DAA-L3-MVC

Auteurs : Bugna, Slimani & Steiner

Explication de l'implémentation de la solution

Nous examinons ici les fichiers `MainActivity.kt` et `activity_main.xml`. Ces deux fichiers travaillent ensemble pour fournir l'interface utilisateur et gérer les interactions dans une activité Android.

Analyse et explication du fichier `MainActivity.kt`

Le fichier `MainActivity.kt` gère la logique de l'activité, y compris la gestion des interactions utilisateur et du sélecteur de date.

Principaux éléments de `MainActivity.kt` :

1. **Initialisation de l'interface** :
   - `setContentView(R.layout.activity_main)` : Cette ligne, dans `onCreate`, charge le fichier `activity_main.xml` pour afficher l'interface utilisateur. Ce fichier XML définit les éléments de l'interface comme les champs de texte et boutons.

2. **Fonction `showDatePicker()`** :
   - La fonction `showDatePicker()` affiche un `DatePickerDialog` pour que l'utilisateur puisse sélectionner une date.
   - Une instance `Calendar` permet de gérer la date sélectionnée. Si une date existe dans le champ de texte `etDateNaissance`, elle est récupérée et affichée dans le sélecteur. Sinon, la date du jour est utilisée.
   - **Affichage du `DatePickerDialog`** :
     - Le `DatePickerDialog` est initialisé avec `this` comme contexte, ce qui signifie que l'activité gère elle-même le dialogue.
     - Lorsque l'utilisateur sélectionne une date, elle est formatée pour apparaître dans `etDateNaissance`.

3. **Gestion de l'orientation et des fuites de mémoire** :
   - Le `DatePickerDialog.show()` est appelé sans enregistrer de référence persistante, ce qui réduit les risques de fuite de mémoire. Si l'orientation change, Android peut normalement gérer le dialogue sans générer d'erreur `WindowLeaked`.

Analyse et explication du fichier `activity_main.xml`

Le fichier `activity_main.xml` définit l'interface de l'activité, avec plusieurs sections pour organiser les données d'utilisateur.

Principales sections du fichier `activity_main.xml` :

1. **Section Données de Base** (`layoutBasicInfo`) :
   - Cette section regroupe les informations principales de l'utilisateur (nom, prénom, date de naissance, nationalité, occupation).
   - Chaque champ est disposé dans un `LinearLayout` et utilise un `TextView` pour le label et un `EditText` pour la saisie.

- Pour la **date de naissance**, `etDateNaissance` est accompagné d'un bouton `btnDatePicker` avec une icône pour lancer `showDatePicker()` (un clique dans le champs text ouvre aussi le selecteur de date).

2. **Données spécifiques aux employés** (`layoutWorkerInfo`) :
   - Cette section, affichée uniquement pour les employés, contient des champs pour l'entreprise `etEntreprise`, le secteur `spinnerSecteur`, et les années d'expérience `etExperience`.
   - Initialement masquée `visibility="gone"`, elle est affichée dynamiquement dans `MainActivity.kt` lorsque l'utilisateur sélectionne « Employé » dans le `RadioGroup rgOccupation` comme occupation.
3. **Données spécifiques aux étudiants** (`layoutStudentInfo`) :
   - Pour les étudiants, cette section inclut des champs pour l'école `etEcole` et l'année de diplôme `etAnneeDiplome`.
   - Cette section est également masquée au départ et n'apparaît que si l'utilisateur sélectionne « Étudiant » dans le `RadioGroup rgOccupation` comme occupation.
4. **Données complémentaires** (`layoutComplementaryInfo`) :
   - Cette section inclut des informations additionnelles comme l'email `etEmail` et des remarques `etCommentaires`.
   - Elle est positionnée sous les sections spécifiques aux employés et aux étudiants, grâce à un `Barrier dynamicContentBarrier` qui s'ajuste en fonction de la section affichée.
5. **Boutons de contrôle** :
   - En bas de l'écran, deux boutons sont proposés : « Annuler » `btnAnnuler` et « OK » `btnOk` pour confirmer ou annuler la saisie.
6. **Organisation des sections avec `ConstraintLayout` et `Barriers`** :
   - Les sections d'information (employé, étudiant, complémentaire) sont organisées en utilisant des `Barriers` qui permettent d'ajuster leur disposition dynamiquement. Par exemple, `data_barrier` et `dynamicContentBarrier` agissent comme des repères pour le positionnement.

Résumé de l'interconnexion entre `MainActivity.kt` et `activity_main.xml`

- `MainActivity.kt` gère la logique de l'interface utilisateur et affiche les dialogues de sélection. Il masque et affiche également les sections dynamiques selon le choix de l'utilisateur (étudiant ou employé).
- `activity_main.xml` fournit l'organisation visuelle avec les différentes sections et définit les interactions de base pour chaque champ, en utilisant `ConstraintLayout` pour une disposition adaptable.

Conclusion de l'analyse des choix d'implémentation

Cette implémentation sépare bien la logique et l'interface, permettant une gestion efficace des interactions utilisateur et des données, tout en évitant les erreurs comme `WindowLeaked` grâce à la structure de `showDatePicker()` et au `ConstraintLayout` qui maintient une interface fluide et réactive.

*Pour plus de détail, merci de regarder la section **Réponses aux questions**, ci-après.*

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.

