LAB 3 DAA

Auteurs

- Duruz Florian
- Ferreira Silva Sven
- Richard Aurélien

Groupe : DAA_B_12

Classe : DAA 2025-26

Date : 02/11/2025

Implémentation

Layout

Les composants visuels ajoutés sont tous dans un seul `ConstraintLayout` contenu dans une `ScrollView` (pour permettre d'afficher tout le contenu en scrollant si l'écran est trop petit par exemple).

Group

Pour gérer l'affichage/apparition des champs pour `worker` ou `student` nous avons ajoutés un Contstraint Group pour chacun.

Ex :

```
<androidx.constraintlayout.widget.Group
                android:id="@+id/group_student"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                app:constraint_referenced_ids="text_student_title,
                    text_view_student_school,edit_text_student_school,
                    text_view_student_year,edit_text_student_year"
                android:visibility="gone" />
```

Cela va permettre de regouper tous les ids lié à une occupation et donc d'avoir la possibité de tous les affecter avec une seule ligne de code (au lieu d'une ligne par champs à cacher/montrer).

```
binding.groupStudent.visibility = Group.VISIBLE
binding.groupWorker.visibility = Group.GONE
```

Barrier

Dans un `ConstraintLayout`, les vues sont positionnées relativement à d'autres vues et nous voulons que la TextView "DONNÉES COMPLÉMENTAIRES" apparaisse en dessous du dernier champ de worker ou student.

Malheureusement une contrainte standard ne peut référencer qu'une seule vue. Pour gérer plusieurs vues de référence, nous introduisons une `Barrier`.

Une Barrier est une vue invisible qui s'adapte automatiquement à la position de ses vues référencées et permet d'aligner d'autres vues par rapport à elle. Ici, nous définissons une Barrier qui se positionne sous le dernier champ visible parmi `edit_text_student_year` et `edit_text_worker_experience`. Ensuite, la TextView "DONNÉES COMPLÉMENTAIRES" est attachée à cette Barrier, garantissant qu'elle s'affiche toujours après le dernier champ actif.

```
<androidx.constraintlayout.widget.Barrier
    android:id="@+id/barrier_after_worker_student"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:barrierDirection="bottom"

app:constraint_referenced_ids="edit_text_student_year,edit_text_worker_experience"
/>
```

Guideline

Afin d'avoir une UI consistante nous avons introduit une `Guideline` qui va permettre d'aligner les chmpas sur cette dernière.

```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_percent="0.4" />
```

Cette guideline va être une ligne vericale invisible à 40% de la largeur. Elle est ensuite utilisée par les champs pour s'aligner dessus:

```
<TextView
    android:id="@+id/text_view_name"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="@string/main_base_name_title"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toStartOf="@id/guideline" //On y fait reference ici
    app:layout_constraintTop_toBottomOf="@id/text_view_base_data"
    app:layout_constraintBottom_toBottomOf="@id/edit_text_name" />
```

Controller

Initialisation de la vue avec une Person

Afin d'initialiser la vu avec une personne déjà existante la fonction `populateView()` fut créé. Elle prend en paramêtre une `Person` et va mettre à jour la valeur des champs avec la valeur de la personne.

Il y'a aussi un switch en fonction du type de `Person` (`Worker` ou `Student` par exemple) pour populer certains champs !

Création de Student ou Worker

Pour créer un Student ou Worker depuis le formulaire nous avons un bouton qui va appeler `validateForm()`.

La fonction va valider les champs "importants" et retourner une erreur si tel n'est pas le cas. Si des champs sont manquants ou faux alors un message d'erreur va informer l'utilisateur.

Si tout passe alors nous instancions un nouveau Student ou Worker en fonction de l'occupation choisie.

Suppression du contenu du formulaire

Afin de supprimer tout le contenu présent dans le formulaire le bouton `Cancel` va `clear` tous les champs text et remettre à 0 les spinner.

Réponse aux questions

4.1

Pour le champ remark, destiné à accueillir un texte pouvant être plus long qu'une seule ligne, quelle configuration particulière faut-il faire
dans le fichier XML pour que son comportement soit correct ? Nous pensons notamment à la possibilité de faire des retours à la ligne,
d'activer le correcteur orthographique et de permettre au champ de prendre la taille nécessaire.

