MultiPlay - Rapport de projet de développement informatique - PRO3600

Nessim OUSSEDIK Raouf ALIOUAT Amine BERRAHO
Sylvain MENEZ
Encadrante : Amel Bouzeghoub

12 février 2019

Table des matières

Partie 1

Introduction

Ce document, produit avec LaTeX, porte sur le développement d'une application Android permettant un usage multifenêtres qui s'inscrit dans le cadre du projet de développement informatique PRO3600. Il constitue une mise à jour de notre pré-rapport de projet.

Il est notamment divisé en deux parties. D'abord le cahier des charges présente le projet et ses attentes. Puis nous nous attardons sur les objectifs, les phases et démarches entreprises dans le cadre du travail de développement.

Le choix d'un projet de développement sur mobile nous donne une première expérience dans un domaine en vogue qui est un excellent prétexte pour mettre en œuvre les connaissances acquises en JAVA durant le module CSC3102.

Partie 2

Cahier des charges

Nous utilisons désormais nos smartphones de la même manière que nos PC : une des caractéristiques est l'usage simultané de plusieurs applications. Le multitâches a été introduit par Android 8.0. Cela ne permet néanmoins pas d'effectuer ces taches simultanément, possibilité présente dans tous les systèmes d'exploitation grâce à l'affichage fractionné. Les développeurs de Google ont rajouté cette possibilité pour les développeurs à partir de l'API 26 d'Android, n'incluant donc que les appareils avec une version d'Android supérieure à 6.0, représentant 57,7 % de la population à la date du 5 Février.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.6%
4.3		18	0.7%
4.4	KitKat	19	12.0%
5.0	Lollipop	21	5.4%
5.1		22	19.2%
6.0	Marshmallow	23	28.1%
7.0	Nougat	24	22.3%
7.1		25	6.2%
8.0	Oreo	26	0.8%
8.1		27	0.3%

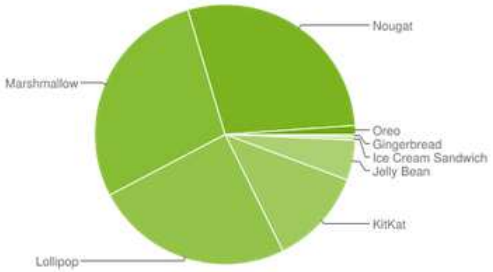

FIGURE 2.1 – Répartition des terminaux Android en fonction des distributions.

Cependant, une grande partie de ces 57,7 % concerne en réalité des tablettes sous Android ; la majorité des smartphones n'ont pas accès au mode multifenêtres.

L'idée de notre projet naît donc de ce besoin, et vise à permettre un affichage fractionné à la majorité des terminaux Android. Plutôt que de créer une application regroupant les applications déjà installées dans un affichage multifenêtres avec la dernière API de Google, l'objectif est de développer une application à part entière disposant de plusieurs modules simultanément accessibles par l'utilisateur.

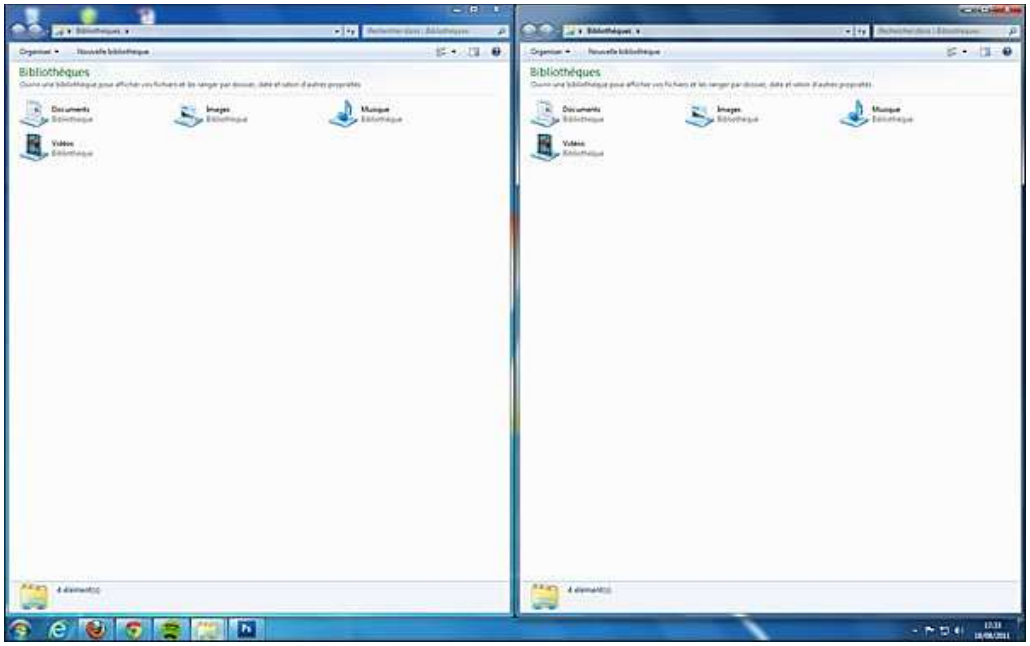

FIGURE 2.2 – Affichage fractionné sur Windows.

Nous souhaitons obtenir une utilisation similaire à la figure 2.2 . Pour cela 2 solutions s'offrent à nous : afficher simultanément les modules dans une même fenêtre, ou développer une application contenant affichant les modules sous formes de bulles, à l'image de l'application Messenger.

