

Projet Développement mobile : Planee

BERJOLA MATTHIAS 37000961 - ADOLPHE BENJAMIN 37001213

15 mai 2020

TABLE DES MATIÈRES

1	Introduction	2
2	L'application Planee	3
2.1	Principe	3
2.2	Architecture de l'application	3
2.2.1	MainActivity	3
2.2.2	AddActivity	4
2.2.3	DetailsActivty	5
2.2.4	UpdateActivity	6
2.2.5	Bases de données	7
2.3	Problèmes rencontrés	8
2.3.1	Fragments	8
2.3.2	Notifications	10
2.3.3	Layout	11
2.4	Amélioration futures	11
3	Conclusion	12
4	Bibliographie	13

CHAPITRE

1

INTRODUCTION

Cette année, afin de valider l'UE développement mobile 2, nous avons dû réaliser un projet. Ce projet consistait à réaliser une application ou un jeu respectant certaines contraintes :

- Le choix de l'application est laissé libre, mais une partie de la notation portera sur sa complexité (quelque chose de trop simpliste ne donnera pas lieu à beaucoup de points).
- L'application doit proposer : plusieurs écrans, une présentation sous forme de liste, une sauvegarde persistante.
- L'application doit fonctionner correctement sur tout type d'écran (grand, petit...), en mode portrait et paysage : ressources alternatives sous Android + contraintes sur les composants graphiques.

De ce fait mon collègue et moi même avons opté pour une application. Après un BrainStorming intensif, nous avons décidé d'appeler notre application Planee et nous expliquerons son fonctionnement par la suite.

CHAPITRE

2

L'APPLICATION PLANEE

2.1 Principe

Planee est une application de gestion d'évènements. Elle permettra à l'utilisateur d'organiser et créer ses différents évènements. En effet, chaque évènement est composé d'un titre, d'une date limite de tâches et chaque tâche est composée d'un titre, d'un nom de magasin et éventuellement d'une URL indiquant le site du magasin si l'utilisateur a besoin de passer une commande. Sur le long terme l'application devrait supposer des magasins ou différents sites pour passer commande.

2.2 Architecture de l'application

L'application Planee est structurée en 3 Activités :

- L'activité MainActivity représente la page d'Accueil.
- L'activité AddActivity représente la page d'ajout d'un évènement.
- L'activité DetailActivity représente la page de détails d'un évènement.
- L'activité UpdateActivity représente la page de modification d'un évènement.

2.2.1 MainActivity

Comme dit précédemment la MainActivity représente la page d'accueil. En effet, la MainActivity est composée principalement d'un élément ListView. La ListView permettra de lister les différents évènements. Les évènements sont récupérés dans la Base de Données locale du téléphone, si aucun évènement est présent dans la Base de Donnée alors un message apparaîtra à l'écran.

Quelques fonctionnalités de la page d'accueil :

- Appuie long sur un évènement pour le supprimer
- Appuie sur un évènement pour avoir les détails suivants :
 - Nom de l'évènement

- Date de l'évènement
- Liste des tâches de l'évènement
- Appuie sur le bouton retour pour quitter l'application

