Université libre de Bruxelles

[INFO-F308]

Mini mémoire

Construction d'un réseau social en Python

Auteur:
Thomas Weyssow

Superviseur:
Thierry Massart

21 Mai 2018

Introduction

Dans le cadre de ce mini mémoire, le travail qu'il nous a été demandé de réaliser peut se diviser en quatre tâches. En premier lieu, établir une liste des fonctionnalités principales que doit comporter un réseau social sur base de l'analyse de réseaux sociaux déjà existants. En second lieu, effectuer la recherche de plateformes permettant l'implémentation d'un réseau social comportant ces fonctionnalités en Python et portable sur système Android et iOS. Ensuite, développer un prototype fonctionnel. Enfin, rendre possible l'utilisation du code produit pour la rédaction d'un projet pour le cours de programmation de la première année du Bachelier en sciences informatiques. Le travail réalisé est expliqué plus en détail dans la suite.

1 Les réseaux sociaux

1.1 Exemples de réseaux sociaux existants

Pour commencer, analysons les fonctionnalités que comportent les deux réseaux sociaux les plus utilisés pour le moment. Il s'agit de Facebook et Twitter.

1.1.1 Facebook

Facebook est le réseau social le plus utilisé, avec plus d'un milliard d'utilisateurs. Décrivons les fonctionnalités principales qu'il propose.

Le cœur de métier de Facebook est le fil d'actualité. En plus d'afficher les publications des personnes auxquelles l'utilisateur est abonné, un algorithme a été implémenté (Newsfeed Ranking Algorithm) pour proposer du contenu pertinent en fonctions des goûts et des affinités de l'utilisateur. Le fil d'actualité est aussi une interface de publication, l'utilisateur peut directement y publier du contenu.

Le profil Facebook permet d'enregistrer, en plus des informations personnelles et de la photo de profil, des photos (avec la possibilité de créer des albums) et des vidéos. De plus, toutes les publications de l'utilisateur y apparaissent.

En plus du fil d'actualité, Facebook propose d'autres outils de publication:

- Les pages: Un utilisateur peut créer une page pour créer une communauté et publier du contenu relatif à un certain thème. Il peut permettre à d'autres utilisateur d'y publier du contenu. Ceux qui n'ont pas le droit de publication peuvent s'y abonner et/ou la partager.
- Les groupes: Les groupes sont des pages dédiées à la création d'une communauté plus restreinte et privée, dotées d'outils facilitant l'échange et la collaboration.

En plus d'une messagerie traditionnelle, Facebook propose un chat permettant à un utilisateur de discuter avec ses contacts en temps réel et de faire des appels vocaux ou vidéo. De plus l'utilisateur peut créer une conversation avec plusieurs de ses contacts.

La barre de recherche de Facebook permet de chercher, par mot-clé, un autre utilisateur, une page ou un groupe.

Facebook permet, en plus de commenter une publication, d'y apposer un "Like". Chaque publication affiche le nombre de "Like" qu'elle a reçu. Il permet aussi de mentionner un autre utilisateur, la publication s'affiche alors dans le fil d'actualité de ce dernier.

1.1.2 Twitter

Twitter est très similaire à Facebook quant aux fonctionnalités qu'il propose, à la différence qu'il ne permet pas de créer de pages, de groupe ni d'ajouter de contacts. Il ne propose pas de chat mais permet toutefois d'envoyer des messages privés aux autres utilisateurs. De plus, l'application ne permet aucune confidentialité quant aux publications, elles sont accessibles par tous les autres utilisateurs. Une autre différence avec Facebook est que le profil d'un utilisateur ne peut pas contenir d'albums photos, ni de vidéos mais seulement une photo de profil et ses publications.

Twitter est lui aussi focalisé sur le fil d'actualités, qui est l'outil de publication de l'utilisateur et qui affiche en temps réel les publications des personnes auxquelles il est abonné. La spécificité de Twitter est qu'il impose des limitations quant à la quantité de contenu que l'on peut publier en une fois. Un utilisateur peut publier soit un texte seul de maximum 140 caractères, soit un texte accompagné d'une photo (l'espace dédié au texte est alors réduit à 117 caractères), soit une vidéo courte de 30 secondes. De plus, l'utilisateur peut utiliser un # pour lier sa publication à un mot-clé (ex: #DonaldTrump) et un @ pour y mentionner un autre utilisateur (ex: @Thomas), ce dernier en est directement notifié.

1.1.3 Autres

Il existe encore beaucoup d'autre réseaux sociaux qui se focalisent sur une fonctionnalité plus spécifique tel que le partage de vidéos (Youtube), de photos (Instagram, Snapchat) ou sur le chat (WhatsApp, Google Hangouts). Tous ces réseaux sociaux proposent, en plus de leur fonctionnalité principale, la majorité des fonctionnalités présentées dans la section 1.

1.2 Le fil d'actualités

Le fil d'actualité est le cœur de métier des réseaux sociaux les plus massivement utilisés. Les publications s'affichant sur le fil d'actualité d'un utilisateur sont choisies algorithmiquement comme, par exemple, par le "Newsfeed Ranking Algorithm" de Facebook. Des études ont montré que ces algorithmes peuvent avoir un effet néfaste sur les utilisateurs. En effet, ils peuvent pousser à la consommation ou enfermer les utilisateurs dans une "bulle" en n'affichant que des publications liées à une certaine opinion politique ou religieuse.

