

Partie II : Développement Mobile

TP2 : Conception d'une interface graphique d'une application

Objectif :

Une application Android est un assemblage de composants de différentes natures, possédant leurs propres entrées/sorties et leurs propres caractéristiques d'interactions. Cette autonomie des différents composants constituant une application rend possible des schémas de fonctionnement dans le cadre desquels ces composants peuvent se voir solliciter, non seulement par les autres composants de l'application, mais également par d'autres composants externes susceptibles de bénéficier des apports fonctionnels de certains composants de l'application.

Aujourd'hui, nous allons étudier les layouts et les vues.

La gestion des Ressources

Les ressources sont les fichiers XML faisant partie du dossier /src/main/res.

- **res/layout** : ce sont les définitions d'écrans des activités, par exemple `activity_main.xml`. Les balises XML et leurs attributs définissent les éléments affichés : boutons, zones de saisie, etc. Dans les classes Java, ce layout sera désigné par `R.layout.activity_main`.
- **res/values** : le fichier `strings.xml` contient les textes de votre application : labels, messages, etc. Ces textes sont identifiés par l'attribut `name` et référencés dans les layouts par `@string/nom`, et par `R.string.nom` dans les sources Java. Si vous voulez traduire ces textes en Arabe, faites une copie du dossier `values` en `values-ar` et traduisez tous les textes sans changer les attributs `name`.
- **res/mipmap-*dpi** et **res/drawable-*dpi** : vous y trouverez les icônes dans différentes tailles. Chaque image PNG devient une ressource identifiée par `R.drawable.son_nom` en Java et `@drawable/son_nom` dans un layout. C'est le système qui choisit automatiquement, selon la densité de votre écran, celle qu'il faut afficher.

Les fichiers XML peuvent être affichés soit sous forme d'un formulaire ou de manière graphique, soit sous forme XML brute.

Dans **layout/activity_main.xml**, par la pratique il faut placer tous les messages dans **res/values/strings.xml**, afin qu'ils puissent être facilement traduits dans d'autres langues puis référencez la chaîne en question dans le layout avec `android:text`.

Pour certains layouts, il faut identifier les vues. Ça consiste à rajouter un attribut `android:id="@+id/nom"` :

Les Composants d'interface :

Android contient plusieurs dizaines de composants d'interface :

- Des vues : TextView, Button, EditText. . .
- Des groupes : LinearLayout, RelativeLayout, TableLayout. . . lire [declaring-layout.html](#).

Les groupements de vues

Ces vues permettent d'arranger d'autres vues. Elles dérivent de la classe ViewGroup. Elles ont des vues associées et décident de leur placement, taille et position, en fonction de propriétés présentes sur ces vues. Voici les quatre à connaître :

- LinearLayout pour former des lignes ou des colonnes,
- TableLayout pour disposer en tableau,
- RelativeLayout pour mises en page statiques plus complexes,
- ConstraintLayout pour des dispositions dynamiques complexes.