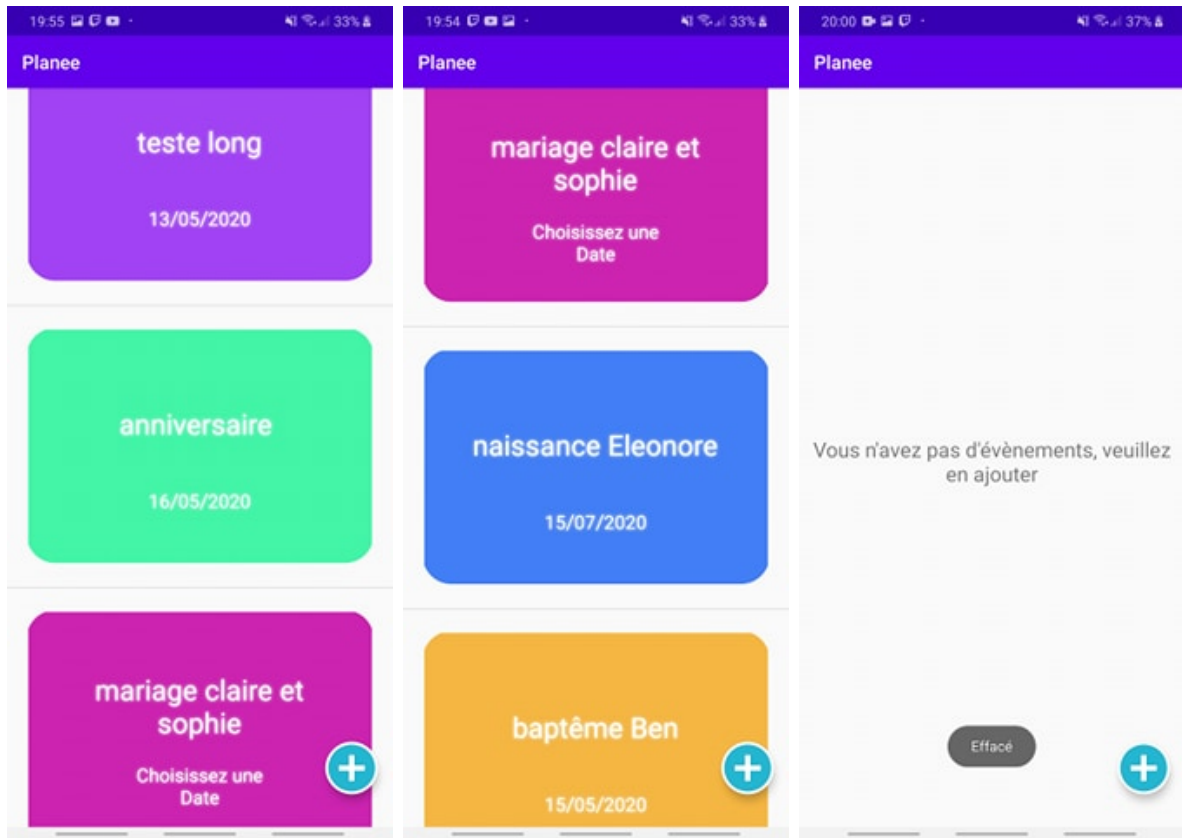


FIGURE 2.1 – Page d'accueil avec et sans évènements

2.2.2 AddActivity

La page contient un formulaire permettant d'entrer le nom, la date limite de l'évènement, l'heure et l'ensemble des tâches. Le formulaire d'ajout de tâches est dynamique grâce au bouton "Nouvelle tâche".

Quelques fonctionnalités du formulaire :

- Appuie sur le bouton "Nouvelle tâche" afin d'ajouter un champs de saisie d'une nouvelle tâches, ces champs sont composées de 3 champs de saisie de textes :
 - Nom de la tâche
 - Potentiellement le nom du magasin
 - Potentiellement l'URL du site du magasin
- Appuie sur le bouton "Ajout de l'évènement" afin d'ajouter l'évènement dans la base de données locale de l'appareil et déclencher une alarme à la date et l'heure entrées par l'utilisateur (Voir partie problèmes rencontrés)



FIGURE 2.2 – Page du formulaire d'ajout avec et sans tâches

2.2.3 DetailsActivity

Comme dit dans la partie concernant la MainActivity, lors d'un appui sur un évènement on accède à la page de détails de cet évènement. Ainsi, on va récupérer tout les éléments dans la Base de données locale en rapport avec cet évènement. Quelques fonctionnalités de la page :

- Appuie sur une tache afin de la supprimer (Tache complétée)
- Appuie sur le bouton flottant afin de modifier l'évènement courant

