Rapport de TP2

Application de Gestion de Notes
MyNotes

Développement Mobile Android

Programmation Java Native

Réalisé par : Souidak Firdaous

Encadré par : Pr. AZYAT Abdelilah

Année Universitaire : 2025-2026

Table des matières

1 Introduction

1.1 Contexte du projet

Dans le cadre du cours de développement mobile, ce travail pratique a pour objectif de concevoir et développer une application Android native permettant la gestion de notes personnelles. L'application **MyNotes** offre une solution simple et organisée pour créer, consulter et gérer des notes avec différents niveaux de priorité.

Le développement Android natif en Java reste une compétence essentielle dans l'industrie du développement mobile. Malgré l'émergence de Kotlin comme langage préféré pour Android, Java demeure largement utilisé dans de nombreux projets existants et continue d'être enseigné comme base solide pour comprendre l'écosystème Android. Ce TP nous permet de nous familiariser avec les API Android, l'architecture des applications mobiles et les patterns de développement spécifiques à cette plateforme.

1.2 Objectifs pédagogiques

Les principaux objectifs de ce TP sont multiples et couvrent différents aspects du développement Android. Premièrement, nous cherchons à maîtriser les concepts fondamentaux du développement Android en Java, notamment la structure d'un projet Android et l'organisation du code selon les conventions de la plateforme. Deuxièmement, nous devons implémenter une architecture multi-activités avec navigation par Intents, ce qui nous permet de comprendre comment les différentes parties d'une application communiquent entre elles. Troisièmement, nous apprenons à créer des interfaces utilisateur adaptatives avec ListView et adaptateurs personnalisés, des composants essentiels pour afficher des collections de données. Enfin, nous appliquons les bonnes pratiques de développement mobile en termes d'organisation du code, de séparation des responsabilités et de design patterns.

1.3 Fonctionnalités principales

L'application MyNotes offre plusieurs fonctionnalités essentielles qui répondent aux besoins basiques de gestion de notes. La première fonctionnalité est la visualisation de la liste complète des notes sur l'écran d'accueil, permettant à l'utilisateur d'avoir une vue d'ensemble de toutes ses notes en un coup d'œil.

La deuxième fonctionnalité majeure est l'ajout de nouvelles notes via un formule dédié et intuitif. Ce formulaire permet de saisir un titre, une description détaillée, une date et de sélectionner un niveau de priorité parmi trois options : basse, moyenne ou haute. Cette structuration permet une organisation efficace des notes selon leur importance.

La troisième fonctionnalité consiste en la consultation détaillée d'une note spécifique. En cliquant sur une note dans la liste, l'utilisateur accède à un écran dédié affichant toutes les informations de manière claire et structurée, facilitant ainsi la lecture et la consultation des détails.

Une fonctionnalité supplémentaire et enrichissante est la capture et l'attachement de photos depuis la caméra ou la galerie. Cette fonctionnalité multimédia permet d'ajouter un contexte visuel aux notes, rendant l'application plus versatile et utile pour documenter des informations nécessitant des images.

2 Architecture de l'application

2.1 Structure générale

L'application MyNotes adopte une architecture en couches inspirée du pattern MVC, couramment utilisée dans les applications Android. Cette organisation assure une séparation claire des responsabilités et facilite la maintenance du code. Le projet est structuré en plusieurs packages principaux, chacun jouant un rôle précis dans le fonctionnement de l'application.

Le package `activities` regroupe les écrans de l'application et gère les interactions utilisateur. Le package `adapter` contient les adaptateurs nécessaires à l'affichage des listes, tandis que le package `model` définit les entités métier, notamment la classe `Note`. Enfin, le dossier `res` rassemble les ressources telles que les layouts XML, les couleurs et les chaînes de caractères.

Structure des packages

```
com.example.mynotes
activities/
adapter/
model/
res/
```

Cette structure améliore la lisibilité du code, respecte les conventions Android et facilite l'évolution future de l'application.

2.2 Diagramme des activités

L'application repose sur trois activités principales. **NoteListActivity** constitue l'écran d'accueil et affiche la liste des notes avec un code couleur indiquant leur priorité. **AddNoteActivity** permet la création d'une nouvelle note à travers un formulaire incluant des options multimédias. **DetailsNoteActivity** affiche les informations détaillées d'une note sélectionnée.

