télécharger et décompresser le dossier comprenant le code de Photoswipe
- Inclure les fichiers css et javascript contenu dans le dossier décompressé

```
<!-- Core CSS file -->
<link rel="stylesheet" href="path/to/photoswipe.css">
```

```
<!-- Skin CSS file (styling of UI - buttons, caption, etc.)
     In the folder of skin CSS file there are also:
     - .png and .svg icons sprite,
     - preloader.gif (for browsers that do not support CSS animations) -->
<link rel="stylesheet" href="path/to/default-skin/default-skin.css">

<!-- Core JS file -->
<script src="path/to/photoswipe.min.js"></script>

<!-- UI JS file -->
<script src="path/to/photoswipe-ui-default.min.js"></script>
```

- Ajouter la structure d'exemple qui contiendra les images et les éléments graphiques de la gallerie, qui donne un résultat comme suit :

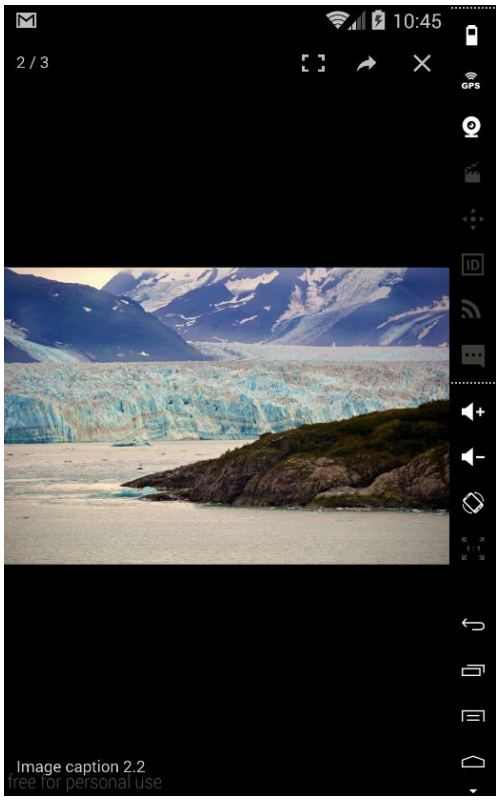


Figure 3 et 4 exemple d'affichage de la galerie et d'une image séléctionnée, le bouton maps est un lien vers la page utilisant google maps

Lecture des données de géolocalisation

Les informations qui permettent de localiser une photo se retrouvent à l'intérieur du fichier de photo, au format exif. Pour récupérer ces informations, j'ai utilisé un code disponible à l'adresse https://github.com/guilhermefarias/cordova-exif

Pour l'utiliser il faut d'abord

Ensuite on peut utiliser les fonctionnalités demandées, voici un cours exemple qui démontre des capacités de ce code.

```
var options = {
        quality: 90,
        sourceType: 2,
        destinationType: 1,
};

function onSuccess(imageURI) {
        CordovaExif.readData(imageURI, function(exifObject) {
         console.log(exifObject);
        });
};

function onFail(message) {
    console.log('Failed because: ' + message);
};

navigator.camera.getPicture(onSuccess, onFail, options);
```

la ligne *console.log(exifObject)* affiche dans la console du navigateur toutes les informations obtenues, qui peuvent comprendre donc les coordonnées GPS mais aussi d'autres informations comme le temps d'exposition de la photographie.

Avantages et inconvénients

L'utilisation de langages web comme HTML et CSS permettent d'avoir rapidement une application utilisable. On peut aussi remarquer le grand nombre de plugins ou de librairies, notamment en javascript. D'un autre côté la gestion de l'affichage peut s'avérer plus contraignante, sans parler de la réactivité qui s'en trouve amoindrie.

Nous avons pu remarquer de nombreuses différences lors de l'implémentation des fonctionnalités, la plus important au niveau Phonegap est un « bug » d'affichage : si on utilise un lien pour passer de la galerie à la carte, lors du retour sur l'ancienne page la carte est toujours affiché en arrière-plan. Sous une application Android native il n'y a pas cet inconvénient

Un désavantage de phonegap est qu'il ne prend pas en charge simplement l'orientation de l'écran. Il est assez difficile d'offrir le même rendu sur navigateur, même si certains éléments natifs HTML/CSS permettent de contourner le problème. Citons les directives CSS @media et les unités vh et vw, vmin, vmax. Les premières permettent de décider de l'application d'une règle (qui servira ici à déterminer la taille et la position des éléments) selon la hauteur et la largeur du navigateur.

Les secondes sont des unités de taille permettent par exemple de déterminer une taille selon le côté le plus petit du navigateur.

Pour offrir un rendu comparable à des applications natives, il est recommandé d'utiliser des plugins spécifiques comme nous l'avons fait. De l'autre côté, toute la partie algorithme, notamment pour le calcul de la distance séparant deux images, se fait aussi simplement sur les deux environnements.

Au final, l'utilisation de langages web pour réaliser une application peut s'avérer plus contraignante que sur un environnement natif.

References

Utilisation des données exif contenant la géolocalisation

Source : https://github.com/guilhermefarias/cordova-exif

Plugin googlemaps

https://github.com/mapsplugin/cordova-plugin-googlemaps

Tutoriel photoswipe

http://photoswipe.com/documentation/getting-started.html

Calcul de distance entre points (pour déterminer quelles sont les photos les plus proches), il s'agit d'une version de l'algorithme réalisé en javascript

Source : http://stackoverflow.com/questions/5260423/torad-javascript-function-throwing-error