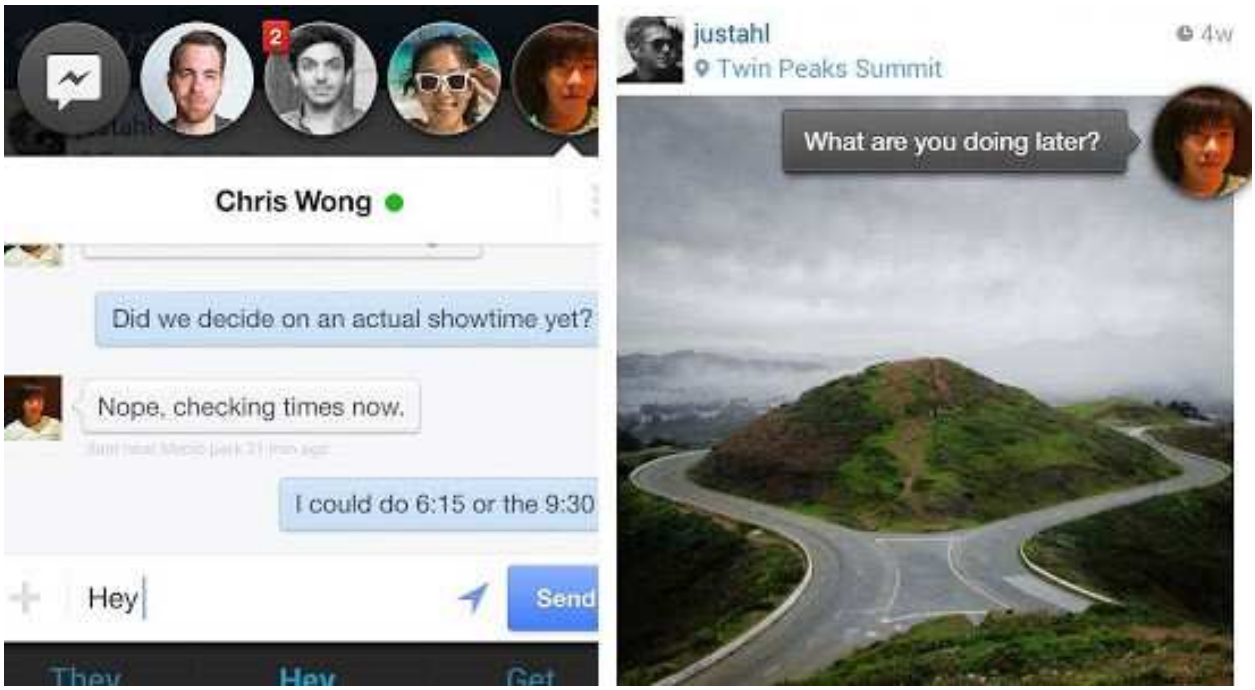

FIGURE 2.3 – Application Messenger utilisant le système de bulles.

Pour l'instant, notre choix se porte sur la première solution, mais nous n'excluons pas le système de bulles si le besoin se présente.

Partie 3

Développement

Cette partie est divisée en deux sections : la première est dédiée à l'analyse du problème et à la spécification fonctionnelle ; la seconde traite de la conception préliminaire de notre application. Afin de planifier les taches et communiquer entre nous, nous utiliserons le site web « Trello ». Nous utiliserons un serveur SVN pour stocker les différentes versions du code source. Enfin, nous communiquerons entre nous via l'application Messenger, le groupe étant déjà créé.

3.1 Analyse du problème et spécification fonctionnelle

L'objet de cette section est de définir les spécifications fonctionnelles détaillées de notre application. Nous décrivons précisément l'ensemble des fonctionnalités de l'application, les objets manipulés, leurs buts et leurs principes de fonctionnement, ainsi que les écrans utilisateurs mettant en œuvre les fonctionnalités de l'application.

La principale fonctionnalité développée par l'application est de proposer à l'utilisateur, conformément au cahier des charges, une interface qui regroupe plusieurs modules, permettant l'utilisation de plusieurs services. Le cœur du développement de notre application est donc l'affichage fractionné. In fine, le développement de l'application est mené pour une utilisation sur des équipements tournant sous Android. Il peut s'agir de smartphones, de tablettes etc.

Dans le cadre fixé par le cahier des charges, l'objectif retenu est celui d'une application fonctionnelle disposant de deux ou trois modules différents. Ainsi, nous souhaitons permettre à chacun de pouvoir regarder une vidéo YouTube tout en répondant à un SMS et en prenant des notes. Parmi ces trois modules développés, l'utilisateur aura la possibilité de fractionner l'affichage de sa machine en sélectionnant deux fonctionnalités.

Dans la suite, nous dressons la spécification fonctionnelle de chacun de ces trois modules, ainsi que la manière dont ils doivent s'agencer entre eux.

Module YouTube

- Barre de recherche pour accéder à une vidéo : l'utilisateur doit retrouver la simplicité de navigation que l'hébergeur de vidéos américain entend offrir.
- Lancement de la vidéo par appui sur un bouton de chargement.
- Possibilité de modifier la résolution de la vidéo afin d'obtenir la meilleure définition possible à taille d'images égale, par la multiplication des pixels.
- Impossibilité dans un premier temps de modifier le format d'affichage de la vidéo : le format retenu par défaut est le 16/9 (format large). Par ailleurs, la taille du module est définie comme étant proportionnelle à celle de l'écran.

- Possibilité de passer à la vidéo suivante (que ce soit par une suggestion ou par passage à l'item suivant dans la playlist).

- Impossibilité pour l'instant de visionner les commentaires relatifs au contenus vidéos.

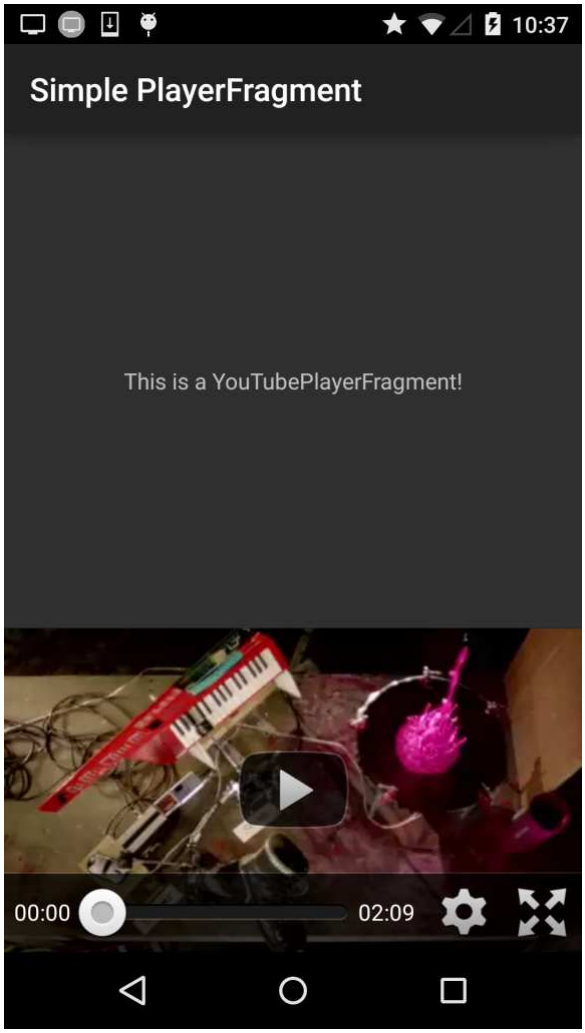

FIGURE 3.1 – Illustration du module YouTube au sein de l'interface.