La navigation entre ces écrans s'effectue via des Intents, suivant un flux simple et intuitif : consultation de la liste, ajout ou affichage d'une note, puis retour à l'écran principal. Cette organisation garantit une expérience utilisateur fluide et conforme aux standards Android.

2.3 Modèle de données

2.3.1 Classe Note

La classe `Note` constitue le modèle de données principal de l'application. Elle encapsule les informations d'une note en suivant le pattern POJO, avec des attributs privés et des méthodes d'accès publiques, garantissant ainsi l'encapsulation et la simplicité du modèle. Sa conception légère facilite la manipulation et la transmission des données entre les différentes composantes de l'application.

Les attributs de la classe répondent directement aux besoins fonctionnels : le nom correspond au titre de la note, la description contient son contenu, la date indique le moment de création ou d'échéance, et la priorité définit son niveau d'importance. Cette structure claire permet une gestion efficace et cohérente des données au sein de l'application.

```java
package com.example.mynotes.model;

public class Note {
    private String nom;
    private String description;
    private String date;
    private String priorite;

    public Note(String nom, String description,
                String date, String priorite) {
        this.nom = nom;
        this.description = description;
        this.date = date;
        this.priorite = priorite;
    }

    public String getNom() { return nom; }
    public String getDescription() { return description; }
    public String getDate() { return date; }
    public String getPriorite() { return priorite; }
}
```

Listing 1 – Classe Note.java

3 Implémentation des activités

3.1 NoteListActivity – Liste des notes

La `NoteListActivity` constitue l'écran principal de l'application et le point d'entrée pour l'utilisateur. Elle affiche l'ensemble des notes sous forme de liste et permet l'accès aux autres fonctionnalités de l'application. Cette activité gère l'affichage des données, la navigation vers les écrans d'ajout et de détail, ainsi que la conservation de la liste des notes en mémoire pendant l'exécution de l'application.

3.1.1 Composants principaux

Cette activité repose sur trois composants essentiels. La **ListView** permet l'affichage dynamique des notes à l'aide d'un adaptateur. Le **bouton d'ajout** offre un accès rapide à la création d'une nouvelle note. Enfin, le **NoteAdapter** assure la liaison entre les objets `Note` et leur représentation visuelle, en appliquant notamment un formatage adapté à la priorité de chaque note.

3.1.2 Fonctionnement général

Lors de l'initialisation dans la méthode `onCreate`, les composants de l'interface sont configurés et l'adaptateur est associé à la `ListView`. Un `Intent` explicite permet de lancer l'activité d'ajout de note, tandis qu'un clic sur un élément de la liste ouvre l'activité de détails correspondante. Les données sont transmises entre activités à l'aide des mécanismes standards d'Intents.

3.1.3 Gestion du résultat

La méthode `onActivityResult` permet de récupérer les informations de la note créée après la fermeture de `AddNoteActivity`. Lorsque le résultat est valide, une nouvelle instance de `Note` est ajoutée à la liste, puis l'affichage est mis à jour via l'appel à `notifyDataSetChanged`. Ce mécanisme illustre la communication inter-activités dans Android.

```
@Override
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == 1 && resultCode == RESULT_OK) {
        notes.add(new Note(
            data.getStringExtra("nom"),
            data.getStringExtra("desc"),
            data.getStringExtra("date"),
            data.getStringExtra("prio")
        ));
        adapter.notifyDataSetChanged();
    }
}
```

Listing 2 – Récupération des données après ajout

3.2 AddNoteActivity – Ajout de notes

La `AddNoteActivity` permet la création d'une nouvelle note à l'aide d'un formulaire complet et ergonomique. L'interface est organisée dans un `ScrollView` afin de s'adapter aux différentes tailles d'écran. Cette activité gère la saisie et la validation des données, l'ajout d'images via la caméra ou la galerie, ainsi que l'utilisation de l'API moderne `ActivityResultLauncher`. Une fois la saisie terminée, les données sont renvoyées à l'activité principale à l'aide d'un `Intent` de résultat.

3.2.1 Composants du formulaire

Le formulaire repose sur plusieurs composants Android : des `EditText` pour le titre, la description et la date, un `Spinner` pour le choix de la priorité, et une `ImageView` pour l'aperçu de l'image associée à la note. Des boutons dédiés permettent l'accès à la caméra, à la galerie et la validation finale du formulaire.