Partie Android

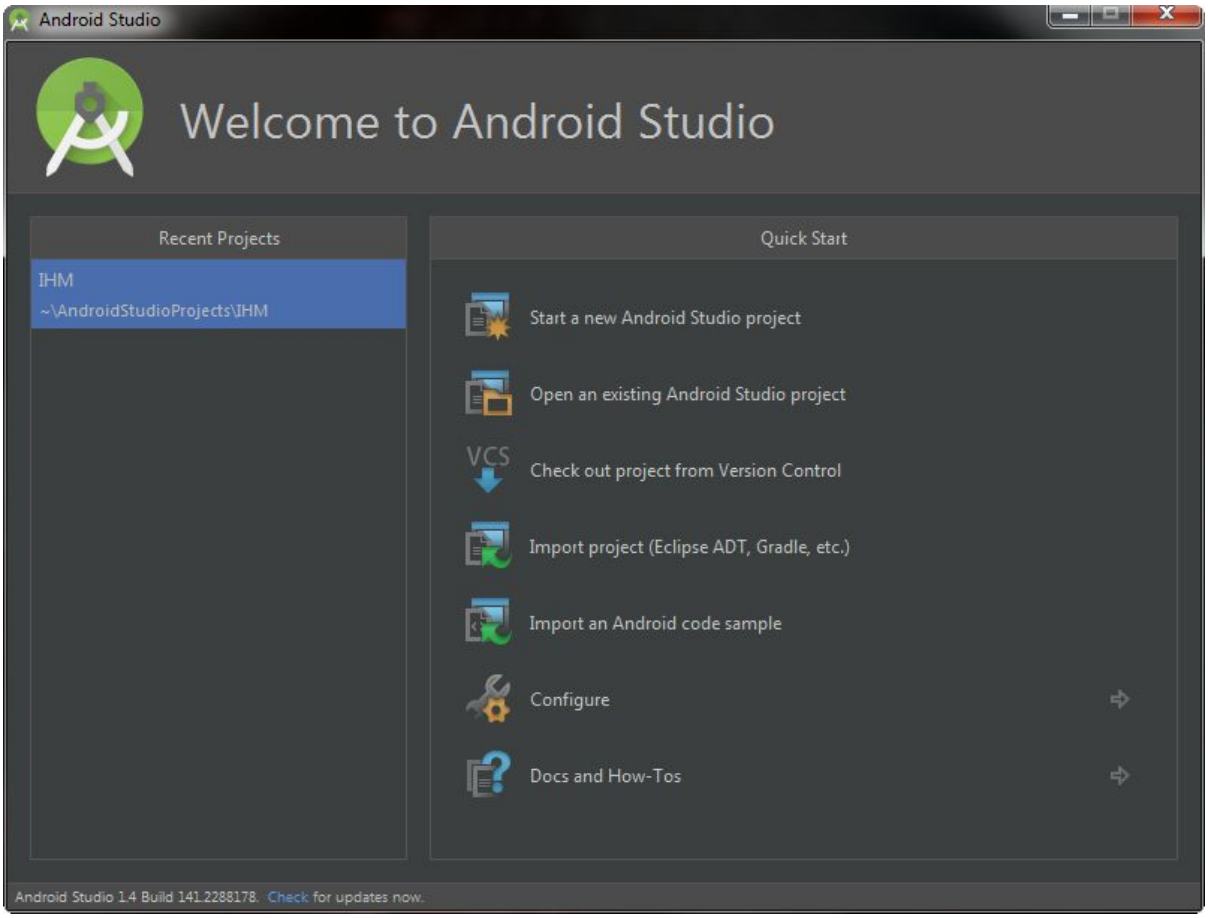
Bonjour à tous, aujourd'hui nous allons développer une application qui va se servir de la géolocalisation des photos pour connaître les quelles ont été prises proche d'une photo en particulier.
Ce tutoriel sera une explication pas à pas qui va nous permettre d'obtenir le résultat attendu.

Notre projet

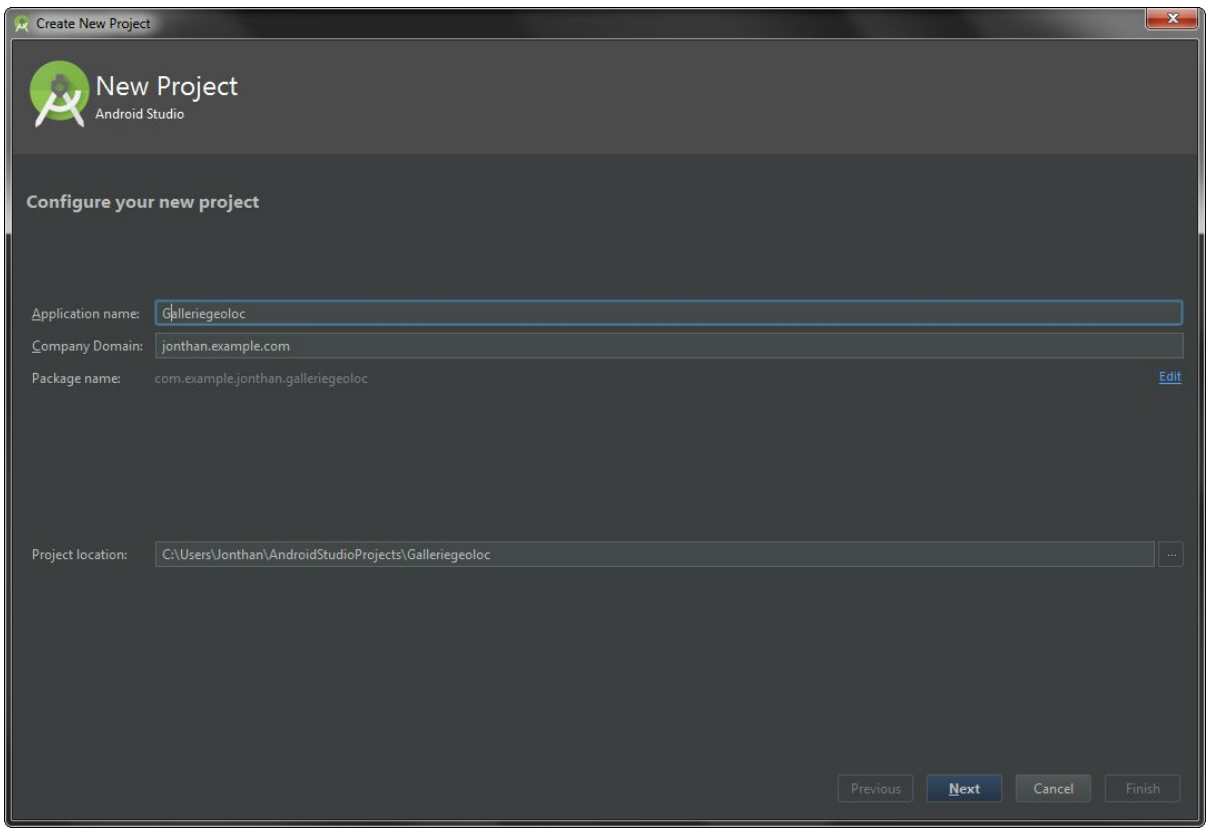
Pour la réalisation de notre projet nous allons avoir besoin d'Android Studio, qui est l'IDE (interface de développement) dédié à la réalisation de projet Android. Vous pouvez le télécharger à l'adresse suivante : https://developer.android.com/sdk/index.html.
Il faut également penser à installer les SDK d'android et de Java.