Module Notes

- Possibilité de rédiger une note simple dans un premier temps, telle qu'un sticker (c'est à dire aucun formatage préalable).

- Possibilité de sauvegarder une note en tant que note courante (par exemple sticker qui restera visible sur l'interface en permanence jusqu'à ce que l'utilisateur le mette à la corbeille.

- Possibilité de sauvegarder une note en tant que note réutilisable pour plus tard.

- Édition d'une note : possibilité de modifier toute note préalablement enregistrée en mémoire par le module.

FIGURE 3.2 – Illustration du module Notes au sein de l'interface.

Module Messagerie

- Recherche simple des contacts parmi une liste de contacts définie par l'utilisateur. L'objectif est de fluidifier la recherche en elle-même.
- Dans un premier temps la fonctionnalité retenue consiste à pouvoir envoyer un message à un contact sélectionné depuis la liste de contacts.
- Obtenir une véritable vue d'ensemble d'une discussion instantanée avec un seul contact. Si nous en avons le temps, nous souhaitons obtenir un véritable historique de conversation.
- Si nous en avons le temps, nous gérerons les requêtes message de type MMS (Multimedia Messaging Service), qui permettent notamment de transmettre des photos, des enregistrements audio ainsi que de la vidéo.

Pour chacun des modules définis ci-dessus ainsi que pour l'agencement général de l'application, il convient de caractériser toutes les données relatives à l'utilisateur, c'est à dire le cadre d'utilisation de l'application par ce dernier. Nous présentons ainsi dans ce qui suit les différents critères et conditions d'utilisation, les modalités et règles d'utilisation, ainsi que les interactions relatives à l'utilisation de notre application.

L'utilisation de l'application requiert un lancement par l'utilisateur. Ce dernier doit disposer d'un appareil compatible avec la dernière version d'Android. Depuis son écran d'accueil, il lui faut se rendre dans l'onglet relatif aux applications installées, puis sélectionner simplement notre application pour la lancer.

Une nouvelle fenêtre apparait et doit attendre les requêtes de l'utilisateur. Concrètement, il doit sélectionner les modules avec lesquels il souhaite pouvoir interagir de façon simultanée. Un message l'invitant à le faire est affiché sur l'interface : " *Veuillez sélectionner vos différents modules pour l'affichage fractionné*". Un bouton de confirmation du choix de l'utilisateur pourra être envisagé dans le développement. Dans ce cas, l'utilisation de l'application requiert de l'utilisateur qu'il fasse glisser les modules qu'il souhaite dans une fenêtre prévue à cet effet. Par la suite, il pourra confirmer son choix et lancer l'affichage fractionné de l'écran. A ce stade, plusieurs cas d'utilisation sont à considérer, chacun d'entre eux menant à des tests de validation différents.

D'une part, si l'utilisateur ne sélectionne qu'un seul module, l'utilisation de notre application perd de son intérêt. L'interface graphique doit alors, par le biais d'un message d'erreur, inviter l'utilisateur à sélectionner au moins un autre module. D'autre part, no-

tons que l'utilisateur doit a priori être en mesure de sélectionner plusieurs fois le même module, avec des contraintes différentes selon les cas. Comme précisé par ailleurs, l'affichage final proposé est limité à trois modules différents. Si l'utilisateur sélectionne deux fois le module YouTube, s'il souhaite par exemple visionner deux vidéos en même temps, il n'aura alors pas la possibilité de sélectionner un troisième module. En effet, la taille et le format d'utilisation de ce module YouTube est limité et correspond à la moitié de l'écran. En revanche, si l'utilisateur souhaite envoyer plusieurs messages à des personnes différentes tout en visionnant un contenu vidéo, ou s'il souhaite gérer plusieurs notes relatives à un contenu vidéo qu'il visionne en temps réel, cela est rendu possible par l'interface de l'application.

Une fois ces différents critères vérifiés, les interactions entre l'utilisateur et l'interface prennent différentes formes. Au cours de l'utilisation, l'utilisateur doit avoir la possibilité, s'il le souhaite, de fermer un ou plusieurs modules et de les remplacer par d'autres. Par exemple, s'il a terminé de visionner une vidéo et qu'il souhaite envoyer un message parlant de cette dernière à un contact, il peut fermer le module YouTube, puis en revenant à l'interface de l'application en elle-même, sélectionner le module de messagerie pour effectuer un copier-coller des notes qu'il aura prises sur la vidéo grâce au module de prise de notes. Similairement, il peut, dans le cas particulier où il n'a sélectionné que deux modules sur trois, ouvrir un nouveau module sur l'écran. Là encore, ceci est limité par les contraintes relatives à l'affichage sur l'écran. Concrètement, si les deux modules initialement sélectionnés sont deux fenêtres YouTube, il ne pourra pas profiter pleinement de cette fonctionnalité.

Une question que l'on peut être amené à se poser au cours de son utilisation de l'application est celle de la gestion de la taille des modules et de leur manière de s'agencer sur l'écran. Hormis le module YouTube, l'utilisateur pourra gérer ces paramètres à sa guise. S'il souhaite positionner ses messages sur la partie supérieure de l'écran et ses notes sur la partie basse, il pourra le faire tout en réglant les paramètres de taille de chaque module. Dans le cas d'une division reposant sur trois modules dont un est le module YouTube, il pourra même gérer les positions relatives des deux autres modules, ceux-ci occupant alors la moitié de l'écran laissée vacante par YouTube. L'interface visuelle de l'application pourra proposer à l'utilisateur de saisir ces paramètres au moment de son choix de modules à insérer à l'écran.