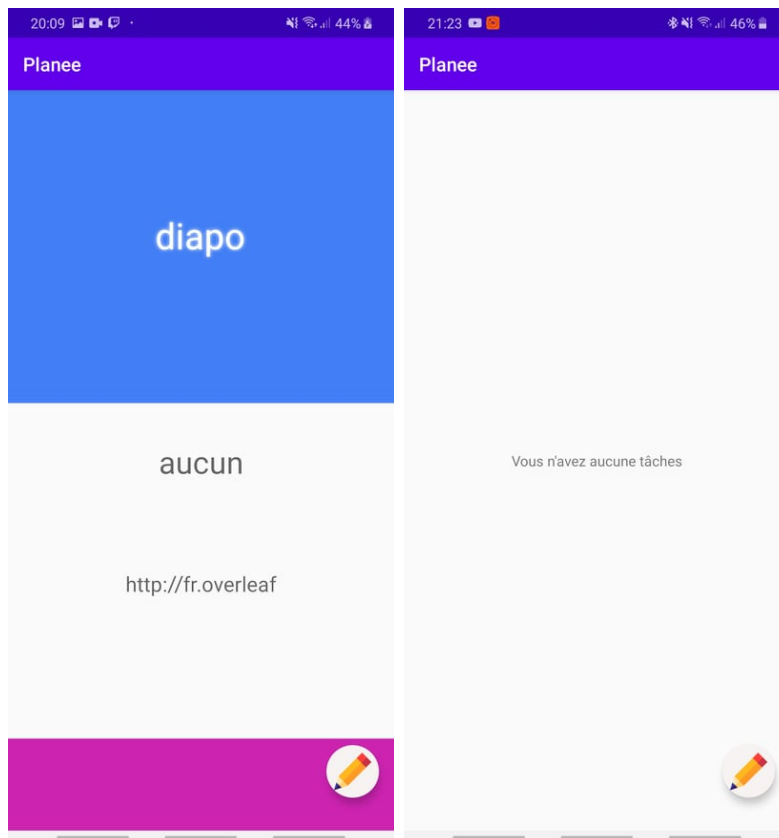


FIGURE 2.3 – Formulaire de mise à jour avec et sans modification(s)

2.2.4 UpdateActivity

Le layout présent dans UpdateActivity est le même que celui présent dans AddActivity. En effet, la seule différence entre ces deux activités est que l'un est fait pour mettre à jour un évènement l'autre pour ajouter. Ainsi, le formulaire de mise à jour sera pré-remplis par les informations de l'évènement voulu par l'utilisateur. À partir de ce formulaire, l'utilisateur pourra mettre à jour les données existantes mais aussi ajouter de nouvelles tâches.