3.2.2 Gestion de la caméra

L'accès à la caméra nécessite une vérification préalable des permissions. Une fois l'autorisation accordée, un URI est créé via le `MediaStore` pour stocker l'image capturée. L'application lance ensuite un `Intent` implicite avec l'action `ACTION_IMAGE_CAPTURE`, dont le résultat est traité de manière asynchrone.

```
private void openCamera() {
    if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.CAMERA)
            != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(
            this,
            new String[]{Manifest.permission.CAMERA},
            101);
        return;
    }

    ContentValues values = new ContentValues();
    values.put(MediaStore.Images.Media.TITLE, "New Picture");

    imageUri = getContentResolver().insert(
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        values
    );

    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
    cameraLauncher.launch(intent);
}
```

Listing 3 – Ouverture de la caméra

3.2.3 ActivityResultLauncher

Les `ActivityResultLauncher` remplacent la méthode dépréciée `startActivityForResult`. Ils offrent une gestion plus claire et sécurisée des résultats d'activités. Dans ce projet, ils sont utilisés pour récupérer les images provenant de la caméra ou de la galerie et les afficher immédiatement dans l'interface.

```
cameraLauncher = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    result -> {
        if (result.getResultCode() == RESULT_OK && imageUri != null) {
            imgPhoto.setImageURI(imageUri);
        }
    }
);
```

Listing 4 – Gestion des résultats avec ActivityResultLauncher

3.3 DetailsNoteActivity – Consultation détaillée

La `DetailsNoteActivity` est dédiée à l'affichage détaillé d'une note sélectionnée. Elle se concentre uniquement sur la présentation des informations, dans une interface claire et lisible, sans logique métier complexe. Les données sont reçues via un `Intent`, rendant l'activité indépendante et simple à maintenir.

3.3.1 Code source

L'implémentation de cette activité est volontairement minimale. Les informations de la note sont récupérées depuis l'Intent et affichées directement dans les `TextView` correspondants. Un bouton permet de revenir à l'écran précédent en appelant la méthode `finish()`.

```
public class DetailsNoteActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details_note);

        ((TextView) findViewById(R.id.txtNom))
            .setText(getIntent().getStringExtra("nom"));
        ((TextView) findViewById(R.id.txtDesc))
            .setText(getIntent().getStringExtra("desc"));
        ((TextView) findViewById(R.id.txtDate))
            .setText(getIntent().getStringExtra("date"));
        ((TextView) findViewById(R.id.txtPrio))
            .setText(getIntent().getStringExtra("prio"));

        findViewById(R.id.btnRetour)
            .setOnClickListener(v -> finish());
    }
}
```

Listing 5 – DetailsNoteActivity.java

4 Adaptateur personnalisé

4.1 NoteAdapter

Le `NoteAdapter` assure la liaison entre les objets `Note` et leur affichage dans la `ListView`. Il hérite de `ArrayAdapter` et permet de personnaliser la représentation visuelle de chaque note. L'adaptateur gère également le recyclage des vues, un mécanisme clé d'optimisation qui améliore les performances et la fluidité du défilement.

4.1.1 Fonctionnalités

L'adaptateur affiche uniquement les informations essentielles de chaque note, à savoir le titre et la date, afin de conserver une liste claire et lisible. Il applique également un code couleur en fonction de la priorité de la note, facilitant l'identification rapide des éléments importants. Les priorités haute, moyenne et basse sont respectivement associées à des couleurs rouge clair, jaune et vert clair.

L'implémentation exploite la réutilisation de `convertView`, conformément aux bonnes pratiques Android, afin de limiter les opérations coûteuses d'inflation de layouts et d'optimiser les performances lors du défilement.

```
public class NoteAdapter extends ArrayAdapter<Note> {
    public NoteAdapter(Context context, List<Note> notes) {
        super(context, 0, notes);
    }
    @Override
    public View getView(int position, View convertView,
                        ViewGroup parent) {
        if (convertView == null) {
            convertView = LayoutInflater.from(getContext())
                .inflate(R.layout.item_note, parent, false);
        }
        Note note = getItem(position);
        ((TextView) convertView.findViewById(R.id.txtNom))
            .setText(note.getNom());
        ((TextView) convertView.findViewById(R.id.txtDate))
            .setText(note.getDate());
        switch (note.getPriorite()) {
            case "Haute":
                convertView.setBackgroundColor(
                    Color.parseColor("#FFCDD2"));
                break;
            case "Moyenne":
                convertView.setBackgroundColor(
                    Color.parseColor("#FFF9C4"));
                break;
            default:
                convertView.setBackgroundColor(
                    Color.parseColor("#C8E6C9"));
        }
        return convertView;
    }}
```

Listing 6 – NoteAdapter.java – Méthode getView

5 Concepts Android utilisés

5.1 Intents

Les **Intents** constituent le mécanisme principal de communication entre les différentes composantes de l'application Android. Ils permettent de lancer des activités, de transmettre des données et d'interagir avec des applications ou services du système.