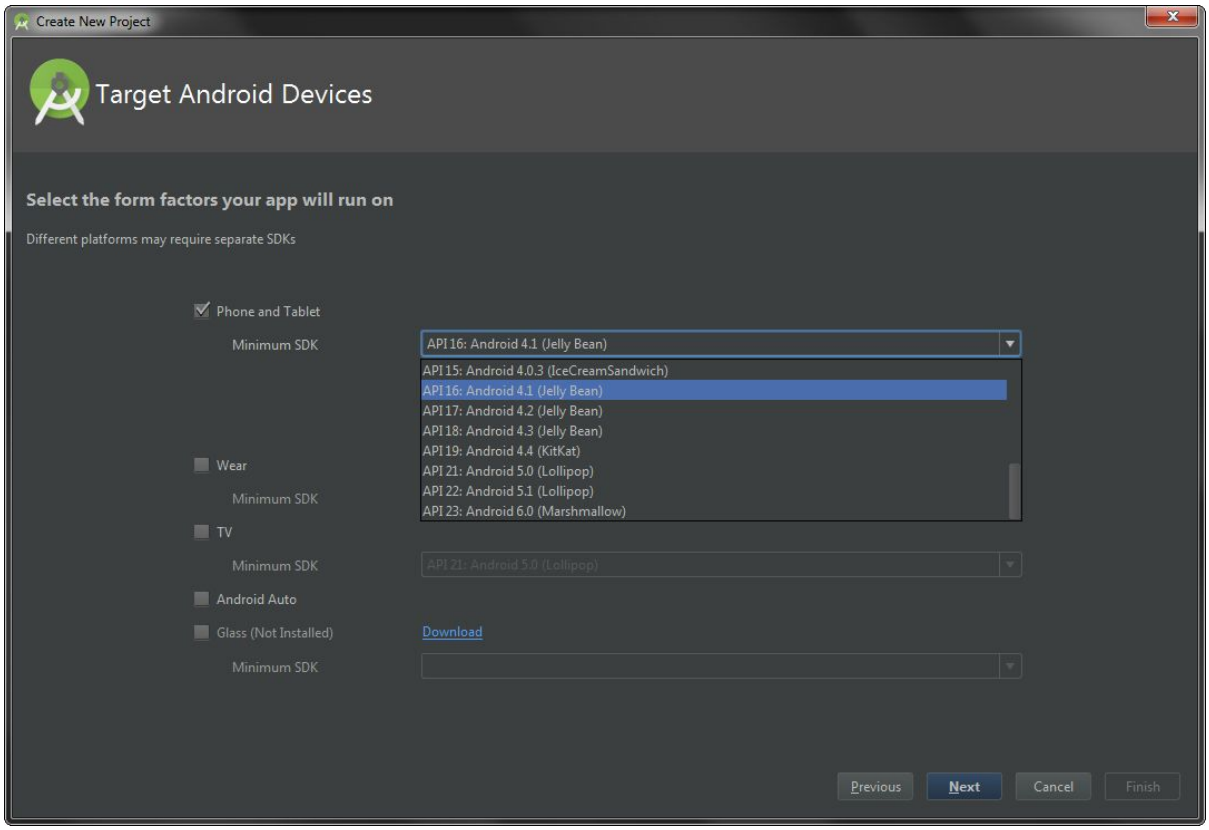
Une fois que toutes ces étapes sont éffectué nous allons pouvoir commencer.

Je vous invite donc à lancer Android Studio.
Vous devez alors tomber sur ma page suivante :

