Todo List pour Android

« Réalisation d’une todo list basique pour Android. »

Par Vasco Valadares Semana – En 2023 – Projet de Cours

Table des matières

1. Introduction

Ce document présente la conception et l'implémentation du projet "Todo List pour Android". Ce projet a été développé dans le but de s'initier à la création d'applications Android interactives, en se concentrant sur la gestion de listes de données et l'utilisation d'adaptateurs personnalisés. L'application permet aux utilisateurs d'ajouter des tâches à une liste et de les supprimer, offrant une fonctionnalité de base pour une application de gestion de tâches.

2. Objectifs du Projet

2.1 Contexte :

Ce projet a été réalisé dans le cadre d'un apprentissage pratique du développement mobile Android, visant à consolider les compétences en Java et l'utilisation d'Android Studio.

2.2 Objectifs principaux :

- **Apprentissage de ListView et ArrayAdapter :** Comprendre et mettre en œuvre l'affichage de listes dynamiques de données ;
- **Création d'un Adaptateur Personnalisé (Custom Adapter) :** Développer un ArrayAdapter personnalisé pour contrôler l'affichage de chaque élément de la liste et y intégrer des interactions spécifiques (comme un bouton de suppression) ;
- **Manipulation de Collections de Données :** Utiliser ArrayList pour stocker et gérer les tâches ;
- **Gestion des Événements UI :** Implémenter la logique pour répondre aux actions de l'utilisateur (ajout et suppression de tâches) ;
- **Principes de Base d'Android Studio :** Renforcer la compréhension de la structure d'un projet Android, des layouts XML et du cycle de vie d'une activité.

3. Architecture de l'Application

3.1. Structure du Projet

L'application "Todo List pour Android" est structurée autour des composants Android standards :

- **Activité Principale (MainActivity.java) :** Gère l'interface utilisateur principale, y compris le champ de saisie pour les nouvelles tâches, le bouton d'ajout, et la ListView affichant les tâches. Elle contient la logique pour ajouter des tâches à la source de données ;
- **Adaptateur Personnalisé (TaskAdapter.java) :** Une classe qui hérite de ArrayAdapter<String>. Elle est responsable de la création et de la gestion des vues pour chaque élément de la ListView, y compris l'affichage du texte de la tâche et du bouton de suppression.
- **Layouts XML :**
    - activity_main.xml : Définit la structure visuelle de l'écran principal (champ de saisie, bouton, ListView) ;

    - list_item.xml : Définit la structure visuelle de chaque ligne (item) dans la ListView (un TextView pour la tâche et un ImageButton pour la suppression).
- **Ressources Android :** Les identifiants des vues (R.id), les chaînes de caractères, etc., sont gérés via le système de ressources Android.

3.2. Choix Technologiques

- **Langage de Programmation :** Java ;
- **Environnement de Développement Intégré (IDE) :** Android Studio ;
- **Composants Android Clés :** AppCompatActivity, EditText, ListView, ArrayAdapter, ImageButton, TextView, LayoutInflater ;
- **Gestion de Version :** Git et GitHub.

4. Structure des Données et Composants Clés

4.1. MainActivity.java

- EditText taskInput;
    - **Rôle :** Champ de texte où l'utilisateur saisit la description d'une nouvelle tâche ;
    - **Initialisation :** findViewById(R.id.task_input) dans onCreate().
- ListView taskList;
    - **Rôle :** Composant d'interface qui affiche la liste des tâches ;
    - **Initialisation :** findViewById(R.id.task_list) dans onCreate().
- ArrayList<String> tasks;
    - **Rôle :** Collection dynamique (ArrayList) qui stocke les chaînes de caractères représentant les tâches. C'est la source de données pour la ListView ;
    - **Initialisation :** new ArrayList<>() dans onCreate().
- TaskAdapter taskAdapter;
    - **Rôle :** Instance de l'adaptateur personnalisé qui fait le lien entre l'ArrayList tasks et la ListView taskList ;
    - **Initialisation :** new TaskAdapter(this, tasks) dans onCreate(), puis assigné à taskList via taskList.setAdapter(taskAdapter).

4.2. TaskAdapter.java

- private final Context context;
    - **Rôle :** Référence au contexte de l'application (généralement l'activité), nécessaire pour le LayoutInflater ;
    - **Initialisation :** Passé via le constructeur.
- private final ArrayList<String> tasks;

- **Rôle :** Référence à la même ArrayList de tâches que celle utilisée dans MainActivity ;
- **Initialisation :** Passé via le constructeur.