Il est bien connu que les compagnies possédant ces réseaux sociaux, bien qu'étant gratuits à l'utilisation, ont une marge de revenu gigantesque. Ces revenus proviennent majoritairement de la publicité. Comme les publicités s'affichant sur le fil d'actualité d'un utilisateur sont choisies en fonction de ses goûts et de ses affinités, on peut facilement tirer la conclusion qu'elles poussent l'utilisateur à la consommation.

Le choix des publications affichées dans le fil d'actualité en fonction des opinions politiques et religieuses peut aussi être dangereux. Il est connu que l'ISIS a utilisé Facebook pour leur recrutement. En effet, si un utilisateur partage beaucoup de publications liées à l'Islam, il est possible que les publications affichées dans son fil d'actualité soient elles aussi liées à l'Islam et parmi celle-ci des publications liées à une vision extrémiste de l'Islam telle que celle de l'ISIS. Si l'utilisateur s'intéresse

à ces publications extrémistes, son fil d'actualité se retrouvera pollué par celle-ci. On peut facilement voir le danger résultant.

1.3 Les fonctionnalités principales d'un réseau social

Sur base de l'analyse présentée ci-dessus, établissons une liste reprenant les fonctionnalités principales qui se doivent d'être implémentées dans un réseau social. Cette liste peut être vue comme le cahier des charges pour l'implémentation d'un prototype de réseau social.

- **Enregistrement et connexion**: Un utilisateur doit s'enregistrer avec un pseudo et un mot de passe pour pouvoir se connecter et utiliser l'application.
- **Création d'un profil**: Un utilisateur doit créer un profil contenant quelques informations le concernant, ainsi qu'une photo. Le profil est accessible par tous les utilisateurs de l'application.
- **Ajout/suppression de contacts**: Un utilisateur peut en ajouter un autre à sa liste de contacts en lui envoyant une demande (l'autre utilisateur a le choix de refuser).
- **Publier du contenu**: Un utilisateur peut publier du contenu (texte, photo, vidéo, lien vers une page web, ...) qui est alors visible par les autres utilisateurs.
- **Messages**: Un utilisateur peut d'envoyer du contenu à une autre personne.
- **Confidentialité**: Une information du profil ou une publication peut être rendue privée. Elle est alors seulement accessible par certaines personnes choisies par l'utilisateur.
- **Recherche**: Une recherche peut être effectuée pour trouver un utilisateur ou une publication.
- **Abonnement**: Un utilisateur peut s'abonner à un autre utilisateur. Il est alors notifié à chaque nouvelle publication de ce dernier.
- **Fil d'actualité**: Les publications des personnes auxquelles l'utilisateur est abonné sont affichées en temps réel.
- **Commentaire**: Les publications peuvent être commentées.
- **Partage**: Les publications peuvent être partagée à d'autres utilisateurs, ils en sont alors notifiés.

2 Les plates-formes permettant l'implémentation d'un réseau social en Python

L'implémentation d'un réseau social, inspiré de Facebook ou Twitter, nécessite d'implémenter: une interface graphique, un réseau et d'utiliser une base de données. Il existe plusieurs plateformes intéressantes permettant de réaliser et/ou de faciliter le développement d'une telle application. On désire développer, en Python 3, une application portable sur smartphone et d'en faire un projet pour le cours de programmation des étudiants en première année en sciences informatiques, ce qui impose des contraintes sur le choix des plateformes que l'on peut utiliser.

2.1 L'interface graphique

Pour l'interface graphique, il existe la plateforme *Kivy*. La particularité de *Kivy* est qu'il permet de créer des applications utilisables avec un écran tactile et qu'il fonctionne sur Android, iOS, Linux, OS X et Windows. De plus, il permet de développer une application utilisant la fonctionnalité multi-touch des smart phones. Il existe un logiciel, appelé *Buildozer*, qui permet de créer un paquet Android ou iOS à partir d'une application développée avec *Kivy* (*Buildozer* ne peut être installé que sur un système Linux). Cette plateforme serait donc le meilleur choix pour l'implémentation de l'interface graphique d'une application portable sur Android et iOS.

Un autre choix possible serait d'utiliser le logiciel *QPython3*. Il s'agit d'un logiciel qui s'installe directement sur un système d'exploitation Android et qui permet de développer des applications en Python 3 sur ce système. Ce logiciel ne permet pas de créer de paquets Android, il se charge d'exécuter les scripts directement. De plus, il intègre la plateforme *Kivy*. Ce choix parait donc beaucoup moins judicieux, il ne permet aucune portabilité et un utilisateur voulant utiliser une application développée avec ce logiciel doit obligatoirement l'avoir installé.

En enlevant la contrainte de devoir développer une application portable sur smart phone, il existe alors de nombreux choix possibles de plateformes pour implémenter l'interface graphique tels que *Tkinter*, *PyQt*, *PyGame* et bien d'autres.