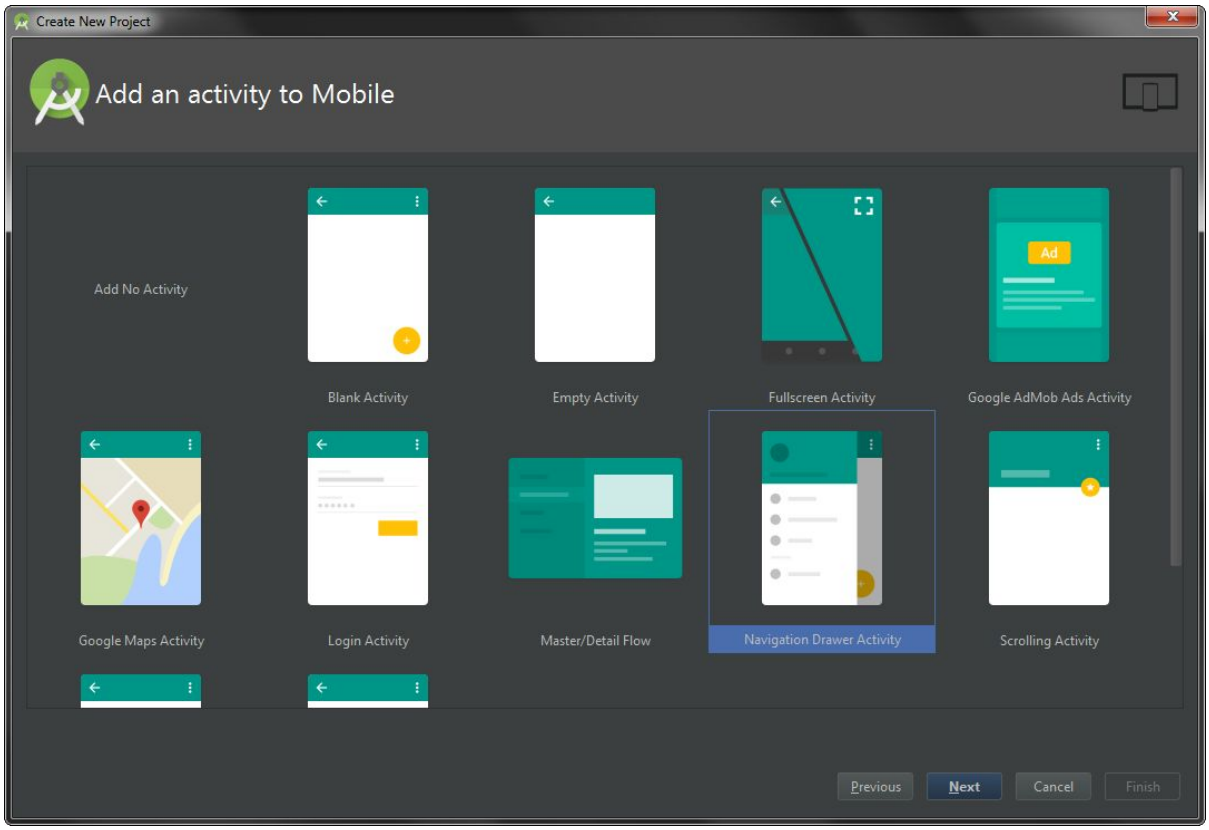


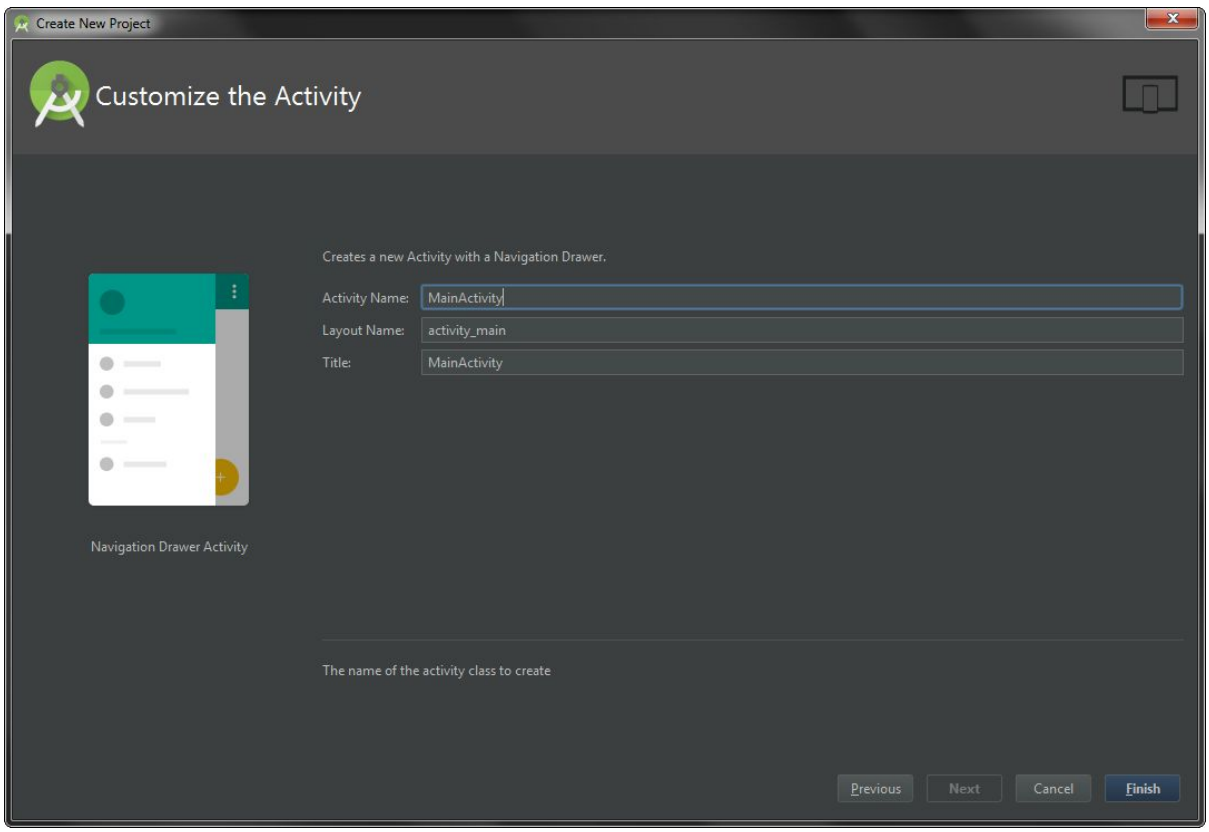
Clicker sur **Start a new Android Studio Project**





Choisier un nom (celui que vous souhaitez, cela n'a pas d'importance pour la suite) et l'API minimum du projet. Tout ce qui va être détaillé par la suite est supporté jusqu'à la version de **l'API16**



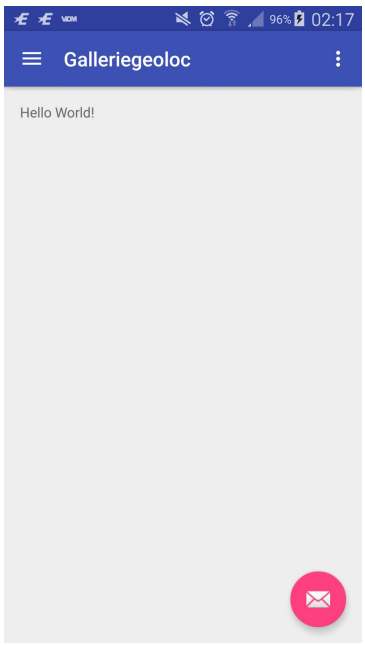


Il faut maintenant choisir une activité de base, nous allons prendre **Navigation Drawer Activity.** Cela va nous permettre d'obtenir une activité possédant déjà un menu et un bouton de settings. Nous n'allons pas nous en servir pour ce tuto. Mais cela pourrait être utile si nous voulions poursuivre plus tard.

Nous allons également hériter de certain élément qui ne nous intéresse pas, mais nous allons les supprimer.
Il faut également lui **donner un nom**, personnellement j'ai décidé de garder celui par défaut.
Une fois choisie clicker sur **Finish**.

Faire disparaitre les éléments en trop

Une fois arrivée à ce stade, il est tout à fait possible de compiler et de lancer le projet.
Si vous le faites vous allez obtenir ceci :



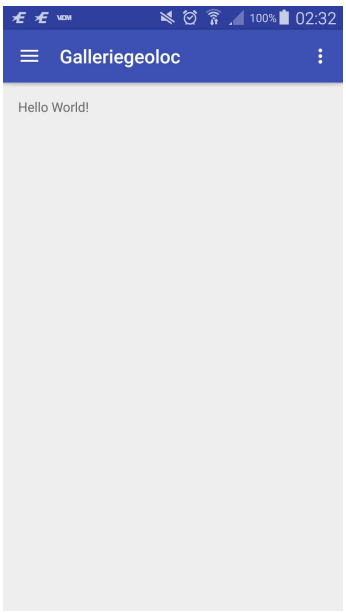
Nous avons bien le menu (en haut à gauche) et les settings (en haut à droite), mais comme je vous l'avez dit on a également des éléments superflus : oui je parle de toi le bouton rose avec l'icon de lettre.
Nous allons donc le supprimer. Pour cela il faut se rendre dans **res/layout/app_bar_main.xml** et supprimer le bout de code suivant qui y fait référence. Les fichiers xml permet de gérer les différents éléments graphiques qui nous donne le rendu final.

```
<android.support.design.widget.FloatingActionButton
android:id="@+id/fab"
    android:layout_width="wrap_content"
android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
android:layout_margin="@dimen/fab_margin"
    android:src="@android:drawable/ic_dialog_email" />
```

Il faut également penser à supprimer le bout de code qui y fait référence dans MainActivity.java. Sinon le système ne le trouvera pas et le projet ne pourra pas compiler.

```
FloatingActionButton fab = (FloatingActionButton)
findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
    }
});
```

Maintenant si on compile on obtient :



On constate bien que le bouton rose a disparu

Faire une galerie des photos géolocalisées

Maintenant que nous avons une base propre, nous allons attaquer le vif du sujet. Tout d'abord notre application ne s'intéresse qu'aux photos géolocalisées. Donc nous allons créer une galerie qui ne s'intéresse qu'à ce type de photo.

Pour obtenir une galerie qui ressemble à quelque chose, nous allons utiliser une GridView pour ranger nos photos. Comme son nom l'indique une GridView est une grille dans la quelle nous allons pouvoir insérer dans une case une photo.