Intéressons nous à présent à la question de l'affichage de notifications à l'écran. Cela concerne par exemple les notifications émises par d'autres applications, c'est à dire différentes de celles prises en charge par notre application. On peut notamment penser aux bulles de messages Messenger ou à des notifications issues d'applications d'actualité, journaux ou autre. Notre application doit être capable de gérer ces phénomènes en les adaptant à l'affichage courant de l'écran. Concrètement, l'utilisateur peut sélectionner s'il souhaite recevoir des notifications au cours de l'affichage fractionné. Cas échéant, on pourra faire en sorte, si le temps ne manque pas, que l'affichage généré par le module YouTube ne soit pas perturbé par une notification visuelle. Si l'utilisateur met en œuvre un affichage fractionné de deux modules YouTube, les notifications pourront se limiter à des notifications auditives. Dans les autres cas, l'affichage de notifications sera celui géré par défaut par la machine de l'utilisateur.

Enfin, l'utilisateur doit savoir comment mettre fin à l'affichage fractionné mis en œuvre par notre application. Il peut le faire en fermant successivement les fenêtres de chacun des modules qu'il aura décidé d'ouvrir, puis en fermant et en désactivant l'application de gestion. De cette façon, il retrouvera l'affichage d'interface normal de son équipement.

3.2 Conception préliminaire

Le but est de créer une interface fonctionnelle et simple d'utilisation. Pour ce faire, nous avons donc choisi de décomposer l'application en trois modules (module YouTube, Messagerie et Notes) que nous développerons de manière indépendante puis nous chercherons à les regrouper de façon à avoir une interface graphique la plus minimaliste et fonctionnelle possible c'est-à-dire en ciblant avant tout les usages les plus fréquents et en prenant en considération les nombreuses contraintes qui peuvent se présenter telles que l'agencement des trois modules sur une même fenêtre qui doit être optimal et doit s'adapter à la taille de l'écran etc. Si le temps nous le permet, nous pourrons envisager de personnaliser l'interface en choisissant les couleurs ou la taille d'affichage de chaque écran. Aussi, nous pourrons inclure d'autres modules à notre application.

Les fonctions incluses dans chaque module sont illustrées dans la figure suivante.

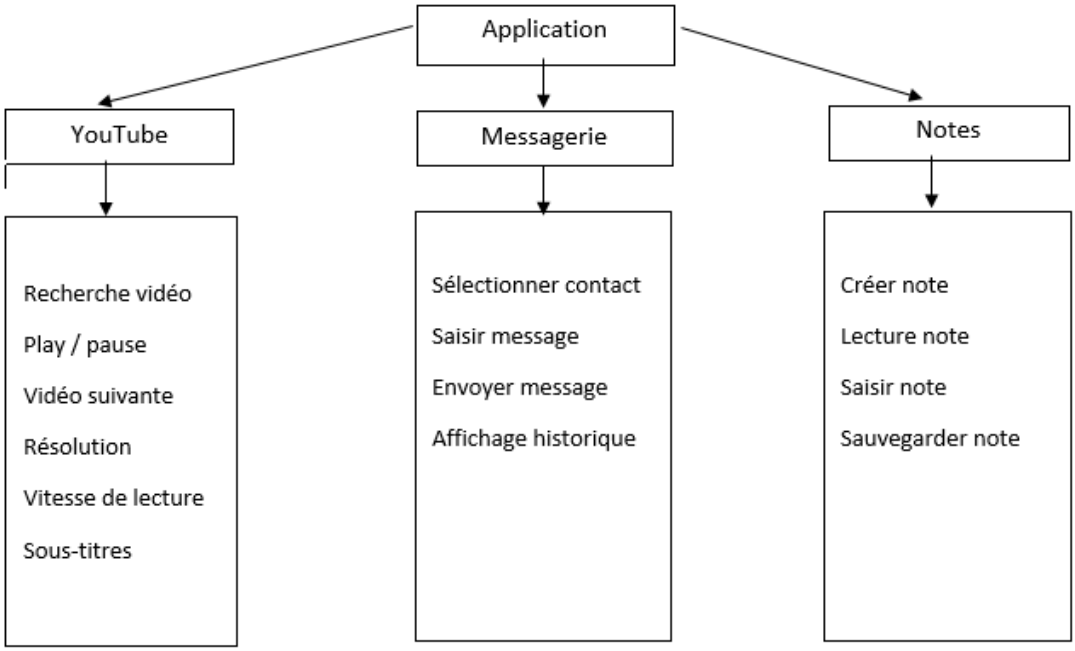

FIGURE 3.3 – Graphe représentant les dépendances de chaque module avec leurs fonctionnalités.

Nous importerons sur l'interface de développement Android Studio, des classes spécifiques à chaque module et qui incluent plusieurs méthodes. Notamment pour le module :

- YouTube : la classe *YouTubePlayerFragment* qui permet de lire des vidéos YouTube ;
- Messagerie : la classe *SmsManager* qui gère les opérations SMS telles que l'envoi du contenu texte du message et la classe *Telephony* qui contient les données liées au fonctionnement du téléphone, en particulier les messages SMS et MMS.

Un autre point important est qu'il faut donner des permissions pour pouvoir utiliser proprement les modules. En particulier pour le module Messagerie les permissions : BROADCAST_ SMS qui permet à notre application de recevoir directement les messages SMS entrants, BROADCAST_ WAP_ PUSH qui permet de recevoir les messages MMS entrants, READ_ CONTACTS qui permet de faire une recherche sur la liste des contacts. Toutes ces permissions sont disponibles dans la classe *Manifest.permission*. On demandera alors l'accord de l'utilisateur via un message qui s'affichera sur l'interface de type : " *L'application souhaite accéder à vos contacts. Accepter / Refuser* ".