2.2 Le réseau

Pour la partie réseau, il existe la plateforme *Twisted*. Cette plateforme est un moteur réseau événementiel écrit en Python 2 mais la plupart de ses modules ont été portés en Python 3. Il permet de faire abstraction d'appels système de bas niveau tels que select() et socket() et propose beaucoup de fonctions et de classes utilitaires qui facilite l'implémentation de serveurs. Il existe aussi d'autre alternatives, parmi lesquelles la plateforme *0mq*. Un concept intéressant de ces plateformes est qu'elles permettent de transférer des flux de messages (ayant une taille bien définie) entre les clients et le serveur et non un flux de bytes, ce qui n'est pas le cas avec les protocoles TCP et UDP.

Il est important de noter qu'il est possible d'intégrer un moteur *Twisted* qui s'exécutera dans la boucle d'événements de *Kivy* grâce à une fonction de ce dernier. Ces deux plateformes semblent donc être les plus adaptés à l'implémentation d'un réseau social en Python. De plus, ils donnent la possibilité de porter l'application sur Android et iOS.

Malheureusement, bien que l'intégration de *Twisted* avec *Kivy* soit fonctionnelle en Python 3, le logiciel *Buildozer* ne permet d'utiliser *Twisted* qu'avec Python 2 et ne permet pas d'utiliser *0mq*.

L'utilisation d'une plateforme pour la partie réseau n'est bien sûr pas obligatoire, on peut utiliser le module *socket*, intégré nativement dans Python 3, qui permet une implémentation plus bas niveau d'une application client-serveur utilisant un protocole TCP ou UDP.

2.3 La base de donnée

Pour l'utilisation d'une base de données, il existe un grand choix de plateformes permettant de se connecter à un serveur de base de données. *PyMySQL* pour MySQL, *sqlite3* pour SQLite, *psycopg2* pour PostgreSQL et bien d'autres. L'utilisation d'une base de données n'est pas obligatoire, une autre

possibilité serait d'utiliser des fichiers pour enregistrer les données de l'application ou encore d'utiliser le module *shelve*.

2.4 Les plateformes choisies

Les plates-formes choisies sont:

- La plateforme *Kivy* pour l'implémentation de l'interface graphique.
- Les modules *socket* et *select* pour le modèle client-serveur.
- Le module *shelve* pour la base de données.
- Le logiciel *Buildozer* pour le portage vers Android et iOS.

2.5 Présentation de Kivy

Présentons, dans cette section, les principes de conception de la plateforme *Kivy*.

Il est important de noter que tous les éléments graphiques (ex: les boutons, les labels contenant du texte, ...) que l'on peut utiliser dans une application écrite en utilisant cette plateforme s'appellent des widgets (les classes représentant les éléments graphiques héritent tous de la classe Widget).

De plus, les widgets utilisés dans une application sont organisés dans une structure arborescente, c'est-à-dire que l'application contiendra un widget racine qui lui-même contient un ou plusieurs autres widgets (ses enfants) et ainsi de suite.

Aussi, les attributs (la taille, la position, Le texte contenu, ...) des widgets peuvent être définit directement lors de leur instanciation en passant des arguments clé/valeur à leur constructeur (ex: `Button(text='hello world')`).

En outre, une taille et une position par défaut sont assignées aux widgets lors de leur instanciation et leur taille et leur position réelles ne sont calculées que lorsqu'on les ajoutes à leur widget parent (avec la méthode `add_widget()`). Il est possible de lier un attribut d'un widget à une méthode qui sera appelée lors d'un changement sur cet attribut grâce à la méthode `bind()`, par exemple si on appelle `bind(size=callback)` dans le constructeur d'un widget, la méthode `callback()` sera appelée lors d'un changement de taille du widget (ajout du widget à son widget parent, changement de taille de la fenêtre, ...).

Enfin, il existe deux manières d'apposer manuellement une taille et une position à un widget:

- En utilisant un pourcentage par rapport à la taille et/ou la position de son widget parent. Il faut alors utiliser les attributs `size_hint` et/ou `pos_hint`. Par exemple:
  - Le bouton `Button(size_hint=(.2, .5))` aura une largeur égale à 20% de la largeur de son widget parent et une hauteur égale à 50% de la hauteur de son widget parent.
  - Le coin inférieur gauche du bouton `Button(pos_hint={'x': .2, 'y': .5})` sera positionné à 20% de la largeur de son widget parent vers la droite et à 50% de la hauteur de son widget parent vers le haut.

- En utilisant une taille et/ou une position fixe. Il faut alors utiliser les attribut `size` et/ou `pos`. Si l'on veut assigner une taille fixe à un widget, il faut alors impérativement assigner `(None, None)` à son attribut `size_hint`. Par exemple:
  - Le bouton `Button(size=(50, 200), size_hint=(None, None))` aura une largeur de 50 pixels et une hauteur de 200 pixels.
  - Le coin inférieur gauche du bouton `Button(pos=(50, 200))` sera positionné à 50 pixels vers la droite et 200 pixels vers le haut.

Présentons, ci-dessous, les modules de la plateforme qui seront utiles lors du développement d'un réseau social.

2.5.1 kivy.app

La classe App est la classe de base pour créer une application *Kivy*, elle est en quelque sorte le point d'entrée vers la boucle d'événements. Voici un exemple simple:

```
from kivy.app import App
from kivy.uix.button import Button

class TestApp(App):
    def build(self):
        return Button(text='hello world')

if __name__ == '__main__':
    TestApp().run()
```

Une classe héritant de la classe App doit obligatoirement définir la méthode build() qui renvoie le Widget racine de l'application.