Tout d'abord commençons par ajouter une **GridView** à notre fichier xml. Allez dans **res/layout/content_main.xml** et ajoutez y une GridView de la manière suivante :

```
<GridView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:numColumns="3"
    android:horizontalSpacing="5dp"
    android:verticalSpacing="5dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
    />
```

Ce bout de code ajoute une GridView qui prend la taille du parent, qui possède 3 colonnes et qui a une séparations entre chaque éléments de 5dp. Si vous le souhaitez, il est tout à fait possible de modifier les paramètres de cette GridView pour obtenir un autre rendu.

Pensez également à supprimer le **TextView** “Hello World” que ne nous serra pas utile.

Maintenant il va falloir récupérer toutes les photos géolocalisées pour pouvoir les insérer dans cette GridView.

Avant de récupérer les photos géolocalisées, il faut déjà être capable de récupérer les photos du téléphone. Et ce n’est pas quelque chose d’évident. Heureusement pour nous, des gens se sont déjà penché sur la question et nous fournisse une méthode qui fonctionne. Ajoutez le code suivant dans votre MainActivity.java

```
//permet de récupérer tous les paths des photos contenus dans le
téléphone (ne pas oublier d'ajouter la permission dans le
manifest.)
public ArrayList<String> getAllShownImagesPath(Activity activity) {
    Uri uri;
    Cursor cursor;
    int column_index_data, column_index_folder_name;
    ArrayList<String> listOfAllImages = new ArrayList<String>();
    String absolutePathOfImage = null;
    uri =
android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI;
    String[] projection = { MediaStore.MediaColumns.DATA,
            MediaStore.Images.Media.BUCKET_DISPLAY_NAME };
    cursor = activity.getContentResolver().query(uri, projection,
null, null, null);
    column_index_data =
cursor.getColumnIndexOrThrow(MediaStore.MediaColumns.DATA);
    column_index_folder_name = cursor
.getColumnIndexOrThrow(MediaStore.Images.Media.BUCKET_DISPLAY_NAME)
;
    while (cursor.moveToNext()) {
        absolutePathOfImage = cursor.getString(column_index_data);
        listOfAllImages.add(absolutePathOfImage);
```

```
    }
    return listOfAllImages;
}
```

Il faut cependant ajouter la permission `<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />` dans le manifest android, pour permettre à votre application de lire des informations sur tout le téléphone.

Cette méthode permet en réalité d'obtenir une list de String qui contient le path de toutes les photos du téléphone.

Maintenant que l'on a accès à toutes les photos il faut faire le tri pour garder que celle qui sont géolocalisées.
Pour connaître la position d'où a été prise une photo, c'est très simple. Elle est contenu dans les méthadonnées de la photo aux quelles on accède grâce à une interface EXIF. Ajouter le code suivant dans la méthode **onCreate**

```
List<String> pathPicture = getAllShownImagesPath(this);
final ArrayList<String> pathPictureLocalized = new ArrayList<>();
for(String path : pathPicture) {
    try {
        ExifInterface exif = new ExifInterface(path);
        float[] latLong = new float[2];
        exif.getLatLong(latLong);
        if(latLong[0] != 0.0 && latLong[1] != 0.0) {
            pathPictureLocalized.add(path);
        }
    } catch (Exception e) {
        return;
    }
}
```

Pour chaque path des photos du téléphone on ouvre une interface EXIF, on récupère la variable latLong, qui est un tableau qui contient la latitude et la longitude.
Une fois ce tableau obtenu, on vérifie que la latitude et la longitude ne sont pas toutes les deux à zéro (ce qui signifie en réalité que la photo n'est pas géolocalisée) et si ce n'est pas le cas on ajoute le path à une nouvelle liste.

Maintenant qu'on possède le path de toutes les photos qui sont géolocalisées, ils ne nous reste plus qu'à les afficher.
Pour cela ajouter le code suivant à la suite dans **onCreate :**

```
ArrayList<Bitmap> bitmapsPicLoc = new ArrayList<>();
for(String s : pathPictureLocalized) {
    File image = new File(s);
    //réduit la taille des images
    BitmapFactory.Options bmOptions = new BitmapFactory.Options();
    bmOptions.inSampleSize = 8;
```

```
    Bitmap bitmap = BitmapFactory.decodeFile(image.getAbsolutePath(),
bmOptions);
    Bitmap bitmapRotated = bitmap;
    //Permet de remettre à l'endroit les images
    try {
        ExifInterface ei = new ExifInterface(s);
        int orientation =
ei.getAttributeInt(ExifInterface.TAG_ORIENTATION,
ExifInterface.ORIENTATION_NORMAL);
        switch (orientation) {
            case ExifInterface.ORIENTATION_ROTATE_90:
                bitmapRotated = rotateImage(bitmap, 90);
                break;
            case ExifInterface.ORIENTATION_ROTATE_180:
                bitmapRotated = rotateImage(bitmap, 180);
                break;
            case ExifInterface.ORIENTATION_ROTATE_270:
                bitmapRotated = rotateImage(bitmap, 270);
                break;
        }
    } catch (Exception e) {
        return;
    }
    bitmapsPicLoc.add(bitmapRotated);
}
//On récupère la GridView, et on y ajoute un adapter (nécessaire
pour afficher des images)
GridView gridview = (GridView) findViewById(R.id.gridview);
gridview.setAdapter(new ImageAdapter(this, bitmapsPicLoc));
```

Le code est assez bien commenté, il permet de faire un traitement sur les images, qui sont trop grosse pour être afficher dans une GridView et de les remettre dans le bon sens pour pouvoir les afficher correctement.
On passe ensuite les photos, sous forme de bitmap, a un Adpater qui va transformer la photo en view pour pouvoir l'afficher dans la Grid.
Il est a noté que tout le traitement peut se faire dans l'adapter, mais cela ralenti très fortement la navigation. En effet, **une GridView ne charge que ce qu'il affiche** donc si on slide il faudrait refaire tout le traitement à chaque fois. Il est donc préférable de le faire avant de l'envoyer à l'adapter.

