Rapport Laboratoire 3 - DAA

Auteurs: Eva Ray, Massimo Stefani, Rachel Tranchida

Date : 30 octobre 2024

Choix d'implémentation

View Binding

Nous avons décidé d'utiliser le `ViewBinding` pour faciliter l'écriture du code qui interagit avec les vues. Pour cela, nous avons simplement ajouté le code ci-dessous dans le fichier `gradle.build`

```
android {
    ...
    buildFeatures {
        viewBinding = true
    }
}
```

Cela permet d'activer la liaison de vue. Une classe de liaison est alors générée pour chaque Fichier de mise en page XML que contient le module. Nous passons donc par notre binding `ActivityMainBinding` pour interagir avec nos vues.

Validation des données

Nous avons décidé d'ajouter un petit mécanisme de validation des données du formulaire. Nous obligeons tout simplement tous les champs à être remplis lors de la validation du formulaire, donc au moment de l'appui sur le bouton `Ok`. Si tel n'est pas le cas, un message `Toast` contenant le texte "Tous les champs sont obligatoires" apparait.

Bouton Ok

Lorsqu'on appuie sur le bouton `Ok`, et qu'on a spécifié l'occupation, le listener du bouton va vérifier si on a sélectionné `Employé` ou `Etudiant`. Si `Employé` est coché, on crée un objet `Worker`, si `Etudiant` est coché, on crée un objet `Student`. Dans les deux cas, l'objet créé est imprimé dans les logs, afin qu'on puisse vérifier que la création s'est bien passée. Une fois la validation faite, tous les champs du formulaire sont vidés.

Bouton Annuler

Lorsqu'on appuie sur le bouton `Annuler`, tous les champs du formulaire sont reset, y compris les champs moins classiques, comme le calendrier, les boutons radio ou les spinners.

Questions complémentaires

> **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 pouvoir faire des retours à la ligne, il faut ajouter la valeur `textMultiline` à l'attribut `inputType` de l'`EditText` pour le commentaire.

Pour activer le correcteur orthographique, on peut ajouter la valeur `textAutoCorrect` à l'attribut `inputType` de l'`EditText` pour le commentaire.

Pour gérer le nombre de lignes visibles dans le champ, on peut utiliser l'attribut `android::maxLines`, dont la valeur définit le nombre de lignes écrites visibles. On peut activer le défilement vertical en ajoutant l'attribut `android:scrollbars="vertical"` à l'`EditText`, dans le cas où le texte dépasse le nombre limite de lignes. Cela permet à l'utilisateur de voir le texte complet en faisant défiler.

Voici un extrait du code de l'`EditText` pour le commentaire, qui contient la configuration pour que son comportement soir correct.

```
android:inputType="textMultiLine|textAutoCorrect"
android:maxLines="5"
android:scrollbars="vertical"
```

> **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 ?

En inspectant certains fichiers fournis. Nous avons constaté que dans le classe `Person`, un attribut `dateFormatter` de type `DateFormat` est déclaré. Cela permet de formater la date de naissance d'une personne en une chaîne de caractères. Pourquoi cela marche? `DateFormat` est une classe abstraite qui permet de formater et de parser des dates en fonction de la locale de l'utilisateur. En utilisant `DateFormat.getDateInstance()`, on obtient une instance de `DateFormat` qui formate les dates en fonction de la locale de l'utilisateur. Cela permet de gérer automatiquement les différences de format de date en fonction de la langue et des habitudes régionales.

ref: DateFormat

> **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 ?

Nous avons utilisé `MaterialDatePicker` de la librairie Material pour afficher un calendrier de sélection de date. Grâce au `MaterialDatePicker.Builder`, il est possible de configurer diverses options pour le dialogue de sélection de date, y compris les contraintes de calendrier. L'objet `CalendarConstraints` permet de limiter les dates sélectionnables et utilise le pattern Builder, ce qui facilite la création de contraintes spécifiques pour notre cas d'utilisation.