- Pour la saisie suer plusieur ligne, il faut activer "textMultiLine" dans le champs InputType.
- Pour l'auto-correcteur, il faut activer "textAutoCorrect" dans le champs InputType.
- Pour que L'EditText ajuste sa taille, il faut mettre la valeur "height" à "wrap-content" et ne pas mettre de valeur à "maxLines"

```
<EditText
android:id="@+id/edit_text_comments"
android:layout_width="0dp"
android:layout_marginTop="8dp"
android:layout_height="wrap_content"
android:inputType="textMultiLine|textAutoCorrect"
android:minLines="3"
android:gravity="top|start"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintTop_toBottomOf="@id/text_view_comments"/>
```

4.2

Pour afficher la date sélectionnée via le DatePicker nous pouvons utiliser un DateFormat permettant par exemple d'afficher 12 juin 1996
à partir d'une instance de Date. Le formatage des dates peut être relativement différent en fonction des langues, la traduction des mois par exemple,
mais également des habitudes régionales différentes : la même date en anglais britannique serait 12th June 1996 et en anglais américain June 12, 1996.
Comment peut-on gérer cela au mieux ?

Pour gérer l'affichage des dates selon la langue et la région on utilise la classe "DateFormat" fournie par Android
qui va ajuster le format de la date aux conventions locales.

```
private val dateFormat = SimpleDateFormat("MMM d, yyyy", Locale.getDefault())

val cal = Calendar.getInstance()
val dateSetListener = DatePickerDialog.OnDateSetListener { _, year, month,
dayOfMonth ->
    cal.set(Calendar.YEAR, year)
    cal.set(Calendar.MONTH, month)
    cal.set(Calendar.DAY_OF_MONTH, dayOfMonth)
    editBirthdate.setText(dateFormat.format(cal.time))
}
```

4.3

Si vous avez utilisé le DatePickerDialog du SDK. En cas de rotation de l'écran du smartphone lorsque le dialogue est ouvert,
une exception android.view.WindowLeaked sera présente dans les logs,
à quoi est-elle due et comment pouvons-nous la corriger ?

L'exception est provoquée par la gestion du dialogue depuis l'activité. Le lifecycle du dialogue dépend directement de celui de l'activité l'ayant instanciée.
Si l'activité est interrompue/détruite alors que le dialogue est encore en cours d'affichage, il y a un soucis de consistence avec un dialogue qui "existe toujours" mais son activité mère n'existe plus.

Pour fixer le bug, une approche possible est la suivante:

- enregistrer l'état du Dialog dans un attribut privé lateinit,
- lors de l'appel à onPause ou onDestroy de l'activité hôte, on utilise cette variable afin de dismiss le dialogue quand nécessaire.

```
// private attribute
private lateinit var datePickerDialog : DatePickerDialog
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    // Varied onCreate code...

    // --- DatePicker ---
    val cal = Calendar.getInstance()
    val dateSetListener = DatePickerDialog.OnDateSetListener { _, year, month,
dayOfMonth ->
        cal.set(Calendar.YEAR, year)
        cal.set(Calendar.MONTH, month)
        cal.set(Calendar.DAY_OF_MONTH, dayOfMonth)
        editBirthdate.setText(dateFormat.format(cal.time))
    }

    datePickerDialog = DatePickerDialog(
        this,
        dateSetListener,
        cal.get(Calendar.YEAR),
        cal.get(Calendar.MONTH),
        cal.get(Calendar.DAY_OF_MONTH)
    )

    fun showDatePicker() {
        datePickerDialog.show()
    }

    // Continue your onCreate...
}

override fun onDestroy() {
    super.onDestroy()
    if (datePickerDialog.isShowing) {
        datePickerDialog.dismiss()
    }
}

override fun onPause() {
    super.onPause()
    if (datePickerDialog.isShowing) {
        datePickerDialog.dismiss()
    }
}
```

4.4

Lors du remplissage des champs textuels, vous pouvez constater que le bouton « suivant » présent sur le clavier virtuel permet de sauter automatiquement
au prochain champ à saisir, cf. Fig. 2.
Est-ce possible de spécifier son propre ordre de remplissage du questionnaire ?
Arrivé sur le dernier champ, est-il possible de faire en sorte que ce bouton soit lié au bouton de validation du questionnaire ?

Nous pouvons définir un ordre custom pour le remplissage du form.
Dans le fichier layout, il existe l'attribut `android:nextFocusDown="<nextEditFormId>"` pour les EditText qui permet de définir le prochain formulaire à sélectionner lors du click sur la flèche "suivant" du clavier.