Pour créer l'adapter, ajouter un classe ImageAdapter.java à votre projet et ajoutez y le code suivant :

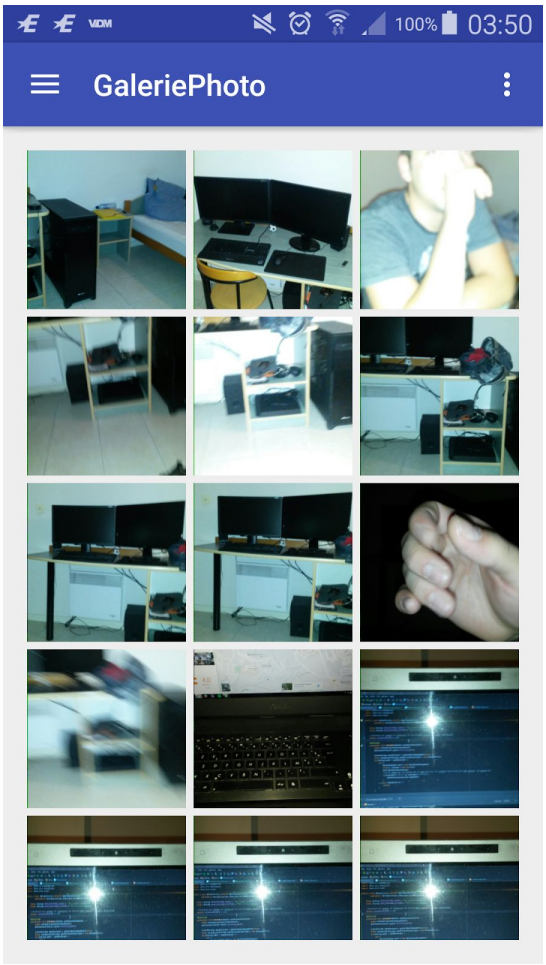
```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;
    List<Bitmap> bitmapList;

    public ImageAdapter(Context c,List<Bitmap> bitmapList) {
        mContext = c;
```

```
        this.bitmapList = bitmapList;
    }
    //permet de connaitre le nombre d'élément dans la liste et donc
le nombre de case de la grid à créer
    public int getCount() {
        return bitmapList.size();
    }
    public Object getItem(int position) {
        return null;
    }
    public long getItemId(int position) {
        return 0;
    }
    // create a new ImageView for each item referenced by the
Adapter
    public View getView(int position, View convertView, ViewGroup
parent) {
        ImageView imageView;
        if (convertView == null) {
            // if it's not recycled, initialize some attributes
            imageView = new ImageView(mContext);
            GridView gridView = (GridView) parent;
            //Permet de rendre les photos carrés
            int size = gridView.getColumnWidth();
            imageView.setLayoutParams(new
GridView.LayoutParams(size,size));
           imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        } else {
            imageView = (ImageView) convertView;
        }
        imageView.setImageBitmap(bitmapList.get(position));
        return imageView;
    }
}
```

Pour que votre code fonctionne il faudra égalemment ajouter la méthode rotate(..) dans MainActivity.java

```
public Bitmap rotateImage(Bitmap source, float angle)
{
    Matrix matrix = new Matrix();
    matrix.postRotate(angle);
    return Bitmap.createBitmap(source, 0, 0, source.getWidth(),
source.getHeight(), matrix, true);
}
```

Arrivé à ce stade, vous devriez avoir un tableau de photo. Pour moi ça donne :



Trouver les photos proches et les afficher.

Maintenant que nous avons réussi à afficher toutes nos photos géolocalisées, ça serait bien de trouver les plus proches et de les afficher sur une nouvelle page et de les localiser sur une map.

Commençons par la sélection d'une image

La GridView permet d'attacher un listener sur ses éléments ce qui est exactement ce que nous voulons. On vient donc ajouter à la suit de **onCreate**

```
gridview.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v,
                            int position, long id) {
```

Je vous mets la suite du code commenté

```
gridview.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View v,
                            int position, long id) {
        //On crée un nouvelle activité pour afficher les photos
proches.
        Intent intent = new Intent(MainActivity.this,
SwipeViewActivity.class);
        ArrayList<String> pathPictureClose = new ArrayList<>();
        // On teste les photos pour savoir les quelles sont assez
proches
        float[] latLongP0 = new float[2];
        try {
            ExifInterface exif = new
ExifInterface(pathPictureLocalized.get(position));
            exif.getLatLong(latLongP0);
        } catch (IOException e) {
            return;
        }
        for (String s : pathPictureLocalized) {
            try {
                ExifInterface exif = new ExifInterface(s);
                float[] latLong = new float[2];
                exif.getLatLong(latLong);
                if (distance(latLongP0, latLong) < distanceForGPS &&
!s.equals(pathPictureLocalized.get(position))) {
                    pathPictureClose.add(s);
                }
            } catch (IOException e) {
                return;
            }
        }
        //On envoie toutes les photos assez proches
        intent.putExtra(PATH_PICTURE_CLOSE, pathPictureClose);
        //On envoie la photo selectionnée
        intent.putExtra(PATH_ORIGINAL_PICTURE,
pathPictureLocalized.get(position));
        startActivity(intent);
    }
});
```

Pour tester la distance, on utilise une nouvelle fois les données EXIF. On va calculer la distance entre notre photo sélectionnée et toutes les photos. Si la distance est inférieur à la distance choisie dans la variable distanceForGPS, le path est ajouté dans une liste.
Cette liste est alors transmise à une nouvelle activité SwipeViewActivity par la variable PATH_PICTURE_CLOSE, tout comme le path de la photo sélectionnée par la variable PATH_ORIGINAL_PICTURE.

Voici la méthode distance qui est la transcription d'un algorithme pour calculer des distances à la surface de la Terre.

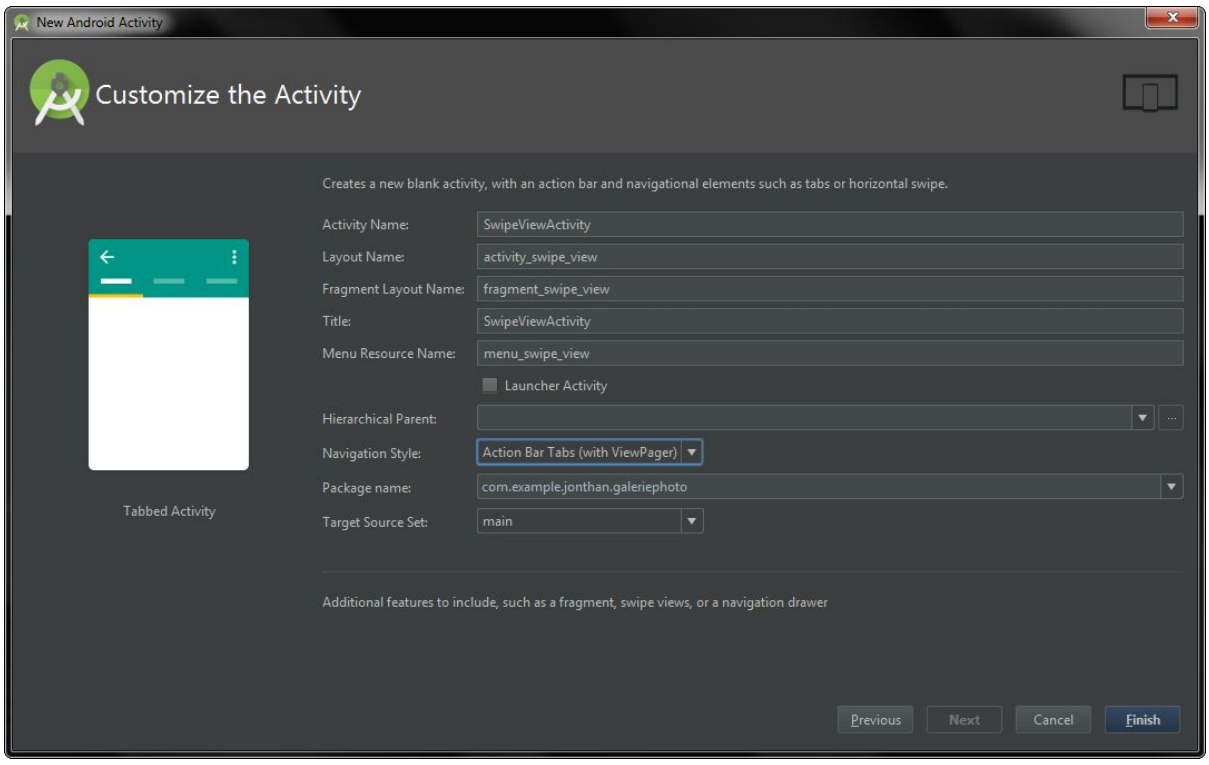
```
public double distance(float[] A, float[] B) {
    double latA = java.lang.Math.toRadians(A[0]), latB =
java.lang.Math.toRadians(B[0]), longA =
java.lang.Math.toRadians(A[1]), longB =
java.lang.Math.toRadians(B[1]);
    return 6371 * java.lang.Math.acos(java.lang.Math.cos(latA) *
java.lang.Math.cos(latB) * java.lang.Math.cos(longB -
longA)+java.lang.Math.sin(latA) * java.lang.Math.sin(latB));}
```

