DAA - Laboratoire 3

November 04, 2024

Émilie Bressoud **Sacha Butty** **Loïc Herman**

Réponses aux questions

Question 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.*

Il faut ajouter les attributs suivants au champ `EditText` dans le fichier XML :

```
<!-- Fait en sorte que le texte soit aligné du haut du champ -->
android:gravity="top"
<!-- Indique à Android que le champ est multi-lignes et peut être auto-corrigé -->
android:inputType="textMultiLine|textAutoCorrect"
<!-- Affiche 3 lignes d'espace vide et grandira en fonction du contenu -->
android:minLines="3"
```

Cela permet de définir le champ comme étant un champ multiligne et de définir le nombre de lignes affichées par défaut. En ce qui concerne le correcteur orthographique, nous avons ajouté l'attribut `android:inputType="textAutoCorrect"`.

Question 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 les différentes langues et les différentes habitudes régionales, nous initialisons l'instance de `DateFormat` du companion object de `Person` qui utilise la locale par défaut du système (via l'appel à `getDateInstance()`). Nous avons également défini la timezone du formater à `UTC` pour éviter les problèmes de décalage lors de la conversion en epoch UTC pour le date picker.

```
companion object {
  private val utcTimezone: TimeZone = TimeZone.getTimeZone("UTC")
  private val utcDateFormatter = Person.dateFormatter.apply { timeZone = utcTimezone }
}
```

Ainsi, la langue de la date affichée dépendra de la langue configurée sur le téléphone et suivra les habitudes régionales.

Question 4.3

- *Si vous avez utilisé le MaterialDatePicker2 de la librairie Material. Est-il possible de limiter les dates sélectionnables dans le dialogue, en particulier pour une date de naissance il est peu probable d'avoir une personne née il y a plus de 110 ans ou à une date dans le futur. Comment pouvons-nous mettre cela en place ?*

Pour limiter les dates sélectionnables dans `MaterialDatePicker`, nous pouvons utiliser la méthode `setCalendarConstraints` qui permet de définir les limites de dates sélectionnables. Nous ajoutons dans ces contraintes un validateur qui validera que la date soit dans le passé, ainsi que les dates de début et de fin pour limiter la sélection à 110 ans en arrière jusqu'à aujourd'hui.

Nous avons également défini la sélection à la date actuelle si une date est déjà définie ou le jour courant le cas échéant.

```
val datePicker = MaterialDatePicker.Builder.datePicker()
  .setCalendarConstraints(
    CalendarConstraints.Builder()
      // Make the date picker start at the current date
      .setOpenAt(currentTimestamp)
      // Validate the date is in the past
      .setValidator(DateValidatorPointBackward.now())
      // Restrict the date picker from 110 years ago to today
      .setStart(utcCalendar.also { it.add(Calendar.YEAR, -110) }.timeInMillis)
      .setEnd(MaterialDatePicker.todayInUtcMilliseconds())
      .build()
  )
  // Select the current date if the input field is not empty
  .setSelection(currentTimestamp)
  .build()
```

Question 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 ?*

Oui, l'ordre peut être configuré en utilisant l'attribut `android:imeOptions` pour donner l'indication au clavier virtuel pour le type d'action qui doit être effectué avec le bouton « suivant ». Nous l'avons par exemple utilisé pour le champ du prénom pour que le focus soit transféré au bouton d'ouverture du `DatePicker` après avoir appuyé sur « suivant », avec un `focusChangeListener` sur le bouton permettant l'ouverture automatique du sélecteur.

Pour spécifier un ordre de remplissage personnalisé, il est aussi possible d'utiliser l'attribut `android:nextFocusDown` pour indiquer le champ suivant à sélectionner lorsque le focus se déplace dans la direction `View.FOCUS_DOWN` (qui est la direction par défaut). L'attribut doit faire référence à l'ID du champ suivant. Il existe des variantes de cet attribut pour toutes les directions de focus possibles.

Pour que le champ des commentaires affiche le bouton « suivant » comme bouton de fin, nous avons spécifié l'attribut `android:imeOptions="actionDone"` pour indiquer que le bouton « suivant » doit être remplacé par un bouton « done » qui permet de valider le formulaire. Avec ceci nous avons ajouté un `OnEditorActionListener` sur le champ pour intercepter l'action « done » et automatiquement cliquer sur le bouton de validation du formulaire.

```
binding.inputComments.setOnEditorActionListener { _, action, _ ->
  if (action == EditorInfo.IME_ACTION_DONE) {
    binding.buttonOk.performClick()
    true
  } else {
    false
  }
}
```

Question 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 ?*

Pour afficher un choix par défaut dans un `Spinner`, nous avons créé un `Adapter` spécifique pour le `Spinner` qui permet de gérer l'affichage du choix par défaut. Le constructeur de ce dernier prend en paramètre le texte de la valeur par défaut et le tableau des valeurs possibles.

```
binding.inputNationality.adapter = SpinnerDefaultValueAdapter(
  this,
  resources.getString(R.string.nationality_empty),
  resources.getStringArray(R.array.nationalities)
)
```

Cet adapter va ajouter la valeur par défaut en première position de la liste des valeurs possibles. La méthode `getDropDownView` est surchargée pour désactiver l'affichage de la première valeur dans le dropdown.