3.3 Conception détaillée

Souhaitant que l'utilisateur puisse utiliser trois « modules » simultanément, nous avons décidé d'utiliser des fragments. Un fragment est une interface utilisateur ou une partie d'une interface. Nous décidons alors de combiner trois fragments dans l'activité associée à notre application. Une activité a un cycle de vie, et chaque fragment a sa propre vue et également son propre cycle de vie. Ainsi, les fragments sont des éléments indépendants et réutilisables et représentent donc une solution intéressante pour notre problème : un fragment équivaut à un « module ». Un fragment est destiné à YouTube, un autre à la messagerie et un dernier pour les notes.

Un fragment peut être statique ou dynamique. La différence réside dans le fait qu'un fragment statique ne peut pas être ajouté, remplacé ou supprimé dynamiquement durant l'exécution de l'activité. Dans un premier temps, nous décidons d'utiliser des fragments statiques en vue d'obtenir un noyau fonctionnel. Puis, nous envisageons de rendre ces fragments dynamiques, ce qui permettrait notamment à l'utilisateur d'ajouter ou de supprimer un module en fonction de ses besoins.

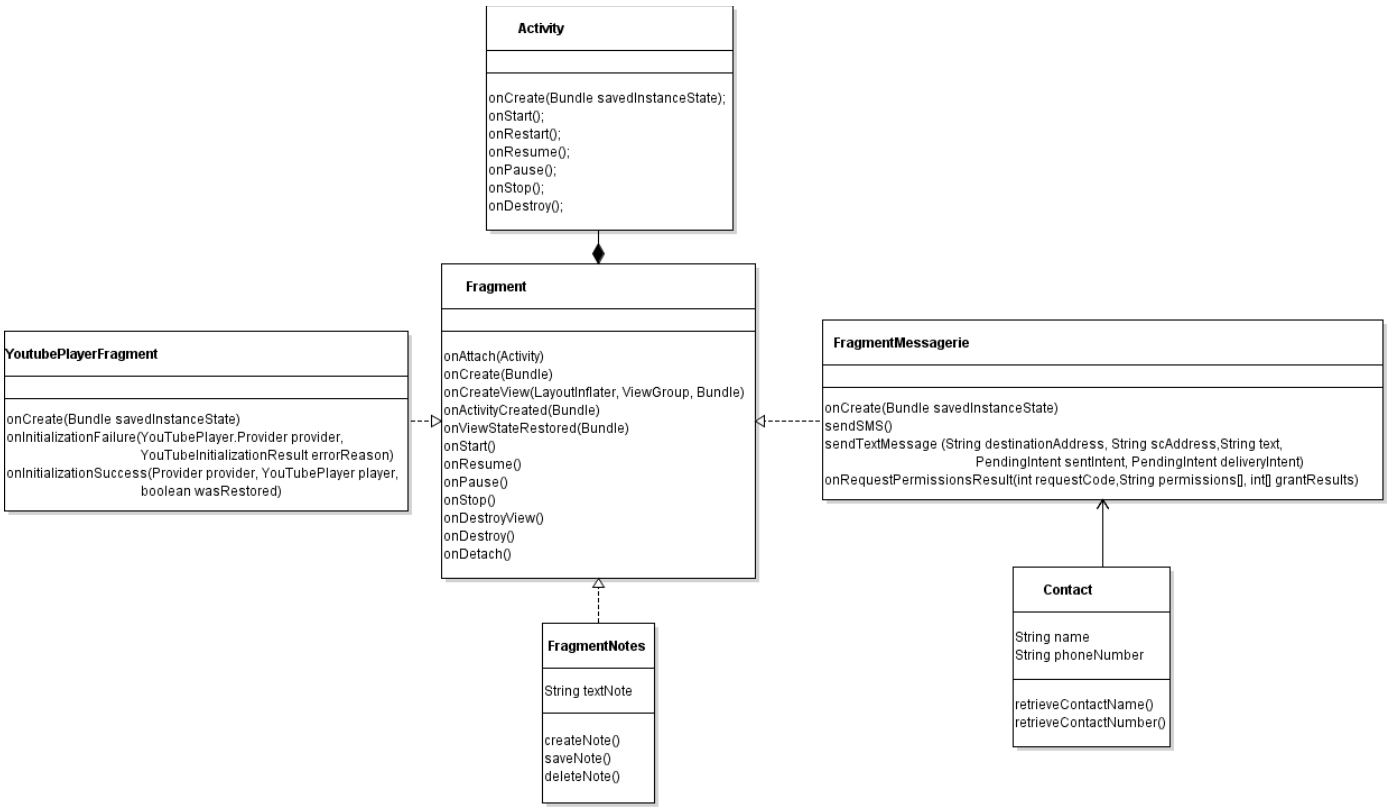

FIGURE 3.4 – Diagramme des classes UML de l'application.

La figure ci-dessus est un diagramme des classes illustrant le fonctionnement de notre application. L'activité est d'abord lancé en utilisant la méthode **onCreate(Bundle savedInstanceState)** dont le paramètre contient l'état précédent de l'activité. Puis, la méthode **onStart()** permet de lancer l'application et donc de la rendre visible. L'appel de la méthode **onResume()** permet d'exécuter les traitements utiles au fonctionnement de l'activité. Enfin, **onStop()** permet de tuer l'activité.

Quant aux fragments, chacun d'entre eux est d'abord rattaché à l'activité parente grâce à **onAttach()**. Il est ensuite initialisé grâce à **onCreate()**. Il faut ensuite créer et charger l'interface du fragment grâce à la méthode **onCreateView()**. Maintenant que l'activité et le fragment sont créés, la méthode **onActivityCreated()** est appelée et marque la fin

du cycle de création de l'interface. De façon analogue à l'activité, les méthodes **onStart()** et **onResume()** sont appelées.

On utilise la méthode **onPause()** pour changer l'état du fragment : ses événements sont désactivés. Cet appel est suivi par **onStop()** et le fragment n'est alors plus au premier plan (ex : Le fragment YouTube passe au premier plan et le fragment messagerie est en second plan).

Pour détruire un fragment, les méthodes suivantes sont utilisées : **onDestroyView()** pour détruire la vue, **onDestroy()** pour détruire le fragment et **onDetach()** pour séparer le fragment de l'activité parente.