— **Intent explicite** : Utilisé pour la navigation interne entre les activités de l'application, par exemple pour passer de la liste des notes à l'écran d'ajout ou de consultation détaillée.

— **Intent implicite** : Employé pour solliciter des fonctionnalités du système Android, telles que l'ouverture de la caméra ou de la galerie, sans connaître à l'avance l'application qui traitera la requête.

— **putExtra() / getExtra()** : Méthodes utilisées pour transmettre des données simples (chaînes de caractères, dates, priorités) entre activités via les Intents.

— **startActivityForResult()** : Mécanisme permettant de lancer une activité en attente d'un résultat, notamment lors de l'ajout d'une nouvelle note (remplacé partiellement par l'API moderne ActivityResultLauncher).

5.2 Cycle de vie des activités

Le respect du cycle de vie des activités est essentiel pour garantir la stabilité et la performance de l'application. Chaque activité suit une succession d'états gérés par le système Android.

— `onCreate()` : Méthode appelée lors de la création de l'activité. Elle est utilisée pour initialiser l'interface utilisateur, les composants graphiques et les structures de données.

— `onActivityResult()` : Méthode invoquée lors du retour d'une activité appelée avec attente de résultat. Elle permet de récupérer les données saisies par l'utilisateur et de mettre à jour l'interface.

— `finish()` : Méthode permettant de fermer proprement une activité et de revenir à l'activité précédente dans la pile de navigation.

5.3 ActivityResultLauncher

L'API **ActivityResultLauncher** représente l'approche moderne recommandée par Google pour gérer le lancement d'activités et la récupération de leurs résultats.

— Remplace la méthode dépréciée `startActivityForResult()`, améliorant la lisibilité et la sécurité du code.

— Permet une meilleure gestion des permissions et des résultats asynchrones.

— Offre un code plus structuré, déclaratif et *type-safe*, réduisant les risques d'erreurs à l'exécution.

5.4 ListView et adaptateurs

L'affichage dynamique des notes repose sur le composant `ListView` associé à un adaptateur personnalisé.

— **ArrayAdapter** : Classe de base utilisée pour relier une liste d'objets `Note` à leur représentation visuelle.

— **BaseAdapter** : Alternative plus flexible permettant un contrôle avancé de l'affichage (non utilisée mais étudiée).

— **ViewHolder pattern** : Technique d'optimisation qui limite les appels répétitifs à `findViewById`, améliorant les performances lors du défilement.

— **notifyDataSetChanged()** : Méthode permettant de rafraîchir dynamiquement l'affichage de la liste après l'ajout ou la modification d'une note.

6 Conclusion

Ce travail pratique a permis de concevoir et de réaliser une application Android fonctionnelle de gestion de notes, répondant pleinement aux exigences du cahier des charges tout en intégrant les principes fondamentaux du développement mobile. Le projet *MyNotes* illustre une maîtrise globale de l'écosystème Android, depuis la création d'interfaces utilisateur intuitives et responsives jusqu'à la gestion du cycle de vie des activités, la navigation par Intents et l'intégration de fonctionnalités système telles que la caméra et la galerie. Il a également mis en évidence l'importance d'une architecture claire et modulaire, favorisant la maintenabilité du code, la lisibilité et le respect des bonnes pratiques de développement.

Au-delà des compétences techniques acquises, ce projet a constitué une expérience d'apprentissage enrichissante, permettant de mieux comprendre les contraintes spécifiques du développement mobile, notamment la gestion des permissions, de la mémoire et des interruptions système. Les difficultés rencontrées ont renforcé nos capacités d'analyse, de débogage et de recherche de solutions adaptées. L'application développée représente une base solide pouvant évoluer vers un gestionnaire de notes plus avancé intégrant la persistance des données, une architecture moderne comme MVVM et de nouvelles fonctionnalités orientées productivité. Ce travail pratique a ainsi renforcé notre motivation et notre confiance pour aborder des projets Android plus complexes et ambitieux.