Nous surchargeons également la méthode `isEnabled(int position)` pour empêcher la sélection de la première valeur dans le `Spinner`.

```
override fun getDropDownView(position: Int, convertView: View?, parent: ViewGroup): View {
  // Hides the first item (default value) in the dropdown list
  if (position == 0) {
    val v = View(context)
    v.visibility = View.GONE
    return v
  }

  return super.getDropDownView(position, null, parent)
}

override fun isEnabled(position: Int): Boolean {
  // Sets the first item (default value) as non-selectable
  return position != 0 && super.isEnabled(position)
}
```

Ces deux surcharges feront que le choix n'est pas visible est non-sélectionnable dans le `Spinner`, permettant d'éviter toute confusion tout en gardant l'affichage de la valeur par défaut quand le champ est dans son état initial (position 0).

Description de l'implémentation

Nous avons défini notre layout en utilisant qu'un seul `ConstraintLayout` imbriqué dans une `ScrollView` nécessaire pour le cas où le formulaire dépasse la hauteur de l'écran.

En termes de contrôleur, notre système utilise la génération de binding automatique pour simplifier la gestion de la récupération des éléments et de pouvoir définir les interactions avec.

Layout

Nous avons implémenté des widgets Barrier qui permettent de basculer dynamiquement entre l'affichage des champs pour un étudiant et ceux pour un employé. Grâce à cette approche, le widget suivant se réfère directement au Barrier plutôt qu'aux labels spécifiques comme `labelExperience` ou `labelGradYear`.

```
<androidx.constraintlayout.widget.Barrier
  android:id="@+id/additionalDataStartBarrier"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  app:barrierDirection="bottom"
  app:barrierMargin="@dimen/element_margin"
  app:constraint_referenced_ids="
    labelExperience,
    labelGradYear
  " />
```

Pour gérer plus efficacement l'affichage des champs spécifiques aux étudiants et aux employés, nous avons créé des groupes de widgets. Cette organisation nous permet de contrôler la visibilité de l'ensemble des éléments associés en modifiant simplement la propriété de visibilité du groupe parent, qui s'applique alors automatiquement à tous les widgets référencés dans ce groupe.

```
<androidx.constraintlayout.widget.Group
  android:id="@+id/additionalDataGroup"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  app:constraint_referenced_ids="
    additionalDataStartBarrier,
    titleAdditionalData,
    labelEmail,
    inputEmail,
    labelComments,
    inputComments
  "
  tools:visibility="visible" />
```

Pour maintenir une cohérence visuelle dans l'application, nous avons défini des dimensions de référence (ressources dimen) qui standardisent les marges et les tailles des éléments. Ces valeurs sont ensuite réutilisées à travers l'ensemble de l'interface, permettant d'assurer une uniformité visuelle et de faciliter les modifications globales.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="title_size">16sp</dimen>
    <dimen name="element_margin">8dp</dimen>
</resources>
```

Interactions controller-view

Remplissage de la date d'anniversaire

Pour la sélection de la date d'anniversaire, nous avons choisi d'utiliser le composant de sélection de la librairie material avec le `MaterialDatePicker`. Nous avons fait le choix d'initialiser ce dernier à chaque ouverture afin de pouvoir sélectionner la date actuelle et la remplir avec la valeur du champ si elle est déjà présente.

Quand le sélecteur de date retourne une valeur, le champ de texte non-modifiable par l'utilisateur est mis à jour avec une version formatée de la date selon la locale par défaut du système.

Choix du type de personne

Dans la méthode `onCreate`, un change listener est configuré pour le radio group de l'occupation et va afficher le groupe de champs spécifiques à l'occupation.

Afin de pouvoir remplir le formulaire avec les données par défaut, nous avons ajouté un menu déroulant, qui permet de remplir les valeurs par défaut pour un étudiant ou celle d'un employé.

Valeurs par défaut pour les spinners

Comme décrit dans la réponse à la question correspondante, un adapter spécialisé est utilisé pour définir l'entrée en position 0 comme étant l'entrée « placeholder ». Les spinners de nationalité et de secteur sont donc modifiés dans la méthode `onCreate` pour y définir l'adapter.

Options menu pour le pré-remplissage

Afin de simplifier la gestion du remplissage du formulaire avec les données d'exemple, nous avons ajouté un contexte menu à l'activité via la définition `res/menu/prefill_menu.xml` qui est rajouté à l'activité via la méthode surchargée `onCreateOptionsMenu()`. Ensuite, quand l'utilisateur fait une sélection, nous lions les options avec notre méthode `restoreData` en fonction du choix effectué.

Validation automatique après le remplissage du champ commentaires

Comme décrit plus haut dans les questions, nous avons configuré le champ `EditText` des commentaires avec l'attribut `android:imeOptions="actionDone"` et un listener `OnEditorActionListener` qui va automatiquement cliquer sur le bouton « OK » si la touche « done » du clavier virtuel est pressée.

Validation des données

La validation est effectuée par le contrôleur et valide que les champs contiennent tous une valeur ou une sélection pour les spinners et radio group.

Si la validation retourne `true`, alors une instance du modèle est retournée et ajoutée dans les logs.

Nous avons également ajouté des toasts pour indiquer l'erreur de validation ou un message de confirmation en cas de succès.