Labo 3 - Approche MVC

DAA

Timothée Van Hove, Léo Zmoos

13 novembre 2023

Table des matières

Introduction

Dans ce laboratoire, nous allons réaliser une simple application contenant un formulaire à remplir par l'utilisateur. Le but de ce laboratoire est d'implémenter de nombreux widgets (Spinner, EditText, DatePicker, RadioGroup) dans un formulaire et de gérer l'application dans une approche MVC, afin d'en voir les limites.

Choix d'implémentation

Gestion des Radio Buttons

Nous avons rendu obligatoire la sélection d'une occupation dans notre formulaire. Ceci est géré dans la méthode `validateInputs()`, qui est appelée lors de l'activation du bouton "OK". De plus, l'affichage des champs spécifiques à un `Worker` ou à un `Student` est dynamiquement ajusté en fonction de la sélection de l'utilisateur. Cela est réalisé grâce à la manipulation de la propriété `visibility` des groupes concernés dans le layout XML, où nous alternons entre les valeurs `gone` et `visible`.

Gestion de la Date d'Anniversaire

Par défaut, si aucune occupation n'est choisie, la date d'anniversaire est réglée sur la date actuelle. Si une occupation est sélectionnée, la date par défaut est celle spécifiée dans `Person.exampleStudent` ou `Person.exampleWorker`. Lorsque l'utilisateur interagit avec l'`EditText` ou l'`ImageButton` de la date d'anniversaire, un `MaterialDatePicker` s'affiche. Nous avons empêché la sélection de dates futures ou de dates remontant à plus de 110 ans.

Gestion du Spinner de Nationalité

Pour le choix de la nationalité, un adaptateur personnalisé a été mis en place pour le Spinner. Cela nous a permis d'intégrer l'option "Sélectionner" comme un choix initial, tout en évitant son apparition dans les options sélectionnables du Spinner.

Gestion du Bouton "OK"

Le bouton "OK" a pour rôle de finaliser la saisie du formulaire. Lorsqu'il est activé, il déclenche la création d'une instance de `Worker` ou `Student` et enregistre les informations correspondantes dans Logcat pour un suivi facilité. La méthode `validateInputs()` est employée pour vérifier que tous les champs nécessaires sont correctement remplis. En cas de champ manquant, une indication visuelle apparaît à côté de l'`EditText` ou du `RadioButton` concerné, signalant à l'utilisateur les corrections à apporter.

Gestion du Bouton "ABORT"

La fonctionnalité du bouton "Cancel" est de remettre à zéro le formulaire. En activant ce bouton, l'utilisateur déclenche la méthode `resetForm()`, qui s'occupe d'effacer toutes les données saisies dans les champs du formulaire, y compris la réinitialisation des sélections des radio buttons et des autres widgets.

Réponse aux questions

Question 1

> 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 activer la saisie sur plusieurs lignes, il faut définir l'attribut `inputType` comme `textMultiLine`.
- Pour activer le correcteur orthographique, il faut définir l'attribut `inputType` comme `textAutoCorrect` et/ou `textAutoComplete`, selon le cas.
- Pour permettre à l'EditText d'augmenter sa taille au fur et à mesure que des lignes de texte sont ajoutées, il faut fixer l'attribut `maxLines` à une valeur appropriée (ou ne pas le fixer pour qu'il n'y ait pas de maximum). Il faut aussi que l'attribut `height` soit défini sur `wrap_content` pour permettre la croissance de la taille. Voici notre configuration pour le champ :

```
<EditText
        android:id="@+id/edit_text_comments"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:autofillHints="@string/additional_remarks_title"
        android:gravity="top"
        android:inputType="textAutoComplete|textMultiLine"
        android:minHeight="100dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/text_view_comments" />
```

Question 2

> 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 manières dont les dates sont affichées selon les langues et les régions, Android offre une solution robuste avec la classe `DateFormat`. Cette classe adapte automatiquement le format de la date en fonction de la locale définie sur le dispositif de l'utilisateur. Ainsi, les mois, les jours de la semaine et l'ordre des éléments dans la date (jour, mois, année) sont automatiquement ajustés pour correspondre aux conventions locales.

Dans notre application, pour afficher la date dans le format adapté à la locale de l'utilisateur, nous utilisons `DateFormat.getDateInstance()`. Cette méthode retourne un formateur de date configuré selon le style et la locale par défaut du dispositif. Par exemple, pour un utilisateur en France, la date sera formatée différemment que pour un utilisateur aux États-Unis ou au Royaume-Uni. Voici comment nous utilisons `DateFormat` dans notre application :

```
val dateFormatter = DateFormat.getDateInstance(DateFormat.LONG, Locale.getDefault())
editTextBirthdate.setText(dateFormatter.format(Date()))
```

Avec cette approche, la date sélectionnée via le `DatePicker` sera affichée dans un format qui est familier à l'utilisateur, respectant ainsi les variations linguistiques et régionales.

Question 3

> 4.3 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 ?

Oui, c'est possible de limiter les dates sélectionnables dans un `MaterialDatePicker`. Pour ce faire, il faut définir un objet `CalendarConstraints` lors de la création de notre datePicker. Cet objet va ajouter des contraintes temporelles minimum et maximum. Voici par exemple l'implémentation de notre méthode `initDatePicker` :

```
    private fun initDatePicker() {

        //Forbid dates in the future and dates less than 110 years in the past
        val dateConstraints = CalendarConstraints.Builder()
            .setValidator(DateValidatorPointBackward.now())
            .setStart(Calendar.getInstance().also { it.add(Calendar.YEAR, -110) }.timeInMillis)
            .build()

        datePicker = MaterialDatePicker.Builder.datePicker()
            .setTitleText(resources.getString(R.string.main_base_birthdate_dialog_title))
            .setSelection(MaterialDatePicker.todayInUtcMilliseconds())
            .setCalendarConstraints(dateConstraints)
            .build()
    }
```

Question 4

> 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.
>
> 1) Est-ce possible de spécifier son propre ordre de remplissage du questionnaire ?
> 2) Arrivé sur le dernier champ, est-il possible de faire en sorte que ce bouton soit lié au bouton de validation du questionnaire ?

1. Oui c'est possible de définir un ordre avec l'attribut `android:nextFocusDown` dans le layout. Cet attribut défini l'ID de la prochaine vue sur laquelle mettre le focus quand le bouton `next` est appuyé
2. Oui, c'est possible avec l'attribut `android:imeOptions="actionSend"` ou `android:imeOptions="actionDone"` dans le dernier `EditText` du layout. Ensuite il faut implémenter un `OnEditorActionListener` sur cet `EditText` dans le code de l'activité pour gérer l'action. Par exemple:

```
<EditText
          android:id="@+id/editText"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:imeOptions="actionSend" />
```

```
binding.editTextB.setOnEditorActionListener { _, actionId, _ ->
    if (actionId == EditorInfo.IME_ACTION_SEND) {
        submitForm() // Methode qui va envoyer le formulaire (qu'il  faut implémenter soi-même)
        true
    } else false
}
```

Conclusion

Au terme de ce laboratoire, nous avons exploré l'approche MVC. Nous avons rencontré des défis, notamment dans la gestion de l'état de l'application et la synchronisation des données entre la vue et le modèle, qui ont parfois complexifié le développement. Nous sommes donc impatients d'adopter l'approche MVVM qui permettra une plus grande simplicité et une meilleure séparation des responsabilités. En particulier, elle devrait nous permettre de simplifier la gestion de l'état, grâce à l'utilisation de LiveData et ViewModel.