2.5.2 kivy.uix.screenmanager

Ce module contient le gestionnaire d'écrans: la classe ScreenManager et les écrans, la classe Screen:

```
from kivy.uix.screenmanager import ScreenManager, Screen

sm = ScreenManager()

for i in range(4):
    screen = Screen(name="Title {:d}".format(i))
    sm.add_widget(screen)

sm.current = "Title 2"
```

Par défaut, le premier écran affiché est le premier écran ajouté au gestionnaire avec la méthode add_widget(). Pour sélectionner un autre écran, il suffit d'assigner la valeur de l'attribut *name* de l'écran que l'on désire sélectionner à l'attribut *current* du gestionnaire.

2.5.3 kivy.uix.layout

Ce module contient des classes servant de conteneur pour les widgets et permettant généralement de faciliter leurs positionnements et le calcul leurs tailles en les calculant automatiquement. Il existe 8 de ces classes, parmi lesquelles les plus importantes sont:

- FloatLayout: qui n'impose aucune restriction sur le positionnement et la taille des widgets, celles-ci doivent alors être définie manuellement.
- BoxLayout: qui empile les widgets de façon verticale ou horizontale dans l'espace disponible.
- GridLayout: qui divise l'espace disponible en lignes et colonnes et empile les widgets dans les cellules ainsi créées.

2.5.4 kivy.uix.label

Ce module contient la classe Label qui permet d'afficher du texte. Le texte est définit dans l'attribut *text*, par exemple:

```
from kivy.uix.label import Label

l = Label(text="Hello world")
```

2.5.5 kivy.uix.textinput

Ce module contient la classe TextInput qui fournit une boite permettant à l'utilisateur de taper du texte. Ce texte est contenu dans l'attribut *text*:

```
from kivy.uix.textinput import TextInput

textinput = TextInput(multiline=False)
```

2.5.6 kivy.uix.button

Ce module contient la classe Button qui est un Label associé à des événements qui sont déclenchés quand le bouton est appuyé ou relâché:

```
from kivy.uix.button import Button

def callback(instance):
    print("The button {:s} is being pressed".format(instance.text))

btn = Button(text='Hello world 1')
btn.bind(on_press=callback)
```

Dans cet exemple, la méthode *callback* est appelée lorsque l'événement *on_press* est déclenché, c'est-à-dire lorsque le bouton est appuyé.

2.5.7 kivy.uix.scrollview

Ce module contient la classe ScrollView. Une instance de cette classe est un conteneur de taille fixe possédant une barre de déroulement dans lequel il est possible d'ajouter des widgets. Une telle classe semble indispensable lors du développement d'un réseau social (pour le fil d'actualité, l'espace de publication, ...). L'exemple présenté ci-dessous correspond à l'implémentation d'une ScrollView de taille extensible, c'est-à-dire que sa taille s'adapte à la taille cumulée des widgets qu'elle contient, à laquelle il est possible d'ajouter des labels.

```
from kivy.uix.scrollview import ScrollView
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout

class ExtensibleScrollView(BoxLayout):

    def __init__(self, **kwargs):
        super(ExtensibleScrollView, self).__init__(**kwargs)
        self.scrollView = ScrollView()
        # Le conteneur des widgets qui seront ajoutes
        self.scrollViewLayout = BoxLayout(
            orientation='vertical',
            size_hint_y=None
        )
        # Adaptation de la taille du conteneur
        self.scrollViewLayout.bind(
            minimum_height=self.scrollViewLayout.setter('height')
        )
        self.scrollView.add_widget(self.scrollViewLayout)
        self.add_widget(self.scrollView)

    def addLabel(self, label):
        self.scrollViewLayout.add_widget(label)
```

2.5.8 kivy.graphics

Ce module contient des classes utilisées pour dessiner des formes à l'écran. Il est important de noter que chaque widget contient un objet Canvas. C'est cet objet qui est utilisé comme conteneur pour les dessins. Changer la couleur du canvas d'un widget est la seule manière de changer la couleur de fond de ce widget. L'exemple présenté ci-dessous correspond à l'implémentation d'un layout auquel il est possible de changer la couleur de fond.

```
from kivy.uix.floatlayout import FloatLayout
from kivy.graphics import Color, Rectangle

class CustomFloatLayout(FloatLayout):

    def __init__(self, colors, **kwargs):
        super(CustomFloatLayout, self).__init__(**kwargs)
        # Le canvas permet de changer la couleur de fond
```

```
        with self.canvas:
            Color(colors[0], colors[1], colors[2], 1)
            self.rect = Rectangle(pos=self.pos, size=self.size)
        # Mise a jour du canvas quand le widget change de taille ou
        # de position
        self.bind(pos=self.updateCanvas)
        self.bind(size=self.updateCanvas)

    def updateCanvas(self, instance, values):
        self.rect.pos = self.pos
        self.rect.size = self.size
```