Réponse :

Pour configurer correctement un champ texte multi-lignes comme `remark`, il est nécessaire de définir certains attributs dans le fichier XML. Voici comment s'assurer que le champ permet les retours à la ligne, le correcteur d'orthographe et que la taille s'ajuste correctement:

- Activer l'entrée multi-lignes: `android:inputType="textMultiLine"` permet d'autoriser les retours à la ligne.
- Correcteur orthographique: L'attribut `android:inputType` inclue aussi le correcteur orthographique `textAutoCorrect`.
- Ajustement de la hauteur: Utiliser `android:layout_height` avec une hauteur `fixe` ou `wrap_content` pour s'adapter aux lignes supplémentaires.
- Défilement vertical (si la hauteur est limitée): `android:scrollbars="vertical"`.

```
<EditText
    android:id="@+id/etCommentaires"
    android:layout_width="0dp"
    android:layout_height="60dp"
    android:layout_weight="0.6"
    android:inputType="textMultiLine|textAutoCorrect"
    android:gravity="top"
    android:scrollbars="vertical"
/>
```

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 ?

Réponse :

Pour afficher les dates en fonction de la langue de l'utilisateur (par exemple, « 12 juin 1996 » en français ou « June 12, 1996 » en anglais), il faut utiliser `DateFormat.getDateInstance()` avec différents styles de DateFormat. Cette méthode formatte les dates en fonction des paramètres régionaux de l'appareil.

Code présent dans Person.kt :

```
val dateFormatter = DateFormat.getDateInstance()
```

Question 4.3

Veuillez choisir une question en fonction de votre choix d'implémentation :

- **a. Si vous avez utilisé le DatePickerDialog1 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 ?**

- *b. 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 ?

Réponse :
L'exception `android.view.WindowLeaked` survient lorsque le `DatePickerDialog` est affiché et que l'activité ou le fragment qui l'a créé est détruite sans que le dialogue ne soit correctement annulé ou fermé. En cas de rotation de l'écran, Android recrée par défaut l'activité pour ajuster l'interface utilisateur à la nouvelle orientation. Pendant cette recréation, si un dialogue reste ouvert et qu'il n'a pas été correctement fermé, Android perçoit cela comme une fuite de fenêtre `WindowLeaked`, car l'activité d'origine est détruite, mais le dialogue peut rester attaché à une "vieille" référence de cette activité.

Cependant nous n'avons pas trouvé trace de cette erreur dans les logs. Les raisons suivantes semblent pouvoir l'expliquer en partie :

- Rotation Rapide ou Instantanée : Il est possible que la rotation soit assez rapide pour que le `DatePickerDialog` n'ait pas le temps de causer une fuite avant d'être automatiquement détruit. Dans les versions plus récentes d'Android, le système peut gérer cela plus efficacement en annulant le dialogue associé à une activité en cours de destruction.
- Absence de Références Persistantes : Le `DatePickerDialog` n'est pas stocké dans une variable de l'activité (comme une variable de classe), ce qui réduit les risques de fuite. Étant donné qu'il est instantanément créé et affiché dans `showDatePicker()` sans être retenu, il est automatiquement libéré avec l'activité dans de nombreux cas, ce qui évite potentiellement l'exception.
- Pas de Création Répétée en Boucle : Puisque `showDatePicker()` est appelé depuis un événement spécifique (un clic), le dialogue n'est pas créé de manière répétitive, réduisant ainsi le risque de fuite.

Une autre solution consisterait à utiliser un `DialogFragment` pour encapsuler le `DatePickerDialog`, ce qui permet à Android de mieux gérer la recréation du dialogue lors des changements de configuration (comme la rotation de l'écran). En encapsulant le `DatePickerDialog` dans un `DialogFragment`, celui-ci persiste lors de la rotation et est recréé automatiquement avec l'activité, éliminant ainsi le problème de `WindowLeaked`. Il semble d'ailleurs que cela soit une bonne pratique d'android studio.

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 ?
Hint : Le champ remark, multilignes, peut provoquer des effets de bords en fonction du clavier virtuel utilisé sur votre smartphone. Vous pouvez l'échanger avec le champ e-mail pour faciliter vos recherches concernant la réponse à cette question.

Réponse :
Oui pour contrôler l'ordre de navigation entre les champs de texte, il faut utiliser `android:nextFocusForward="@id/nextField"` sur chaque EditText pour définir le champ suivant à atteindre lorsque « Suivant » est pressé. Pour le dernier champ, il faut associer le bouton d'action du clavier à la soumission du formulaire en utilisant `android:imeOptions="actionDone"` et gérer l'événement `EditorActionListener`.

Dans le fichier activity_main.xml:

```
<EditText
    android:id="@+id/etEmail"
    android:nextFocusForward="@id/etCommentaires"
    android:imeOptions="actionNext"
/>

<EditText
    android:id="@+id/etCommentaires"
    android:imeOptions="actionDone"
/>
```

Et dans le fichier MainActivity.kt :

```
etCommentaires.setOnEditorActionListener { _, actionId, _ ->
    if (actionId == EditorInfo.IME_ACTION_DONE && validateFields()) {
        processForm() // appel à la gestion du submit par exemple
        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 ?

Réponse :

Pour afficher un choix par défaut (comme « Sélectionner ») dans les champs Spinner (par exemple, pour la nationalité), il faut ajouter une entrée par défaut à la position 0 dans la liste de l'adaptateur. Lors de la validation de la soumission du formulaire il faut vérifier si la sélection du spinner correspond à ce choix.

Par exemple :

```
val nationalities = listOf("Sélectionner", "Suisse", "Française", "Allemande")
val adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item, nationalities)
spinnerNationalite.adapter = adapter

// Vérification de la sélection par défaut
// autre possibilitée : spinnerNationalite.selectedItemPosition == 0
if (spinnerNationalite.selectedItem.toString() == "Sélectionner") {
    // Gérer comme non sélectionné
}
```