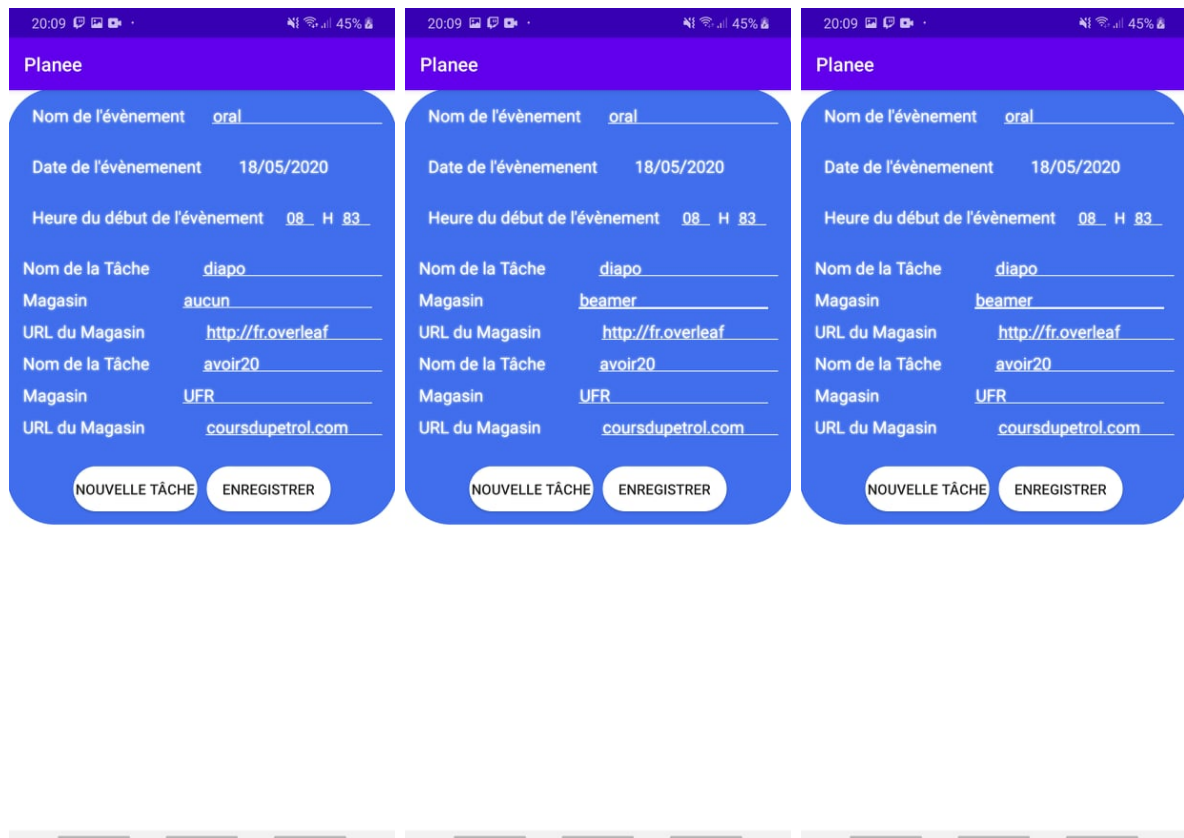


FIGURE 2.4 – Formulaire de mise à jour avec et sans modification(s)

2.2.5 Bases de données

Pour ce projet, nous avons utilisé une base de données composée de 2 tables :

- Une table Event_Table
- Une table Tache_Table

La table Event_Table sert à stocker les différents évènements créés. À l'intérieur on y trouve :

- l'id de l'évènement
- Le nom de l'évènement
- Sa date limite (La date où l'évènement se produira)
- L'heure limite (L'heure à laquelle l'évènement se déroulera)

La table Tache_Table sert quand à elle à stocker les différentes tâches à effectuer afin de préparer l'évènement convenablement. Elle est composée de :

- L'id de la tâche
- Du nom du magasin dans lequel la tâche doit être effectuée
- De l'URL du magasin si l'achat peut être effectué en ligne
- IdEvent qui est une clé étrangère permettant de savoir à quel évènement appartient la tâche

On trouve également deux méthodes indispensables à la base de données :

- La méthode InsertEvent
- La méthode getAllEvent

La première méthode a été créée dans le but de pouvoir insérer chaque événement (composé de tâches) dans la base de données. La seconde méthode permet de récupérer tous les événements créés par l'utilisateur afin de pouvoir les afficher de nouveau lorsque l'utilisateur revient sur l'application après l'avoir fermée.

Le choix d'une base données a été fait dans le but de garantir la persistance des données de l'utilisateur à chaque fois que l'application est quittée puis ré-ouverte.

SQLITE EXPLORER									
▼ Planee.db					#	id	Nom	date_limite	heure
> android_metadata					1	9	test	10/06/2020	15h59
> table_evenement					2	10	test 2	12/08/2020	15H00
> table_tache					3	11	test 3	12/05/2023	15H00
					4	12	test 4	12/07/2020	15H02

#	id	Nom	magasin	url_magasin	id_event
1	6	fifbb	rkadjk	didj	9
2	7	dbdidj	fifk	fifkk	10
3	8	fifdj	rifjif	rifjfn	10
4	9	dhdu	didib	didi	11

FIGURE 2.5 – Schéma de la Base de données

2.3 Problèmes rencontrés

Au cours de ce projet, nous avons rencontré plusieurs problèmes. Nous allons donc citer dans ce rapports les différents problèmes que nous avons rencontré.

2.3.1 Fragments

Au début de ce projet, lors du choix de la première activité, nous avons choisi l'activité "Navigation Drawer Activity".

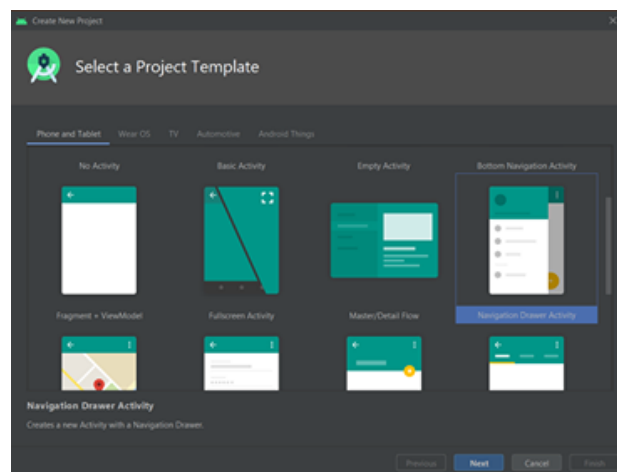


FIGURE 2.6 – Template

Cependant, ce template est basé sur l'utilisation des Fragments, une notion qui n'a pas été vue en cours. Ainsi nous avons suivi de nombreux tutoriaux concernant les Fragments. Voici une liste des différents problèmes rencontrés lors de l'utilisation de Fragment :