Penser également à déclarer les variables en en-tête de MainActivity.java :

```
final String PATH_PICTURE_CLOSE = "The closest picture";
final String PATH_ORIGINAL_PICTURE = "The picture choosen";
```

Maintenant que les informations sont envoyées vers une nouvelle activité, nous devons la créer. Pour cela nous allons utiliser une activité proposée par Android Studio directement la Tabbed Activity, qui va nous permettre de faire un swipe entre la GridView résultat et la carte.

Pour cela, aller dans **file/New.../Activity/Gallery** et la choisissez Tabbed Activity.
Vous devriez tomber sur la page suivante :



Pensez à changer le nom et à bien sélectionner pour le champ Navigation Style : **Action Bar Tabs (with ViewPager).**

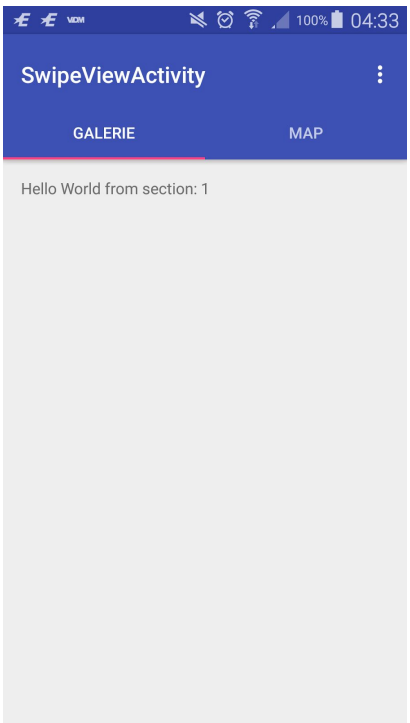
Supprimer le bouton rose comme précédemment et poursuivons.

L’affichage dans cette activité se fait à partir de la classe statique PlaceHolderStatement.

Mais d’abord modifions les méthodes de SectionPagerAdapter comme de la sorte, car nous n’avons que deux pages à afficher.

```
@Override
public int getCount() {
    // Show 2 total pages.
    return 2;
}

@Override
public CharSequence getPageTitle(int position) {
    switch (position) {
        case 0:
            return "Galerie";
        case 1:
            return "Map";
    }
    return null;
}
```

A ce moment si vous cliquez sur une image vous allez obtenir le résultat suivant :



Maintenant remplissons ces sections. Comme je l’ai dit précédemment ça se passe dans la classe PlaceHolderStatement plus précisément dans la méthode **onCreateView.**

Dans cette méthode il y a deux cas possibles, c'est pourquoi nous allons utiliser un switch. Dans le premier on veut afficher les photos proches sous forme de GridView et dans le second une map qui recense la position de toutes les photos.

Commençons par le premier, qui ressemble fortement à ce qu'on a fait avant. A ceci près que le layout n'est pas directement dans le xml de l'activité mais dans un **fragment_grid.xml**.

```
View rootView;

switch (getArguments().getInt(ARG_SECTION_NUMBER)) {
   case 1 :
       rootView = inflater.inflate(R.layout.fragment_grid,
container, false);

       File imageOri = new File(pathOriginalPicture);
       BitmapFactory.Options bmOptionsOri = new
BitmapFactory.Options();
       bmOptionsOri.inSampleSize = 2;
       Bitmap bitmapOri =
BitmapFactory.decodeFile(imageOri.getAbsolutePath(), bmOptionsOri);
       Bitmap bitmapRotatedOri = bitmapOri;
       try {
           ExifInterface ei = new
ExifInterface(pathOriginalPicture);
           int orientation =
ei.getAttributeInt(ExifInterface.TAG_ORIENTATION,
ExifInterface.ORIENTATION_NORMAL);

           //Permet de rotationner les images
           switch (orientation) {
               case ExifInterface.ORIENTATION_ROTATE_90:
                   bitmapRotatedOri = rotateImage(bitmapOri, 90);
                   break;
               case ExifInterface.ORIENTATION_ROTATE_180:
                   bitmapRotatedOri = rotateImage(bitmapOri, 180);
                   break;
               case ExifInterface.ORIENTATION_ROTATE_270:
                   bitmapRotatedOri = rotateImage(bitmapOri, 270);
                   break;
           }
       } catch (Exception e) {
           return rootView;
       }
       ImageView imageView = (ImageView)
rootView.findViewById(R.id.imageoriginal);
       //permet de prendre toute la place dispo en sclalant à
partir du centre
       imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
```

```
        imageView.setImageBitmap(bitmapRotatedOri);


        GridView gridView = (GridView)
rootView.findViewById(R.id.gridfragment);
        ArrayList<Bitmap> bitmapsPicClose = new ArrayList<>();
        for(String s : pathPictureClose) {
            File image = new File(s);
            BitmapFactory.Options bmOptions = new
BitmapFactory.Options();
            bmOptions.inSampleSize = 8;
            Bitmap bitmap =
BitmapFactory.decodeFile(image.getAbsolutePath(), bmOptions);
            Bitmap bitmapRotated = bitmap;
            try {
                ExifInterface ei = new ExifInterface(s);
                int orientation =
ei.getAttributeInt(ExifInterface.TAG_ORIENTATION,
ExifInterface.ORIENTATION_NORMAL);

                //Permet de rotationner les images
                switch (orientation) {
                    case ExifInterface.ORIENTATION_ROTATE_90:
                        bitmapRotated = rotateImage(bitmap, 90);
                        break;
                    case ExifInterface.ORIENTATION_ROTATE_180:
                        bitmapRotated = rotateImage(bitmap, 180);
                        break;
                    case ExifInterface.ORIENTATION_ROTATE_270:
                        bitmapRotated = rotateImage(bitmap, 270);
                        break;
                }
            } catch (Exception e) {
                return rootView;
            }
            bitmapsPicClose.add(bitmapRotated);
        }
        gridView.setAdapter(new ImageAdapter(getContext(),
bitmapsPicClose));
        return rootView;
```

On trouve deux fois des rotations et des affichages car, en plus d'afficher les photos dans une GridView, j'affiche également la photo sélectionnée.