L'exemple suivant permet de tourner en boucle entre "name" et "first name" du form avec le button next du clavier:

```
<!-- Name -->
<TextView
    android:id="@+id/text_view_name"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="@string/main_base_name_title"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toStartOf="@id/guideline"
    app:layout_constraintTop_toBottomOf="@id/text_view_base_data"
    app:layout_constraintBottom_toBottomOf="@id/edit_text_name" />

<EditText
    android:id="@+id/edit_text_name"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:hint="@string/main_base_name_title"
    android:autofillHints="@string/main_base_name_title"
    android:inputType="textPersonName"
    app:layout_constraintStart_toStartOf="@id/guideline"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id/text_view_base_data" />

<!-- Surname -->
<TextView
    android:id="@+id/text_view_surname"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="@string/main_base_firstname_title"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toStartOf="@id/guideline"
    app:layout_constraintTop_toBottomOf="@id/edit_text_name"
    app:layout_constraintBottom_toBottomOf="@id/edit_text_surname" />

<EditText
    android:id="@+id/edit_text_surname"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:hint="@string/main_base_firstname_title"
    android:autofillHints="@string/main_base_firstname_title"
    android:inputType="textPersonName"
    app:layout_constraintStart_toStartOf="@id/guideline"
    app:layout_constraintEnd_toEndOf="parent"
```

```
    android:nextFocusDown="@id/edit_text_name"

    app:layout_constraintTop_toBottomOf="@id/edit_text_name" />
```

Concernant le dernier champ, il est important de distinguer deux choses:

1. si nous considérons le dernier champ comme étant le champ "Commentaires", il n'est pas possible avec les outils intégrés, sauf si nous implémentons le champ "Commentaires" comme étant un single-line, auquel cas le clavier ne proposera jamais de retour à la ligne et offrira uniquement l'option pour passer au champ suivant;
2. si nous considérons le dernier champ comme étant l'adresse e-mail, il est possible de le faire:
   a. définissons un ImeOptions dans la view du e-mail TextEdit, qui nous servira de code de détection afin de trigger le button OK,
   b. ensuite, définissions dans le code de l'activité un custom editorAction listener qui récupérera ce code et fera un call au button OK (fait ainsi afin de limiter la répétition de code). En pressant sur "suivant" du clavier au niveau du form E-mail, le submit button sera appelé.

```
<EditText
    android:id="@+id/edit_text_address"
    android:layout_width="0dp"
    android:layout_marginTop="8dp"
    android:layout_height="wrap_content"
    android:hint="@string/additional_email_title"
    android:inputType="textEmailAddress"

    android:imeOptions="actionGo"

    app:layout_constraintStart_toStartOf="@id/guideline"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id/text_complementary_data_title" />
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    // Varied onCreate code...

    // Important: the OK button must be defined, and its action already defined
before this strip of code
    findViewById<EditText>(R.id.edit_text_address).setOnEditorActionListener { v,
actionId, event ->
        if (actionId == EditorInfo.IME_ACTION_GO) {
            buttonOk.performClick()
            true
        } else {
            false
        }
    }
}
```

4.5

Pour les deux Spinners (nationalité et secteur d'activité),
comment peut-on faire en sorte que le premier choix corresponde au choix null, affichant par exemple le label « Sélectionner » ?
Comment peut-on gérer cette valeur pour ne pas qu'elle soit confondue avec une réponse ?

Le Spinner n'est malheureusement pas programmé pour offrir une valeur par défaut non-sélectionnable, ou un placeholder text.
Il faut donc être créatif. On pourrait créer des listeners custom afin de pouvoir gérer certains items (qu'on utiliserait comme placeholders) pour qu'ils ne soient pas sélectables ou visibles,
ou encore créer un Spinner maison héritant du Spinner avec des overrides de diverses méthodes pour effectuer le même résultat (ou encore, utiliser un autre widget pour lister et choisir, un qui supporte la fonctionnalité du placeholder).

Pour cette implémentation, nous avons fait simple: ajouter dans strings.xml la valeur par défaut, et effectuer des comparaisons lors de la validation du form pour vérifier que les placeholders n'aient pas été choisis.

Note: les exemples de code ci-dessous ne concernent que Nationality spinner, mais l'implémentation est identique concernant le Sector spinner.

```
    <string-array translatable="false" name="nationalities">
        <item>@string/nationality_empty</item>
        <item>@string/ch</item>
        <item>@string/fr</item>
        <item>@string/de</item>
        <item>@string/it</item>
    </string-array>

    <string name="nationality_empty">Sélectionner</string>
    <string name="ch">Suisse</string>
    <string name="fr">Française</string>
    <string name="de">Allemande</string>
    <string name="it">Italienne</string>
```

```
buttonOk.setOnClickListener {
    val nationality = spinnerNationality.selectedItem.toString()
    // Other declarations...

    if (nationality == getString(R.string.nationality_empty)) {
        Toast.makeText(this, "Veuillez choisir une nationalité",
Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }

    // Rest of the form validation...
}
```