Mise en page avec des LinearLayouts :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:text="Valider"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:text="Supprimer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Supprimer tout" />
</LinearLayout>
```

Figure 1 : Utilisation des **LinearLayouts**

La plupart des groupes utilisent des paramètres de taille et de placement sous forme d'attributs XML. Ces paramètres sont de deux sortes :

- Obligatoires pour toutes les vues :
 - android:layout_width : largeur de la vue
 - android:layout_height : hauteur de la vue

Ils peuvent valoir :

- "wrap_content" : la vue prend la place minimale
- "match_parent" : la vue occupe tout l'espace restant
- "valeurdp" : une taille fixe, ex : « 100dp » (dp : unité de taille indépendante de

l'écran)

- Sont demandés par le groupe englobant et qui en sont spécifiques, comme
 - `android:layout_weight` : spécifier un rapport de taille entre plusieurs vues.
 - `android:layout_alignParentBottom`,
 - `android:layout_centerInParent`.

Examinez la mise en page en comparant avec les propriétés des vues, puis faites les changements suivants :

- Mettez `match_parent` pour `android:layout_width` de l'un des boutons, puis annulez le changement.
- Mettez `match_parent` pour `android:layout_height` du deuxième bouton, regardez son effet, puis annulez.
- Mettez 200dp puis 300dp pour `android:layout_height` du deuxième bouton, regardez son effet, puis annulez.
- Changez `android:orientation` en `horizontal` pour le `LinearLayout`, laissez ce changement. Passez en mode paysage si les vues sont trop encombrantes.

On voit que les vues ont une largeur dépendant de leur contenu. On voudrait qu'elles aient la même largeur.

- Ajoutez `android:layout_weight="1"` pour chacune. Ce n'est pas très visible, mais la largeur n'est pas exactement la même.
- Changez `android:layout_width="0dp"` pour les trois vues. En principe, c'est le meilleur résultat. La largeur nulle combinée avec un poids non nul fait faire un calcul correct.
- Changez `android:layout_width="wrap_content"` pour le `LinearLayout` puis annulez ce changement.

Maintenant, vous devez implémenter l'interface suivante (créer une activité vide) :

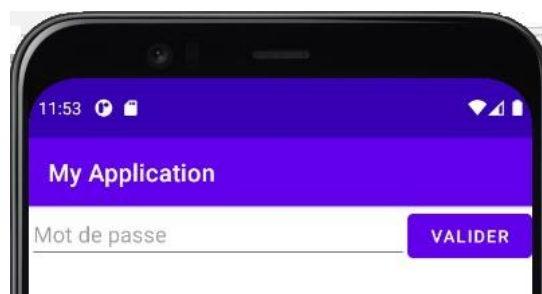


Figure 2 : Interface de saisie de mot de passe

- Ajoutez un `EditText` devant un bouton Valider.
- Remplacez la propriété `android:text` par `android:hint` et mettez la chaîne "mot de passe".
- Puis modifiez les largeurs pour obtenir le résultat demandé : l'`EditText` prend presque toute la place en largeur, tandis que le bouton Valider est le plus petit possible. Pour cela, il faut mettre un poids nul au bouton OK mais aussi lui donner une taille minimale.

Dans un cas général, on est amenés à combiner des `LinearLayouts` horizontaux et verticaux.

Implémenter l'interface suivante :

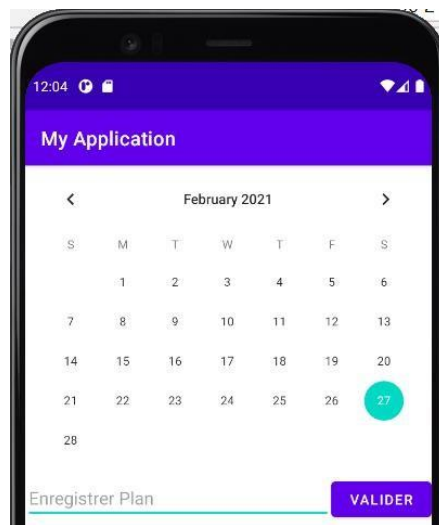


Figure 3 : Interface combinant deux types de LinearLayouts

Mise en page avec des TableLayouts :

TableLayouts sert à organiser les éléments comme dans un tableau HTML. Ce sont les mêmes notions codées en XML.



Figure 4: Utilisation des **TableLayouts**

Par défaut, dans une colonne toutes les cases font la même largeur, et même si une colonne est plus étroite, elle reste quand même assez large. Deux propriétés peuvent être configurées :

- Ajoutez `android:shrinkColumns="0"` dans la balise `<TableLayout>` en haut. La première colonne 0 est autorisée à devenir plus étroite si la place manque, ou carrément disparaître. Pour voir cela, allongez le texte de l'un des boutons 2e colonne.
- Ajoutez `android:stretchColumns="1"` dans la balise `<TableLayout>`. La 2e colonne va s'élargir autant que possible – allongez le texte d'un bouton. Plusieurs colonnes peuvent être étirées : `android:stretchColumns="1,2"`.

Mise en page avec un RelativeLayout :

RelativeLayout permet une mise en page précise mais un petit peu complexe : Le principe est de positionner une vue fils par rapport soit aux bords du RelativeLayout, soit à des vues précédemment posées.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/HAUT"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:backgroundTint="#ff683e"
        android:text="HAUT" />

    <Button
        android:id="@+id/DROIT"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_toRightOf="@id/HAUT"
        android:backgroundTint="#8cff5d"
        android:text="DROIT" />

    <Button
        android:id="@+id/button3" android:text="BAS"
        android:backgroundTint="#19C7C1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/DROIT"
        android:layout_alignLeft="@id/HAUT"
        android:layout_alignRight="@id/HAUT"/>

</RelativeLayout>
```