2.5.9 kivy.event

Ce module contient la classe EventDispatcher. Tous les widgets qui peuvent déclencher un événement héritent de cette classe. Elle permet également de créer des événements personnalisés, ce qui est fort utile lors du développement d'une application client-serveur. En effet, on peut, par exemple, créer un événement qui se déclenche lorsque l'on a besoin de communiquer avec le serveur et de gérer cet événement avec une classe spécifique. Cette astuce permet de fortement découpler la communication avec le serveur et l'interface graphique. Visualisons ce concept par un exemple:

```
from kivy.app import App
from kivy.event import EventDispatcher
from kivy.uix.label import Label
from threading import Thread
import socket

class Client(EventDispatcher):

    def __init__(self, **kwargs):
        # Enregistrement de l'evenement personalise
        self.register_event_type("on_message_received")
        super(Client, self).__init__(**kwargs)
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect(("192.168.0.1", 80))

    def start():
        while(True):
            message = socket.recv(1024)
            # Declenchement de l'evenement personalise
            self.dispatch("on_message_received", message)

    def on_data_received(self, data):
        """L'evenement personalise"""
        pass

class BaseApp(App):
```

```
    def __init__(self, **kwargs):
        super(BaseApp, self).__init__(**kwargs)

        self.client = Client()
        # Liaison de l'evenement a la methode displayMessage()
        self.client.bind(on_message_received=self.displayMessage)

        self.thread = Thread(target=self.client.start)
        self.thread.start()

        self.label = Label()

    def displayMessage(instance, message):
        self.label.text = message

    def build:
        return self.label

if __name__ == '__main__':
    BaseApp().run()
```

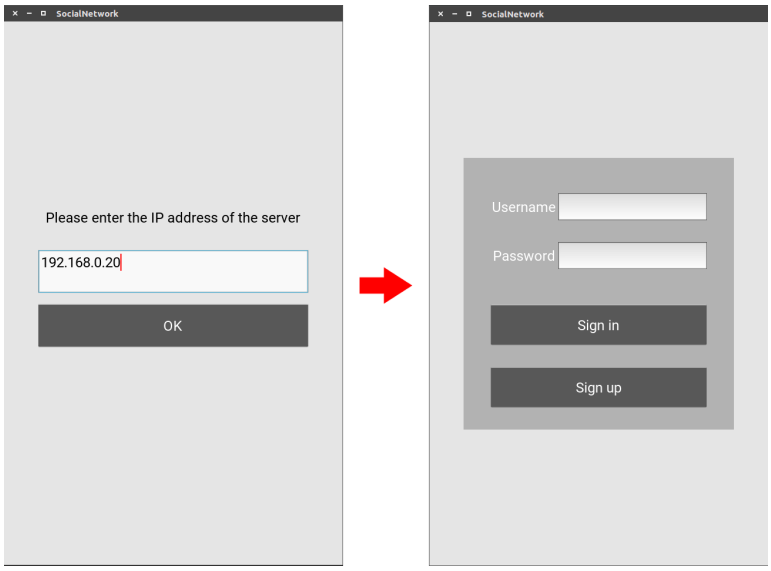
3 Le prototype

L'application qui à été développée en guise de prototype de réseau social écrit en Python et portable sur Android et iOS est décrite dans cette section. Cette application comporte les fonctionnalités listées dans la section 1.3 et utilise les plateformes listées dans la section 2.4. Le code est disponible dans le dossier *MiniMemoire*.

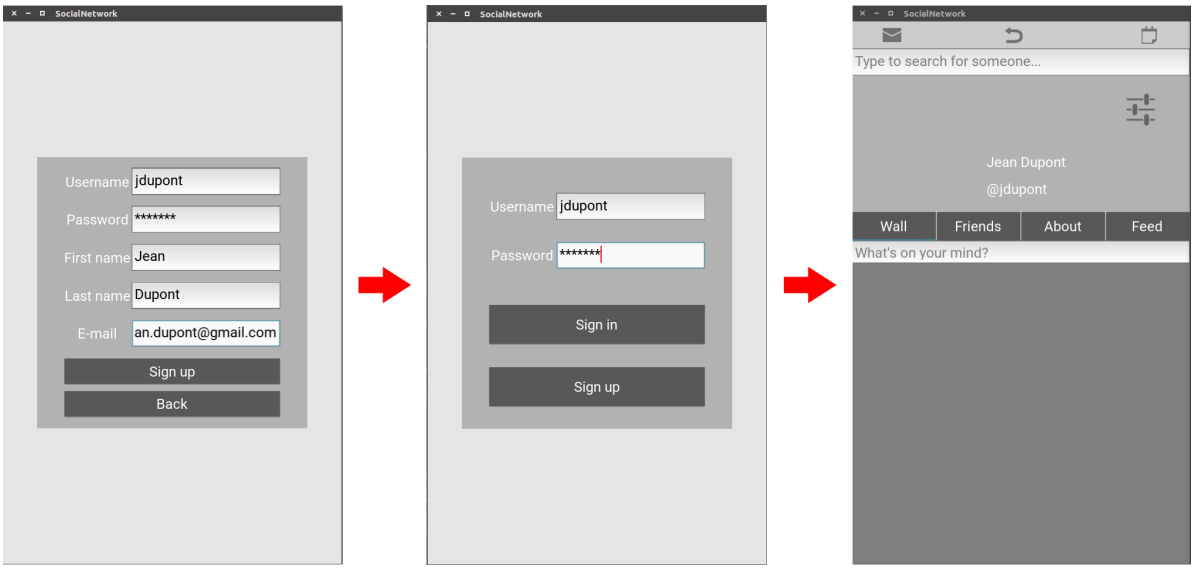
3.1 Les fonctionnalités implémentées

Présentons, dans cette section, les fonctionnalités implémentées dans le prototype.