3.4 Mise à jour finale du projet

Nous présentons ici la version retenue de la conception de notre application. Les dernières spécifications sont précisées ainsi que les fonctionnalités. Le changement principal retenu est que nous ne laissons pas le choix à l'utilisateur de pouvoir "sélectionner", comme dit plus haut, ses différents modules.

Nous avons utilisé la **YouTube Android Player** Application Programming Interface (API) qui permet d'intégrer la fonctionnalité de lecture vidéo dans notre application à l'aide de méthodes définies permettant de charger des vidéos dans un lecteur de l'interface utilisateur de notre application, en utilisant les identifiants des vidéos, de personnaliser et de contrôler le lecteur vidéo. Toutefois cette dernière ne permettait pas d'effectuer une recherche, car nous devions fournir les identifiants de chaque vidéo qu'on veut charger ; nous avons alors pensé à utiliser la YouTube Data API pour pouvoir coder le système de recherche, mais celle-ci s'est avérée trop complexe. Une autre solution était d'utiliser une requête HTTP vers les serveurs de YouTube (*https ://www.googleapis.com/youtube/v3/search*), nous permettant de récupérer un fichier JavaScript Object Notation (JSON) contenant toute sortes d'informations sous forme d'une collection de couples clés/valeur : le titre, l'identifiant, la catégorie, la date de mise en ligne, la description, etc. En particulier, pour notre application, nous récupérerons à l'aide de la classe **JSONObject** les titres des vidéos d'une part, afin que l'utilisateur puisse choisir la vidéo à charger, ainsi que les identifiants d'autre part qui serviront à lancer le lecteur en utilisant les méthodes **getJSONObject** et **getJSONArray**.

Au fil du développement, l'usage des fragments a été conservé. En revanche, la façon de les afficher a été repensée. D'abord, nous avons affiché les trois fragments dans la même fenêtre, comme nous pouvons le voir sur la figure ci-dessous.

Cette capture (figure 3.5) est celle d'une version antérieure, fonctionnelle. Comme nous pouvons le remarquer, l'interface n'est pas très belle et l'utilisation peu intuitive, bien que remplissant les objectifs que nous nous sommes fixés. De plus, nous avons réalisé qu'une des habitudes phares des utilisateurs regardant une vidéo sur YouTube était de faire des recherches sur Internet en parallèle. Ainsi, nous avons rajouté un module de navigation Web, ainsi que la possibilité de consulter un calendrier, pour par exemple planifier une date durant une discussion menée avec un ami.

Ainsi, nous avons totalement repensé l'interface : le but était de la rendre plus belle, plus simple et surtout plus pratique. Nous avons alors décidé de changer de stratégie : plutôt que de scinder l'écran en 3 fragments, nous scindons l'écran en deux blocs. Chaque bloc est en réalité un ViewPager, permettant de slider entre des fragments. En résumé, l'interface est composée de deux ViewPager, chacun contenant les fragments correspondant à nos modules.

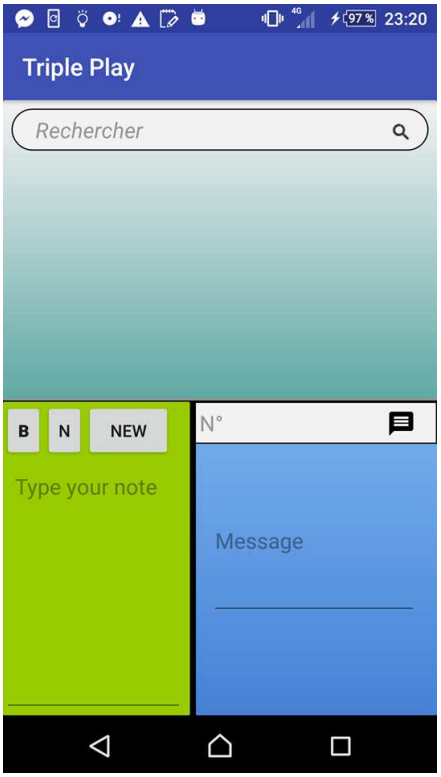

FIGURE 3.5 – Vue de la première version de l'application.

Détaillons comment nous avons codé l'interface. Le cœur de travail réside dans la façon de peupler les ViewPager. Il s'agit d'insérer les fragments dynamiquement. Pour cela, chaque bloc est composé de fragments dynamiques et statiques. Chaque fragment a son interface créée grâce à **onCreateView()**. D'abord, on crée un fragment racine nommé RootFragment() qui sert de support : il ne sert que de base, et on y greffe successivement les différents fragments. On crée le slide grâce à un Adapter : SlidePagerAdapter qui se trouve dans la classe **MainActivity()**, qui permet de rajouter pour un bloc le fragment racine RootFragment() ainsi que les fragments statiques. Puis, le RootFragment instancié laisse place au fragment que l'on veut réellement afficher, à savoir FragmentYouTube() pour le bloc du haut et FragmentContacts() pour le bloc du bas. Ces deux fragments laissent ensuite chacun place respectivement à FragmentYoutubeVideo() et FragmentMessagerie(), pendant que les fragments statiques eux restent inchangés.

3.5 Tests unitaires

Dans notre développement, tous les modules communiquent entre eux, ce qui signifie que nos méthodes n'implémentent pas de résultats attendus. Précisons que la réalisation de tests unitaires est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme (appelée «unité» ou «module»). Pour ces raisons, nous n'avons pas mis en œuvre de démarche de tests unitaires.