Figure 5 : Utilisation des **RelativeLayouts**

Dans un RelativeLayout les vues fils sont positionnées et dimensionnées par quatre contraintes, une par côté, haut, bas, gauche, droite. Les côtés droit et gauche sont aussi désignés par start et end pour pouvoir tenir compte des régionalisations (écriture de droite à gauche, etc.) : Il se peut qu'il y a avertissements vu qu'on doit rajouter des propriétés comme toEndOf quand il y a toRightOf.

Il y a deux catégories de contraintes :

- Positionnement par rapport au layout : La valeur de ces contraintes est de simples booléens et on ne les mentionne que quand ils sont vrais.
 - Collé au bord du parent : layout_alignParentXXX avec XXX valant Top, Bottom, Left et Right, ainsi que les variantes Start et End.
 - Centré dans le parent : layout_centerInParent, layout_centerHorizontal et layout_centerVertical.

- Positionnement par rapport à une autre vue : La valeur à mettre dans ces contraintes est la référence à l'identifiant d'une autre vue @id/autre_vue. Chaque vue concernée doit donc définir un identifiant.
 - Collé au bord opposé : layout_toRightOf, layout_toLeftOf, layout_above et layout_below. Par exemple, il faut coller le bord droit de cette vue au bord gauche de l'autre vue.
 - Bords alignés : layout_alignXXX comme pour alignParent. Cette vue aligne son bord avec celui de l'autre vue.

Pour voir l'effet de ces contraintes, essayez ces changements et comprenez-les :

- Dans le bouton HAUT, changez la contrainte alignParentTop en alignParentBottom.
- Dans le bouton HAUT, allongez le titre android:text et constatez les répercussions sur le bouton BAS.
- Dans le bouton BAS, allongez le titre, et constatez ce qui se passe.
- Dans le bouton BAS, changez la valeur alignRight avec le bouton DROIT au lieu de bouton HAUT.

Maintenant, Essayez d'implémenter cette page. Le bouton BUTTON1 est mis au centre de l'écran. Les boutons sont définis dans l'ordre de leur nom.

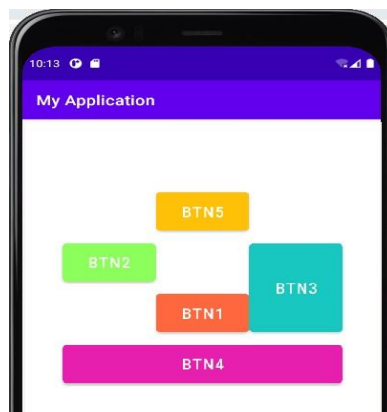


Figure 6 : Interface à Implémenter

Mise en page avec un ConstraintLayout :

Ce layout fait partie de la bibliothèque androidx, celle qui permet de créer des applications fonctionnant sur de vieux smartphones. Cela impose une déclaration dans le fichier app/build.gradle de l'application :

```
dependencies {
    ....
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    ...
}
```

Figure 7 : Ajout des dépendances

ConstraintLayout est une extension de RelativeLayout, permettant de mettre à jour la disposition

de manière plus dynamique et plus polyvalente. Les attributs pour mettre en place les contraintes ont des noms plus homogènes.

Forme générale des contraintes : `layout_constraintXXX_toYYYOf` avec XXX et YYY valant Top, Bottom, Left, Right ou Baseline.

Cela donne des attributs comme `layout_constraintTop_toTopOf`, `layout_constraintRight_toLeftOf`, etc. Le code Baseline fait aligner sur la base du texte des vues.

Tous ces attributs prennent pour valeur une référence d'identifiant (et non pas un booléen comme certains avec les `RelativeLayout`).