Contraintes de date

Dans notre exemple, pour restreindre la sélection de dates à une plage plausible pour une date de naissance, nous allons définir deux contraintes :

1. **Empêcher la sélection de dates futures**
    - Cela assure que l'utilisateur ne puisse pas choisir une date au-delà de la date actuelle.
2. **Limiter l'âge maximum**
    - En fixant une limite de 110 ans en arrière, nous restreignons la sélection à une période de temps raisonnable pour une date de naissance.

Implémentation

Voici le code correspondant pour initialiser un `MaterialDatePicker` avec ces contraintes :

```
private fun initDatePicker() {
    // Création d'un MaterialDatePicker pour la sélection de date de naissance
    val datePicker = MaterialDatePicker.Builder.datePicker()

.setTitleText(resources.getString(R.string.main_base_birthdate_dialog_title)) //
Titre du dialogue
        .setSelection(MaterialDatePicker.todayInUtcMilliseconds()) // Sélectionne
la date actuelle par défaut
        .setCalendarConstraints(
            CalendarConstraints.Builder()
                .setValidator(DateValidatorPointBackward.now()) // Limite à
aujourd'hui (pas de date future)
                .setStart(Calendar.getInstance().apply {
                    add(Calendar.YEAR, -110)
                }.timeInMillis) // Limite l'âge max à 110 ans
                .build()
        )
        .build()
}
```

ref: MaterialDatePicker

Explications des éléments clés

- `setTitleText` : Définit le titre du dialogue, par exemple "Sélectionner la date de naissance".
- `setSelection` : Par défaut, la sélection est positionnée sur la date actuelle.

- `setCalendarConstraints` : Permet de définir des contraintes pour la sélection de dates.
  - `DateValidatorPointBackward.now()` : Limite la sélection aux dates antérieures ou égales à aujourd'hui.
  - `setStart` : Spécifie la date minimale sélectionnable en soustrayant 110 ans à la date actuelle pour éviter des dates de naissance improbables.

> **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. 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, il est possible de spécifier son propre ordre de remplissage de questionnaire. Pour cela, on peut utiliser l'attribut' `android:nextFocusDown`. Chaque champ peut être configuré pour spécifier quel champ doit être le suivant lors de l'appui sur le bouton "Suivant" du clavier. Cela se fait avec `android:nextFocusDown="<id prochain champ>"`. En configurant les bons `nextFocusDown` dans tous les `EditText`, on peut garantir le bon enchaînement des champs du formulaire.

Oui, il est aussi possible de faire en sorte que ce bouton soit lié au bouton de validation du questionnaire. Cela se fait à nouveau avec le paramètre `imeOptions`. En utilisant `android:imeOptions="actionDone"` sur le dernier champ du formulaire, Cela change l'apparence du bouton "Suivant" pour qu'il affiche "Valider" ou "Terminé". Ensuite, on peut gérer l'action de validation dans le code pour déclencher le traitement du formulaire. Dans le code Kotlin, on peut par exemple écouter l'action "Done" sur le dernier `EditText` avec un `setOnEditorActionListener`met déclencher la validation du formulaire. Voici un code d'exemple.

```
binding.editTextRemark.setOnEditorActionListener { _, actionId, _ ->
    if (actionId == EditorInfo.IME_ACTION_DONE) {
        submitFormData()
        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 ?

On peut créer un spinner adapter custom. Ici, on l'appelle `PlaceholderSpinnerAdapter` qui attend une liste dont le premier élément sera considéré comme un placeholder. On chosisit de le cacher dans la dropdown lorsque l'on clique dessus avec la fonction `getDropDownView`. De plus, lorsque la position 0 est sélectionnée (celle du placeholder), `isEnabled` retourne `false`. Pour donner les éléments à notre nouveau dropdown, on donne une liste qui est la concaténation du placeholder "Séléectionner" et de la liste de string représentants les choix du DropDown.