Lors du lancement de l'application, l'utilisateur est invité à entrer l'adresse IP de la machine sur laquelle la partie serveur est exécutée:

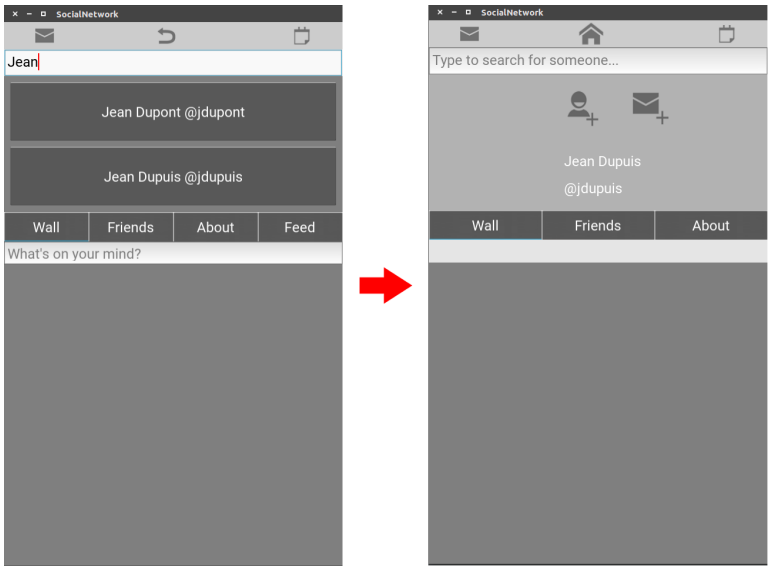


Une fois l'adresse IP validée, l'utilisateur peut s'enregistrer et se connecter à l'application (ou se connecter directement s'il est déjà enregistré), il passe alors à l'écran de profil contenant une barre de menu, un bouton permettant de choisir un paramètre de privacité, et un panneau d'onglet contenant son espace de publication, ses amis et demandes d'amis, ses informations personnelles et son fil d'actualité:

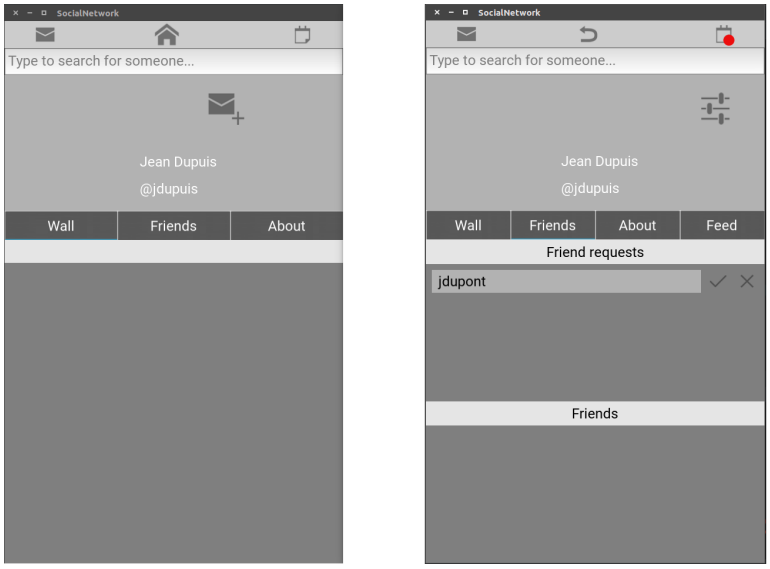


L'écran de profil offre la possibilité d'effectuer la recherche d'un autre utilisateur. Si un résultat de la recherche est sélectionné, alors la page de profil de l'utilisateur correspondant est affichée. Le bouton permettant de choisir le paramètre de privacité est remplacé par un bouton permettant d'envoyer une

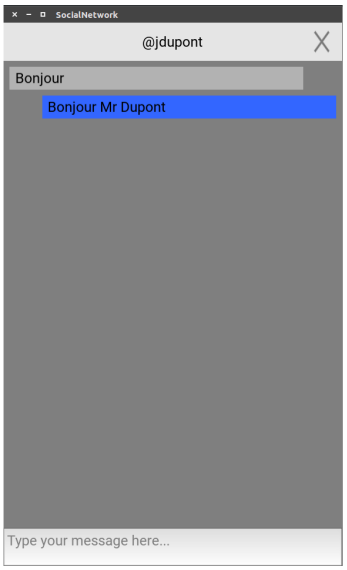
demande d'ami à cet utilisateur et un bouton permettant de lui envoyer un message. Son espace de publication, ses amis et ses informations personnelles peuvent être consultés:



Si une demande d'ami est envoyée à l'autre utilisateur, il en sera alors notifié et aura le choix d'accepter ou de refuser la demande:

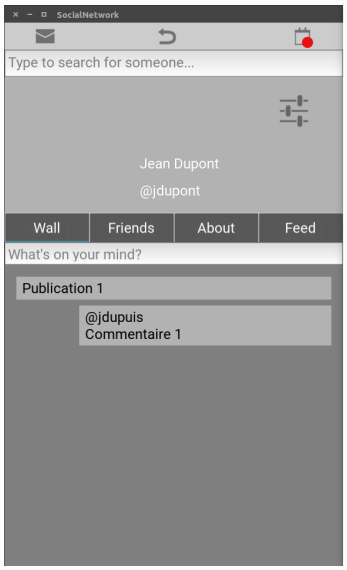


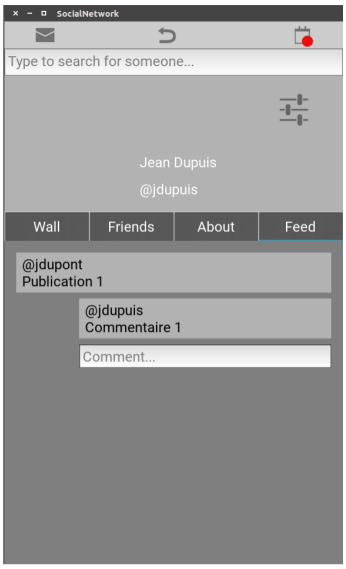
Si un message est envoyé à l'autre utilisateur, il en sera alors également notifié:



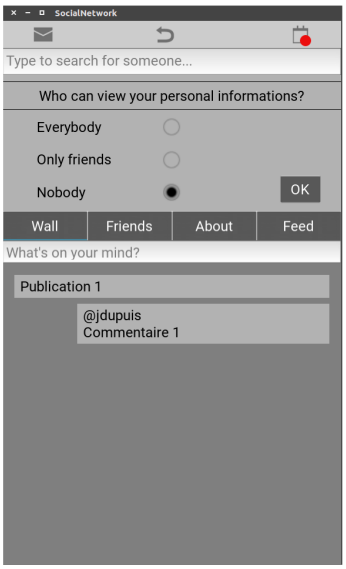
L'utilisateur à la possibilité de revenir sur sa page de profil grâce au bouton central (ce bouton sert à la déconnexion si l'utilisateur se trouve déjà sur sa page de profil) de la barre de menu se trouvant en haut de l'application.

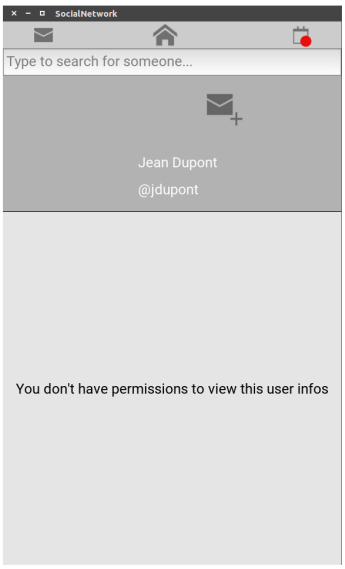
L'espace de publication offre à l'utilisateur la possibilité d'émettre une publication qui s'affichera sur le fil d'actualité de ses amis qui auront la possibilité de la commenter (les utilisateurs concernés en seront alors notifiés):





Le paramètre de privacité permet à l'utilisateur de choisir qui à le droit de voir les informations affichées sur sa page de profil. Il peut s'agir de tous les utilisateurs, seulement ses amis ou aucun utilisateur:





Enfin, le bouton se trouvant à gauche de la barre de menu lui permet d'ouvrir la boite de messagerie lui permettant d'ouvrir une conversation initialisée en envoyant un message à un autre utilisateur et le bouton se trouvant à droite de la barre de menu permet d'afficher ses notifications.

3.2 L'implémentation

Présentons, dans cette section, l'implémentation du prototype. Les classes essentielles utilisées par l'application sont décrites plus en détail.

3.2.1 La base de données

La base de données a été implémentée en utilisant le module *shelve*. Elle est contenue dans le fichier **Users.db** contenu dans le sous-dossier *SocialNetworkServer*. La classe **DAO** (Data Access Object) contenue dans le même sous-dossier contient toutes les méthodes nécessaires à l'ouverture et la fermeture de la base de données ainsi qu'à l'écriture et la lecture de données. Cette classe n'est utilisée que par la partie serveur de l'application.

3.2.2 Le modèle client-serveur

Le modèle client-serveur de l'application à été implémenté en utilisant les modules *socket*, *select* et *pickle*. Il est important, avant tout, de décrire la conception choisie pour le transfert de donnée entre les clients et le serveur. La partie client et la partie serveur de l'application utilisent toute deux un unique appel à la méthode recv() (dans la boucle des classes **Client**, contenue dans le paquet *Network* du sous-dossier *SocialNetwork*, et **Serveur**, contenue dans le sous-dossier *SocialNetworkServer*, respectivement). Les données sont transférées sous la forme de dictionnaires Python sérialisés en utilisant le module *pickle*. Dès qu'un client doit envoyer des données au serveur ou que le serveur doit envoyer des données à un client dans le but d'effectuer une opération (l'enregistrement d'un utilisateur, l'envoi d'un message d'un utilisateur à un autre utilisateur, ...), un nom correspondant à cette opération est associé aux données. Les données peuvent donc être associé à une opération et être gérées dans une

méthode spécifique. Ces méthodes se trouvent dans la classe **Server** pour la partie serveur, et dans la classe **SocialNetworkApp**, contenue dans le sous-dossier *SocialNetwork*, pour la partie client.