5. Fonctionnalités Principales et Implémentation

5.1. Interface Utilisateur et Affichage (activity_main.xml, list_item.xml)

- **activity_main.xml :**
  - Contient un EditText (R.id.task_input) pour la saisie de nouvelles tâches ;
  - Un Button (ou ImageButton) avec la méthode android:onClick="addTask" pour déclencher l'ajout de la tâche ;
  - Une ListView (R.id.task_list) pour afficher la liste des tâches.
- **list_item.xml :**
  - Ce layout est "gonflé" (inflated) par TaskAdapter pour chaque tâche dans la liste ;
  - Il contient un TextView (R.id.task_item) pour afficher le texte de la tâche ;
  - Un ImageButton (R.id.remove_task) à côté de chaque tâche pour permettre sa suppression.

5.2. Ajout de Tâches (MainActivity.addTask)

Cette méthode publique est appelée lorsque l'utilisateur clique sur le bouton d'ajout (grâce à l'attribut android:onClick="addTask" dans le fichier XML du layout activity_main.xml).

- Récupère le texte saisi dans taskInput ;
- Vérifie si le texte n'est pas vide ;
- Si non vide, la nouvelle tâche (chaîne de caractères) est ajoutée à l'ArrayList tasks ;
- taskAdapter.notifyDataSetChanged() est appelé pour informer la ListView que les données ont changé, ce qui provoque un rafraîchissement de l'affichage ;
- Le champ taskInput est vidé pour la saisie suivante.

```java
// Dans MainActivity.java
public void addTask(View view) {
    String task = taskInput.getText().toString();
    if (!task.isEmpty()) {
        tasks.add(task);
        taskAdapter.notifyDataSetChanged(); // Notifie l'adaptateur du changement
        taskInput.setText(""); // Vide le champ de saisie
    }
}
```

5.3. Affichage des Tâches (TaskAdapter.getView)

La méthode getView() de TaskAdapter est au cœur de l'affichage de chaque élément de la liste.

- Elle est appelée par la ListView pour chaque tâche à afficher ;
- **Réutilisation des Vues (convertView) :** Si convertView est null, un nouveau layout list_item.xml est "gonflé" (inflated) en utilisant LayoutInflater. Sinon, la vue existante est réutilisée pour des raisons de performance ;
- **Population des Données :** Le TextView (R.id.task_item) dans la vue de l'item est récupéré, et son texte est défini avec la tâche correspondante de l'ArrayList tasks à la position donnée ;
- **Configuration du Bouton de Suppression :** Un OnClickListener est attaché à l'ImageButton remove_task de l'item.

```java
// Dans TaskAdapter.java
@Override
public View getView(final int position, View convertView, ViewGroup parent) {
    if (convertView == null) {
        LayoutInflater inflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        convertView = inflater.inflate(R.layout.list_item, parent, false);
    }

    TextView taskItem = convertView.findViewById(R.id.task_item);
    taskItem.setText(tasks.get(position)); // Affiche le texte de la tâche

    ImageButton removeTaskButton = convertView.findViewById(R.id.remove_task);
    // ... (logique de suppression, voir section suivante) ...

    return convertView;
}
```

5.4. Suppression de Tâches (Logique dans TaskAdapter)

La logique de suppression est implémentée dans l'OnClickListener du bouton remove_task à l'intérieur de la méthode getView() de TaskAdapter.

- Lorsque le bouton de suppression d'un item est cliqué :
    - La tâche correspondante est supprimée de l'ArrayList tasks en utilisant tasks.remove(position). La position est final car elle est accédée depuis une classe interne anonyme ;
    - notifyDataSetChanged() est appelé sur l'adaptateur lui-même pour signaler à la ListView de se rafraîchir et de refléter la suppression.

```java
// Dans TaskAdapter.java, à l'intérieur de getView()
ImageButton removeTaskButton = convertView.findViewById(R.id.remove_task);
removeTaskButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        tasks.remove(position); // Supprime la tâche de la liste de données
        notifyDataSetChanged(); // Notifie l'adaptateur pour rafraîchir la ListView
    }
});
```

