Rapport Labo 03

Autheurs

- Nathan Füllemann
- Mathéo Lopez
- Arnaud Tribolet

Réponses aux questions

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 rendre un champ EditText multiligne permettant les retours à la ligne dans le XML, il suffit d'ajouter les attributs :

```
android:inputType="textMultiLine"
android:gravity="top"
android:minLines="3"
```

Cela permet aux utilisateurs d'écrire sur plusieurs lignes, avec le texte aligné en haut et un minimum de trois lignes affichées par défaut.

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 habitudes régionales, nous utilisons `DateFormat.getDateInstance()` qui récupère automatiquement la locale du système. La timezone est définie en UTC pour éviter les problèmes de conversion avec le `DatePicker` :

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

Ainsi, le format de date s'adapte automatiquement à la configuration linguistique du téléphone.

3. **Si vous avez utilisé le MaterialDatePicker de la librairie Material. Est-il possible de limiter les dates sélectionnables dans le dialogue ? Ceci en particulier pour une date de naissance car 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, il est possible de limiter les dates sélectionnables dans `MaterialDatePicker` grâce à `CalendarConstraints`. Pour une date de naissance, nous appliquons trois restrictions : empêcher les dates futures avec `DateValidatorPointBackward.now()`, limiter la plage à 110 ans en arrière avec `setStart()`, et bloquer les dates après aujourd'hui avec `setEnd()` :

```
val datePicker = MaterialDatePicker.Builder.datePicker()
    .setCalendarConstraints(
        CalendarConstraints.Builder()
            .setOpenAt(currentTimestamp)
            .setValidator(DateValidatorPointBackward.now())
            .setStart(utcCalendar.also { it.add(Calendar.YEAR, -110)
}.timeInMillis)
            .setEnd(MaterialDatePicker.todayInUtcMilliseconds())
            .build()
    )
    .setSelection(currentTimestamp)
    .build()
```

Le paramètre `setOpenAt()` permet d'ouvrir le calendrier sur la date déjà sélectionnée, tandis que `setSelection()` présélectionne cette date dans le dialogue. Cela garantit que seules les dates de naissance réalistes sont sélectionnables.

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, il est possible de personnaliser la navigation au clavier avec deux approches :

Personnaliser l'ordre de navigation - L'attribut `android:nextFocusDown` définit quel champ recevoir le focus lors de l'appui sur "suivant" en spécifiant l'ID de la vue cible. L'attribut `android:imeOptions="actionNext"` indique au clavier d'afficher le bouton "suivant".

Lier le dernier champ à la validation - Pour le champ final (commentaires), nous utilisons `android:imeOptions="actionDone"` qui remplace "suivant" par "terminé". Un `OnEditorActionListener` intercepte cette action pour déclencher automatiquement la validation :

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

```
            false
        }
    }
```

Cette configuration permet une navigation fluide du formulaire jusqu'à sa soumission automatique.

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 comme "Sélectionner" dans un `Spinner`, nous créons un adapter personnalisé `SpinnerDefaultValueAdapter` qui ajoute cette valeur en première position :

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

Pour éviter toute confusion, deux méthodes empêchent la sélection de cette valeur par défaut :

```
override fun getDropDownView(position: Int, convertView: View?, parent:
ViewGroup): View {
    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 {
    return position != 0 && super.isEnabled(position)
}
```

La première masque l'option par défaut dans la liste déroulante, la seconde empêche sa sélection. Ainsi, la valeur par défaut reste visible uniquement quand aucun choix n'est fait, permettant de valider que l'utilisateur a effectué une sélection réelle.

Implementation

Architecture

Notre implémentation utilise un **single ConstraintLayout** imbriqué dans un `CoordinatorLayout` avec `NestedScrollView` pour gérer l'affichage adaptatif du formulaire. Le binding automatique simplifie la récupération des éléments et la gestion des interactions. Une `Toolbar` personnalisée remplace l'ActionBar par défaut.

Gestion des états dynamiques

Nous utilisons deux composants clés pour basculer entre l'affichage étudiant et employé :

Barrier - Référence automatiquement les éléments du bas du formulaire, permettant aux champs suivants de se positionner indépendamment du contexte actif (employé ou étudiant).

Group - Contrôle la visibilité de plusieurs éléments associés en une seule action, simplifiant la gestion des champs spécifiques à chaque type de personne.

Cohérence visuelle

Les dimensions standardisées (`title_size: 16sp`, `element_margin: 8dp`) sont réutilisées dans tout le layout pour assurer l'uniformité et faciliter les modifications globales.

Interactions principales

Date d'anniversaire - `MaterialDatePicker` initialise la date courante et met à jour le champ texte avec la locale du système. Les contraintes de calendrier limitent les dates sélectionnables à 110 ans en arrière jusqu'à aujourd'hui.

Type de personne - Un listener sur le `RadioGroup` affiche/masque dynamiquement les champs correspondants selon l'occupation sélectionnée.

Spinners - Un adapter spécialisé ajoute une valeur par défaut "Sélectionner" non-sélectionnable en position 0. Les méthodes `getDropDownView()` et `isEnabled()` empêchent la sélection et l'affichage de cette valeur par défaut.

Validation - Le champ commentaires avec `android:imeOptions="actionDone"` déclenche automatiquement la validation au clavier virtuel. La validation contrôleur utilise des méthodes réutilisables pour vérifier les champs obligatoires et affiche des toasts de confirmation ou d'erreur.

Utilisation IA

Nous avons utilisé ChatGPT afin de restructurer notre rapport et corriger l'orthographe. Github Copilot a été utilisé pour la génération de commentaires dans le code.