Voici le code de **fragment_grid.xml** :

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
tools:context="com.example.jonthan.ihm.GridFragment">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">


        <ImageView
            android:id="@+id/imageoriginal"
            android:layout_width="match_parent"
            android:layout_height="200dp"
            android:layout_marginBottom="5dp"/>


        <GridView
xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/gridfragment"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:numColumns="3"
            android:horizontalSpacing="5dp"
            android:verticalSpacing="5dp"
            android:stretchMode="columnWidth"
            android:gravity="center"
            />
    </LinearLayout>

</FrameLayout>
```

Il faut également pour que ça fonctionne, récupérer les path envoyés précédement, pour ça il faut aller dans le **onCreate** de **SwipeViewActivity** et ajouter :

```
Intent intent = getIntent();
if (intent != null) {
    pathPictureClose =
intent.getStringArrayListExtra(PATH_PICTURE_CLOSE);
    pathOriginalPicture =
intent.getStringExtra(PATH_ORIGINAL_PICTURE);
}
```

Ainsi qu'en variable de classe :

```
final String PATH_PICTURE_CLOSE = "The closest picture";
final String PATH_ORIGINAL_PICTURE = "The picture choosen";
private static ArrayList<String> pathPictureClose;
private static String pathOriginalPicture;
```

De cette manière on stocke le path de toutes les photos proches dans pathPictureClose et le path de la photo sélectionnée dans pathOriginalPicture.

Afficher la carte

Pour terminer il nous reste à afficher les points sur la carte. C'est la case 2 :

```
case 2 :
    rootView = inflater.inflate(R.layout.fragment_maps, container,
false);
    // Obtain the SupportMapFragment and get notified when the map
is ready to be used.
    SupportMapFragment mapFragment = (SupportMapFragment)
getChildFragmentManager().findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
    return rootView;
```

On remarque que l'on crée une nouvelle fois un nouveau layout **fragment_maps.xml** dont voici le code :

```
<fragment
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:map="http://schemas.android.com/apk/res-auto"
android:layout_width="match_parent"
    android:layout_height="match_parent" android:id="@+id/map"
tools:context=".SwipeViewActivity"
    android:name="com.google.android.gms.maps.SupportMapFragment" />
```

C'est un fragment qui ne fait qu'afficher une carte.

Il faut toutefois ajouter un fichier contenant un clé, si l'on souhaite utiliser GoogleMaps. Dans **res/value** ajouter le fichier google_maps_api.xml :

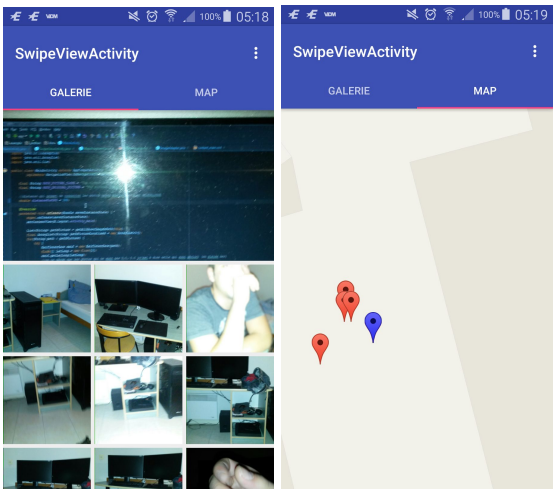
```
<resources>
      <string name="google_maps_key" translatable="false"
templateMergeStrategy="preserve">
"Your_Key_Here"
    </string>
</resources>
```

Vous pouvez obtenir cette clé à l'adresse suivante : https://console.developers.google.com dans API et Authentification -> API

Maintenant il ne reste plus qu'à ajouter des marqueurs pour toutes les photos

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    for(String s :pathPictureClose) {

        try {
            ExifInterface exif = new ExifInterface(s);
            float[] latLong = new float[2];
            exif.getLatLong(latLong);
            LatLng marker = new LatLng(latLong[0], latLong[1]);
            mMap.addMarker(new MarkerOptions().position(marker));
        } catch (Exception e) {
            return;
        }
    }

    try {
        ExifInterface exif = new ExifInterface(pathOriginalPicture);
        float[] latLong = new float[2];
        exif.getLatLong(latLong);
        LatLng marker = new LatLng(latLong[0], latLong[1]);
        mMap.addMarker(new
MarkerOptions().position(marker).icon(BitmapDescriptorFactory.defau
ltMarker(BitmapDescriptorFactory.HUE_BLUE)));
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(marker,
15));
    } catch (Exception e) {
        return;
    }
}
```

Voila on est arrivé au bout de ce tuto. Vous avez maintenant une application qui répond à notre besoin initial. Vous pouvez également jouer avec les layouts pour changer l'affichage selon la plateforme.



Avantages et inconvénients

Le plus gros avantage d'Android et le plus trivial, c'est que celui-ci implémente toutes les fonctionnalités liées au téléphone. Cela nous permet d'y accéder facilement, sans devoir faire appel a des plugins ou librairies extérieurs.

Son plus gros désavantage est selon moi ça limitation de rendu. On est obligé d'utiliser les layouts déjà existant et la personnalisation n'est pas évidente. Par exemple, je n'ai pas réussi à afficher sur la même page, pour les tablettes, la galerie de résultat et la map qui lui est lié. Je pense que c'est faisable. J'en suis même certain, car j'ai vu des applications natives le faire, mais je n'ai pas trouvé le moyen.

Nous avons pu remarquer de nombreuses différences lors de l'implémentation des fonctionnalités, la plus important au niveau Phonegap est un « bug » d'affichage : si on utilise un lien pour passer de la galerie à la carte, lors du retour sur l'ancienne page la carte est toujours affiché en arrière-plan. Sous une application Android native il n'y a pas cet inconvénient

Cependant, malgré ce soucis il est plus évident d'utiliser android lorsque l'on doit développer une application qui demande des fonctionnalités du téléphone.