- Dès lors que l'on a voulu changer de page, le premier problème est apparu. En effet, lors de l'utilisation du FragmentManager et des transitions afin de changer les Fragments ils avaient tendance à se superposer.

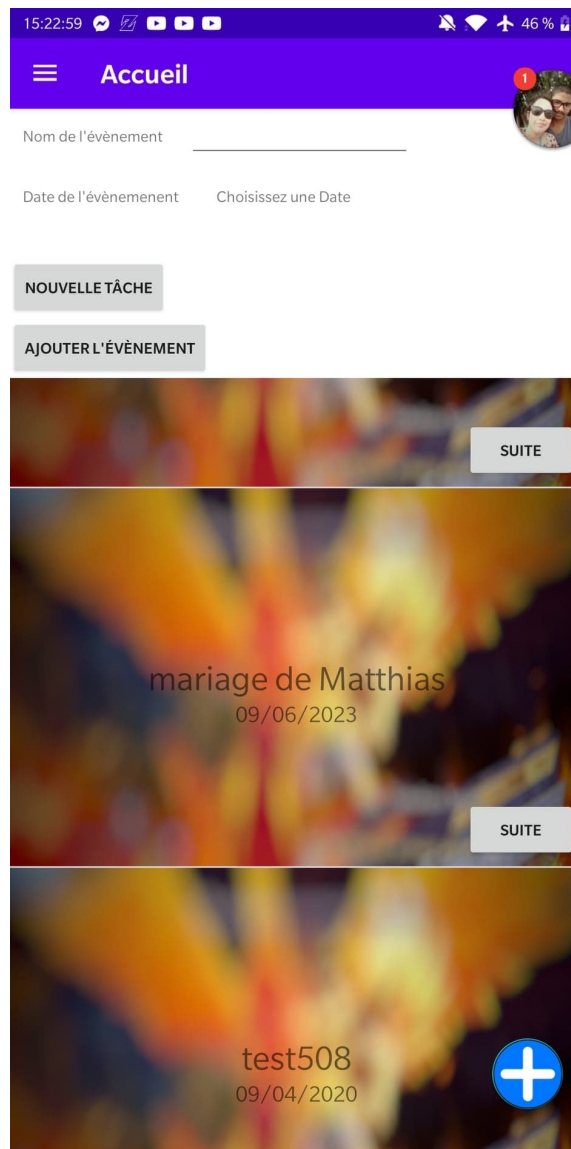


FIGURE 2.7 – Template

Nous avons pu résoudre ce problème en vérifiant dans le conteneur si un fragment était présent et si c'était le cas on enlevait tout fragment du conteneur grâce au code suivant :

```
if (container != null) {
    container.removeAllViews();
}
```

- Nous devons gérer l'appuie sur le bouton retour de l'utilisateur. En effet, lorsque l'utilisateur appuyait sur la touche retour il se retrouvait devant une interface blanche car nous vidions le conteneur à chaque fois. Ainsi dans chaque fragment nous avons du gérer de façon programmatique l'appui sur le bouton retour sachant que la méthode "OnBackPressed" ne fonctionnait pas dans le cas de Fragments :

```
// This callback will only be called when
// MyFragment is at least Started.
OnBackPressedCallback callback =
    new OnBackPressedCallback(true /* enabled by default */) {
        @Override
        public void handleOnBackPressed() {
            getFragmentManager().beginTransaction()
                .replace(R.id.DetailsFragment, new HomeFragment())
```

```

        .addToBackStack(null)
        .commit();
    }
};
requireActivity().getOnBackPressedDispatcher()
    .addCallback(this, callback);

```

- Le problème le plus important que nous avons rencontré était le temps. En effet, se former sur les Fragments et tout ce qui touchait aux Fragments demandaient énormément de temps. Plus de temps nous passions à nous former moins de temps nous avions pour terminer l'application. Ainsi après 3 séances supplémentaires avec le responsable de cet UE et de nombreuses galères dans la programmation avec les Fragments, nous avons décidé de repasser dans un format classique en privilégiant les activités aux Fragments malgré l'économie d'énergie apporté par les Fragments en utilisant qu'une seule activité pour tout gérer.