6. Points Particuliers et Solutions Techniques

- **ArrayAdapter Personnalisé (TaskAdapter) :** La création d'une classe TaskAdapter héritant d'ArrayAdapter était nécessaire pour avoir un contrôle total sur le layout de chaque item de la liste (incluant un TextView et un ImageButton) et pour gérer l'événement de clic sur le bouton de suppression spécifique à chaque item ;
- **ListView et ArrayList :** Utilisation de composants Android standards pour la gestion et l'affichage de listes dynamiques. ArrayList<String> sert de source de données simple et flexible ;
- **LayoutInflater :** Utilisé dans TaskAdapter pour créer des instances de vues à partir du fichier de layout XML list_item.xml pour chaque tâche ;
- **notifyDataSetChanged() :** Méthode cruciale appelée après toute modification de la source de données (tasks ArrayList) pour indiquer à la ListView (via son adaptateur) qu'elle doit redessiner ses vues pour refléter les changements (ajout ou suppression de tâches) ;
- **Gestion des Événements onClick :**
    - Pour le bouton d'ajout dans MainActivity : via l'attribut android:onClick dans le XML, liant le bouton à la méthode addTask(View view) ;
    - Pour les boutons de suppression dans TaskAdapter : via setOnClickListener programmatiquement sur chaque ImageButton dans la méthode getView().

7. Interface Utilisateur (Description et Interactions)

7.1. Vue Générale de l'Interface

L'interface utilisateur de l'application "Todo List" est simple et fonctionnelle :

- **Champ de Saisie :** Un EditText en haut de l'écran permet à l'utilisateur de taper le texte d'une nouvelle tâche ;
- **Bouton d'Ajout :** Un bouton « Ajouter » à côté du champ de saisie ;
- **Liste des Tâches :** Une ListView occupe la majeure partie de l'écran, affichant chaque tâche sur une ligne distincte ;

- **Items de la Liste :** Chaque ligne de la liste affiche le texte de la tâche et un bouton (icône de corbeille) pour supprimer cette tâche spécifique.

7.2. Interactions Utilisateur

1. **Ajouter une tâche :**
    - L'utilisateur tape le nom de la tâche dans le champ EditText ;
    - L'utilisateur clique sur le bouton "Ajouter" ;
    - La nouvelle tâche apparaît à la fin de la ListView. Le champ de saisie est vidé.
2. **Visualiser les tâches :**
    - Les tâches ajoutées sont listées verticalement dans la ListView.
3. **Supprimer une tâche :**
    - L'utilisateur clique sur le bouton de suppression (icône) à côté de la tâche qu'il souhaite enlever ;
    - La tâche est immédiatement retirée de la ListView.

8. Perspectives d’Évolution et Améliorations Futures

L'application actuelle constitue une base fonctionnelle. De nombreuses améliorations sont possibles :

- **Persistance des Données :** Utiliser SharedPreferences pour une sauvegarde simple des tâches entre les sessions, ou implémenter une base de données SQL pour une gestion plus robuste et structurée des tâches, permettant de stocker plus d'informations (ex : date d'échéance, priorité, etc) ;
- **Fonctionnalités Étendues pour les Tâches :** Permettre la modification (édition) des tâches existantes, ajouter la possibilité de marquer des tâches comme "complétées" (par exemple, avec une case à cocher et un style visuel différent, comme un texte barré), introduire des dates d'échéance et des rappels (notifications), permettre de définir des priorités pour les tâches, etc ;
- **Améliorations de l'Interface Utilisateur (UI/UX) :** Améliorer le design visuel (thèmes, icônes plus esthétiques), ajouter des animations ou des transitions pour une expérience plus fluide, permettre le tri ou le réordonnancement manuel des tâches (par drag-and-drop), afficher des messages de confirmation ou d'annulation pour la suppression, etc ;
- **Gestion des Erreurs et Cas Limites :** Ajouter des méthodes de gestion des erreurs, comme des « try/catch » et l’affichage de notifications d’erreur ;
- **Architecture et Code :** Pour des fonctionnalités plus complexes, envisager une architecture plus complexe pour une meilleure séparation des préoccupations et une gestion du cycle de vie plus robuste. Écrire des tests unitaires avec JUnit par exemple.

9. Conclusion

Le projet "Todo List pour Android" m’a permis de mettre en œuvre une application de gestion de tâches basique, atteignant les objectifs d'apprentissage fixés concernant l'utilisation de ListView, la création d'adaptateurs personnalisés (TaskAdapter), et la manipulation de collections de données en Java pour Android. Bien que simple dans ses fonctionnalités actuelles, ce projet m’a aidé à explorer des concepts plus avancés du développement Android et pour ajouter des fonctionnalités plus riches que les précédents.