3.6 Tests d'intégration et de validation

Pour les tests d'intégration et de validation,nous présentons les Logcat présentés par Android Studio. Le logcat est l'interface permettant de visualiser tous les messages imprimés par les différentes applications présentes sur un appareil. Par cet outil, nous pouvons voir les messages de logs affichés par l'appareil cible, filtrer les messages à afficher selon l'application, l'identifiant ou le tag d'une application, rechercher un message, filtrer les

messages par importance et sauvegarder les logs sélectionnés dans un fichier. Nos tests montrent que chacune des méthodes effectue bien la fonction voulue et que les différentes communications entre elles se passent convenablement. Le fichier vidéo fourni avec le présent rapport constitue un complément à ces captures.

```
D/listeContacts: Aliouat Raouf : 555-55
D/listeContacts: Berraho Amine : 1 222-3
D/listeContacts: Maman : 1 234-5
D/listeContacts: Mon Numéro : 555-4
D/listeContacts: Oussedik Nessim : 666-66
D/listeContacts: Papa : 234-5
```

FIGURE 3.6 – Logcat liste de contacts.

```
D/appuiContact: Vous avez appuyé sur le contact dont le nom est Mon Numéro  et le numéro est 555-4
D/destinataire: Vous écrivez à Mon Numérode son 555-4
```

FIGURE 3.7 – Logcat message.

```
05-22 22:06:32.832 6321-6339/com.example.hp.finale D/listeTitres: E-santé : les plateformes de Télécom SudParis
    Télécom SudParis | Clip des Admissibles 2017
    FR - TELECOM SudParis - Programme Ingénieur - Prépas
    Télécom SudParis | Clip des Admissibles 2016
    Télécom SudParis | Clip des Admissibles 2015
    Clip diplômés Télécom Sudparis promotion 2015
    Le Best Of - TELECOM SudParis - Programme Ingénieur - Prépas
    TELECOM SudParis.wmv
    Etudiant en échange à Télécom SudParis
    Télécom SudParis fait sa JPO
05-22 22:06:32.833 6321-6339/com.example.hp.finale D/listeTitres: UniJam 2017 à Télécom SudParis
    Télécom SudParis | Clip des Admissibles 2014
    Télécom SudParis sur Campus-Channel
    Le métier d'ingénieur, par Guillaume Contet, TF1, diplômé Télécom SudParis 2001
    La Campagne BDE 2015 Télécom SudParis
05-22 22:06:32.834 6321-6339/com.example.hp.finale D/listeTitres: La Campagne BDE 2015 Télécom SudParis
    Télécom SudParis | Clip des Admissibles 2012
    UniJam 2017 à Télécom SudParis
    Pompom Telecom SudParis et Management - Préparation TOSS 2016
    Télécom SudParis - Nissem Selmene
    Présentation Télécom SudParis
    Assia Benbihi - Télécom SudParis - candidate aux Prix des meilleurs stages 2017
    Harlem Shake à Télécom Ecole de Management et Télécom SudParis
    SCI Télécom SudParis & Télécom Ecole de Management
    Visite virtuelle du nouveau bâtiment de Télécom SudParis à Palaiseau
05-22 22:06:32.834 6321-6339/com.example.hp.finale D/resultatTitres: La liste des titres a été affichée avec succès
```

FIGURE 3.8 – Logcat titres des vidéos - recherche Télécom SudParis.

FIGURE 3.9 – Logcat id vidéos.

FIGURE 3.10 – Logcat sélection vidéo.

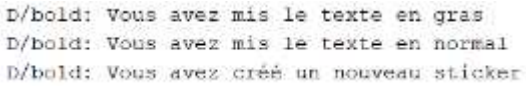

FIGURE 3.11 – Logcat NotePad.

Partie 4

Manuel utilisateur

L'utilisation de notre application repose sur une interface utilisateur facile d'accès. Au lancement s'affichent deux blocs, l'un sur la partie supérieure de l'écran du téléphone, l'autre sur la partie inférieure. Ces deux portions regroupent plusieurs activités chacune.

La partie du haut affiche par défaut une interface YouTube. L'utilisateur peut, par l'intermédiaire de la barre de recherche, saisir une requête de vidéo. Les résultats s'affichent au-dessous de la barre de recherche. On n'a alors qu'à sélectionner la vidéo qu'il souhaite visionner. Un lecteur s'ouvre et la vidéo se lance. Pour revenir à l'écran de recherche, il suffit de cliquer sur le bouton retour. C'est d'ailleurs le cas pour chaque retour à une ancienne activité au sein de chaque module.
En glissant vers la droite, l'utilisateur peut profiter du module NotePad. Ce module offre la possibilité de saisir une note et, grâce à ses boutons sur le haut de l'écran, d'en éditer la police (normal ou gras), ainsi que de passer à une nouvelle note.

La partie du bas affiche, au lancement, la liste des contacts enregistrés sur le téléphone. En sélectionnant l'un deux, on peut alors lui envoyer un ou plusieurs messages grâce à l'interface d'envoi qui s'affiche. En glissant vers la droite, l'utilisateur a la possibilité d'effectuer des recherches sur Google ainsi que de naviguer sur internet. Cette fonctionnalité s'avère très utile lorsqu'elle est combinée avec la lecture d'une vidéo YouTube. On peut glisser une dernière fois vers la droite pour accéder à un module de calendrier. Notons enfin qu'à chaque instant un scroll vers la gauche permet de revenir à l'activité antérieure.

(a) (b)

FIGURE 4.1 – Vue de l'interface utilisateur offerte par l'application : (a) Recherches YouTube et Google simultanées. (b) Lecture d'une vidéo et vue d'une page internet simultanées.

Partie 5

Bilan du projet

Les développements ainsi que le codage effectués au cours de notre projet nous ont permis d'avoir une première approche technique du monde du développement Android. Ce type de développement est basé sur le langage Java, que nous avons eu l'opportunité d'étudier et de pratiquer dans le cadre de notre formation.

Android est un système d'exploitation mobile : tout comme Windows ou Linux c'est un gros programme, composé de petits programmes, qui permet d'exécuter d'autres logiciels. Par exemple, Windows permet d'exécuter Internet Explorer, et pour ce faire, il doit faire le lien entre la souris et le curseur à l'écran, entre le clavier et les champs de saisie, etc. Et avec l'explosion des ventes de smartphones ces dernières années, Android a pris une place importante dans la vie quotidienne de millions de personnes, au point qu'il s'agit du système d'exploitation mobile avec le plus d'applications en circulation. Nous avons pu, en entrant dans cet univers, mettre au point une application complète et complexe.

Du point de vue technique, les méthodes que nous avons mises en œuvre se sont inscrites dans notre cap initial qui était d'imbriquer et d'inter-corréler chacun des modules de l'application. L'architecture globale de l'application correspond donc à cette volonté de présenter une application aussi facile d'accès et intuitive que de haut niveau en termes techniques.