2.3.2 Notifications

Lors de la réalisation du code concernant l'envoi de notifications de l'application afin de prévenir l'utilisateur le jour d'un évènement. Ainsi, pour créer la notification nous sommes passé par le code suivant :

```

Intent intent = new Intent(AddActivity.this, ReminderBroadcast.class);
intent.putExtra("IdEvent", myDbAdapter.getEventID(evenement.getNom(), evenement.getDateLimite()));
PendingIntent pendingIntent = PendingIntent.getBroadcast(AddActivity.this, 0, intent, 0);
AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
long timeAtButtonClick = System.currentTimeMillis();
long tenSecondes = 1000 * 10;
alarmManager.set(AlarmManager.RTC_WAKEUP, timeAtButtonClick + tenSecondes, pendingIntent);

```

FIGURE 2.8 – Code de mise en place de l'alarme

Le but de ce code est de créer une alarme pour la date de l'évènement afin d'envoyer une notification à l'utilisateur lorsque l'utilisateur ajoute un évènement à la base de données. Pour la création de cette notification, nous avons utilisé le code suivant :

```

public class ReminderBroadcast extends BroadcastReceiver {
    private MyDBAdapter myDb;

    @Override
    public void onReceive(Context context, Intent intent) {
        myDb = new MyDBAdapter(context);
        myDb.open();
        Evenement event = myDb.getEvent(intent.getLongExtra("IdEvent", 0));
        myDb.close();

        NotificationCompat.Builder builder = new NotificationCompat.Builder(context, "NotifyLemubit")
            .setSmallIcon(R.drawable.ikki)
            .setContentTitle("Planee")
            .setContentText("C'est l'heure de " + event.getNom())
            .setPriority(NotificationCompat.PRIORITY_DEFAULT);
        Log.i("Notification----", "" + event.getId());
        Log.i("Notification----", event.getNom());
        NotificationManagerCompat notificationManager = NotificationManagerCompat.from(context);
        notificationManager.notify(200, builder.build());
    }
}

```

FIGURE 2.9 – Code de mise en place de la notification

Pour créer la notification, nous récupérons l'évènement récemment créé grâce à la méthode `getEvent` de `MyDBAdapter` et ensuite créer la notification à l'aide du `Notification.Builder` et du

NotificationManagerCompat. Cependant, nous avons rencontré un problème qui, aujourd'hui, n'est toujours pas résolu. Pour les tests nous avons créé l'alarme pour un déclenchement dans 10 secondes. Lors de la création de 2 événements, le texte de la notification ne se met pas à jour. La notification conserve les éléments de la 1^{ère} notification. Quelques fois, les informations sont null.



FIGURE 2.10 – Problème : Notifications sur 2 événements

2.3.3 Layout

Lors de la réalisation du design de l'application nous devons gérer en plus des éléments de bases (couleurs formes etc) les layouts. Deux problèmes revenaient souvent : La conservation de l'échelle lorsque l'on passe à un appareil plus petit et la superposition des éléments lors de ce passage.

Au départ le design de l'application était réalisé entièrement avec des ConstraintLayout. Le problème avec les Constraint Layout est le manque de précision et de rigidité dans les contraintes que nous mettons. De ce fait des éléments qui semblaient bien placés se retrouvaient déplacés sans aucune intervention de notre part lorsque nous lançons l'application. Nous avons donc pris la décision de remplacer les Constraint Layout par des Relative Layout là où cela posait problème (page détail, page addDétail notamment). Après ce changement nous avons pu convenablement placer les différents éléments sans erreurs apparentes.

2.4 Amélioration futures

CHAPITRE

3

CONCLUSION

CHAPITRE

4

BIBLIOGRAPHIE