Par exemple, on écrit `layout_alignParentTop="true"` dans un `RelativeLayout`, mais `layout_constraintTop_toTopOf="@id/id_du_layout"` dans un `ConstraintLayout`.

Pour positionner une vue à une certaine distance d'une autre, on utilise les attributs `layout_marginXXX`. Deux autres contraintes existent : `layout_constraintZZZ_bias` avec ZZZ étant Vertical ou Horizontal. Elles permettent de positionner une vue de manière proportionnellement à l'espace disponible : elles déséquilibrent une contrainte en faveur de l'un ou l'autre côté. La valeur à mettre est un réel entre 0 et 1, par défaut, c'est 0.5.

La taille d'une vue peut être minimale `wrap_content`, maximale `match_parent` ou donnée par les contraintes de placement `0dp` (aussi nommée `match_constraint`)

Essayer de mettre en place ce layout :

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/haut"
        android:text="haut"
        android:backgroundTint="#ff683e"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="32dp"
        android:layout_width="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintHorizontal_bias="0.25"/>
    <Button
        android:id="@+id/bas"
        android:text="bas"
        android:backgroundTint="#19C7C1"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@id/haut"
        android:layout_width="0dp"
        app:layout_constraintLeft_toLeftOf="@id/haut"
        app:layout_constraintRight_toRightOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Figure 8 : Utilisation des `ConstraintLayouts`

Le bouton haut est positionné à 32 pixels du haut et à 25% à gauche dans la largeur de son parent.

Le bouton bas est mis sous le premier, et comme sa largeur est nulle, elle est forcée à ce que donnent ses contraintes : du bord gauche du bouton haut au bord droit du parent.

Essayez de reproduire cette disposition avec un ConstraintLayout :



Figure 9 : Interface à Implémenter

Le bouton 1 est centré en haut. Le bouton 2 est placé en bas et à gauche du n°1. Le bouton 3 est sous le n°2 et va horizontalement du bord gauche de la fenêtre au bord droit du bouton 1. Le bouton 4 remplit l'espace à droite entre le haut de la fenêtre et le bas du bouton 3.

Exercices :

Exercice 1 :

Créez une interface pour une application toute simple qui permet d'enregistrer des achats de vêtements au cours d'un solde. Il faut au moins pouvoir saisir la quantité et le prix. Rajoutez aussi de quoi saisir l'ancien prix avant le solde. Il serait bien qu'il y ait la possibilité d'indiquer le type d'article d'habillement (pantalon, tricot, T-shirt, ..). Il faudrait également pouvoir saisir la date et le lieu de l'achat.

Il faudrait voir du côté des attributs `inputType` des `EditText` pour restreindre la saisie des valeurs autorisées, nombres décimaux par exemple.

En ce qui concerne le sélecteur de type d'habillement, utiliser un `Spinner` :

`Spinner` est un `ViewGroup` permettant à l'utilisateur de sélectionner une valeur dans une liste de valeurs. Par défaut, `Android Spinner` fonctionne comme un `Dropdown List` (Menu déroulant) ou `Combox` dans les autres langages de programmation.

Après placez les différents types dans une ressource de type `string-array` et indiquez leur identifiants dans l'attribut `android:entries` du `Spinner`. Voici un extrait :



Figure 10 : Extrait d'utilisation d'un Spinner

Exercice 2 :

Créer un écran de saisie pour un smartphone : nom du smartphone, fabricant, année de sortie, le processeur, la ram, caméra frontale, caméra arrière et notation.

La notation, c'est à dire la note de 0 à 5 que vous attribuez au smartphone sera réalisée par un RatingBar : Cette vue est faite pour attribuer une note sous forme d'étoiles :

- Imposer une largeur autre que "wrap_content" sinon le nombre d'étoiles dépendra de la largeur effective,
- Pour fixer le nombre d'étoiles, affectez sa propriété android:numStars, mettez-en 5,
- Vous pouvez aussi fixer le pas, c'est à dire la granularité des notes, affectez par exemple android:stepSize="0.5" qui signifie qu'on peut mettre des notes de 0.5 en 0.5.

Configurer aussi l'attribut inputType pour que l'année de sortie, la ram, le taille et la résolution de la caméra n'accepte que des chiffres.