Pour l'utilisation des nombreuses subtilités de l'interface de développement offerte par Android Studio comme pour l'implémentation de certaines méthodes, nous nous sommes abondamment servi de l'excellent ouvrage de Nazim Benbourahla intitulé *Android 7 - Les fondamentaux du développement d'applications Java*. Un nombre certain de questions ont ainsi trouver des réponses claires et détaillées.

Cette démarche de sans cesse remettre en question l'existant est sans doute celle qui nous a le plus animés au cours de notre travail. Nous avons ainsi vagabondé d'idées en idées, de codage en codage pour améliorer l'application finale. C'était également une démarche très enrichissante qui, nous en sommes certains, et au vu du résultat final de ce projet tant ambitieux que de qualité, nous apportera énormément dans nos parcours de futurs développeurs ou, plus généralement, de membre d'un projet de haut niveau.

Enfin, concernant l'organisation des membres de l'équipe-projet, nous avons pris la décision d'utiliser l'outil collaboratif BitBucket pour le partage, la mise en commun et la mise à jour de la partie codage. Nous tenons par ailleurs ici à remercier vivement notre encadrante, Amel Bouzeghoub, pour sa bienveillance, sa disponibilité et les précieux conseils qu'elle a su nous promulguer dans des moments de blocage. Nous remercions également l'ensemble du département informatique de Télécom SudParis et tous les enseignants, coordonateurs et membres du personnel sans qui cette aventure enrichissante n'aurait pu voir le jour.

Annexe A

Interface de développement Android Studio

Le développement ainsi que la mise en œuvre de notre application est réalisé par l'intermédiaire de l'environnement de développement Android Studio. Il s'agit en effet d'un environnement de développement d'applications Android.

FIGURE A.1 – L'environnement de développement Android Studio permet de produire des applications compatibles Android de haut niveau.

Android Studio permet principalement d'éditer les fichiers Java/Kotlin et les fichiers de configuration XML d'une application Android. Il propose entre autres des outils pour gérer le développement d'applications multilingues et permet de visualiser la mise en page des écrans sur des écrans de résolutions variées simultanément.

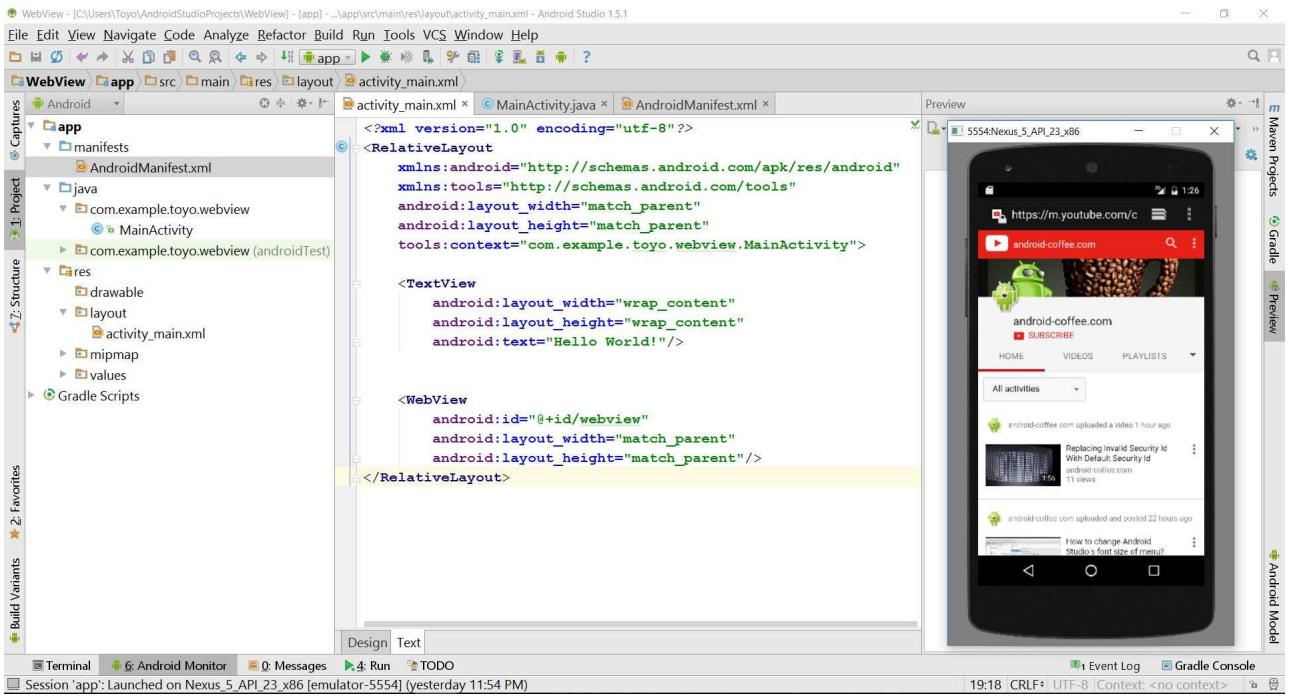

FIGURE A.2 – Illustration des fonctionnalités de développement offertes par Android Studio.

Annexe B

Références

[1] Documentation de la classe *Manifest.permission* de l'API Android - disponible sur **https ://developer.android.com/reference/android/Manifest.permission.html# BROADCAST_ SMS** .

[2] Documentation de la classe *YouTubePlayerFragment* de l'API Android Player - disponible sur **https ://developers.google.com/youtube/android/player/reference/com/google/android/youtube/player/YouTubePlayerFragment** .

[3] Documentation de la permission READ_ CONTACTS de l'API Android - disponible sur **https ://developer.android.com/training/contacts-provider/retrieve-names.html# Permissions** .

[4] Documentation de la classe *SmsManager* de l'API Android - disponible sur **https ://-developer.android.com/reference/android/telephony/SmsManager.html** .

[5] Documentation de la classe *SmsMessage* de l'API Android Telephony - disponible sur **https ://developer.android.com/reference/android/telephony/SmsMessage.html**.

[6] Ouvrage *Android 7 - Les fondamentaux du développement d'applications Java* de Nazim Benbourahla - Éditions ENI - janvier 2017.