La partie serveur: Commençons par présenter la partie serveur de l'application. Elle est représentée par trois classes contenues dans le sous-dossier *SocialNetworkServer*:

- **Server**: qui représente la partie serveur de l'application et comprend les méthodes:
  - *initConnection*: permettant l'initialisation d'un socket TCP de bienvenue
  - *start*: contenant la boucle principale du serveur permettant la gestion des requêtes de connexion ainsi que l'envoi/réception de messages vers/depuis un client. Les messages sont des dictionnaires Python sérialisés grâce au module *pickle*.
  - *handleConnectionRequest*, *sendMessage*, *receiveMessageChunck*, *closeClientSocket*: qui sont uniquement utilisées par la méthode *start*.
  - *registerClientUsername*: permettant d'associer un socket ouvert pour un client au nom d'utilisateur qui a été entré lors de la connexion de l'utilisateur à l'application.
  - *getClientSocketByUsername*: permettant de récupérer le socket ouvert par un client sur base du nom d'utilisateur associé
  - *send*: permettant de préparer un message à l'envoi vers un socket, le message est envoyé automatiquement par la méthode *sendMessage* par après.
  - *handleRequestReceived*: permettant de gérer les messages reçus de la part d'un client. Comme expliqué plus haut, chaque type de message est renvoyé vers une méthode spécifique.
- **ClientSocket**: qui sert essentiellement à l'association des sockets avec les noms d'utilisateur.
- **MessageBuffer**: qui permet de correctement extraire chaque messages reçu de la part des client à partir du flux de bytes reçus au travers des sockets.

La partie client et l'interface graphique: La partie client de l'application, qui est lancée dans une interface graphique, a été implémentée en utilisant les modules *socket*, *select*, *pickle* et la plateforme *Kivy*. Elle est contenue dans le sous-dossier *SocialNetwork*. Commençons par décrire les éléments permettant de communiquer avec le serveur et de lancer l'application, il s'agit:

- Des trois classes contenues dans le paquet *Network*:
  - La classe **Client** permettant la connexion au serveur et contenant une boucle permettant l'envoi de messages vers le serveur grâce à sa méthode *send* et la réception de messages provenant du serveur. Il est important de noter que, lors de la réception d'un message, cette classe émet un signal accompagné du message reçu qui sont automatiquement réceptionné par la classe **SocialNetworkApp**.
  - La classe **DTO** (Data Transfer Object) permettant transformer les données envoyées au serveur sous la forme de dictionnaires Python qui sont ensuite sérialisés grâce au module *pickle* avant d'être envoyé vers le serveur.
  - La classe **MessageBuffer** permettant de correctement extraire chaque message reçu de la part des client à partir du flux de bytes reçus au travers des sockets.

- De la classe **SocialNetworkApp** (contenue dans le fichier main.py) permettant de lancer la boucle principale de l'interface graphique et contenant les méthodes permettant la gestion de chaque type de message reçu de la part du serveur. De plus cette classe s'occupe d'instancier l'objet *Client*, de lancer sa boucle dans un thread pour ne pas interférer avec la boucle de l'interface graphique et de le passer aux classes représentants les écrans de l'application lors de leur instanciation pour qu'elles puissent directement envoyer des messages vers le serveur. Enfin, cette classe permet la gestion des passages d'un écran à l'autre.

Les autres classes utilisées par cette partie de l'application étant trop nombreuses pour que l'on puisse les décrire chacune dans le détail, décrivons simplement les paquets dans lesquels elles sont contenues et leur utilité en une ligne, il s'agit des paquets:

- *Models*: reprenant les classes représentant les modèles utilisés par l'application:

Classe	Utilité
User	Représente un utilisateur simple
LoggedInUser	Représente un utilisateur connecté à l'application
Publication	Représente une publication
Comment	Représente un commentaire
Conversation	Représente une conversation entre deux utilisateurs

- *Screens*: contenant les quatre classes représentant les quatre écrans de l'application:

Classe	Utilité
WelcomeScreen	Écran contenant le champ de saisie de l'adresse IP du serveur
ProfileScreen	Écran contenant le profil d'un utilisateur
SignInScreen	Écran de connexion à l'application
SignUpScreen	Écran d'enregistrement à l'application

- *MainWidgets*: contenant les classes représentant les widgets principaux de l'application:

Classe	Utilité
PublicationsWidget	Widget contenant les publications de l'utilisateur
FriendsWidget	Widget contenant les amis et les demandes d'amis d'un utilisateur
AboutWidget	Widget contenant les informations personnelles de l'utilisateur
ActivityFeedWidget	Widget contenant le fil d'actualité
ProfileTabbedPanel	Panneau d'onglets contenant les quatre widgets précédents
MenuBarWidget	Widget contenant la barre de menu de l'application
InboxWidget	Widget contenant la boite de messagerie d'un utilisateur
ConversationWidget	Widget contenant une conversation entre deux utilisateurs
NotificationsWidget	Widget contenant les notifications de l'utilisateur
SearchResultsWidget	Widget contenant les résultats de la recherche d'un autre utilisateur
UserInteractionWidget	Widget contenant le bouton de choix de paramètre de privacité
PrivacySettingWidget	Widget contenant le choix du paramètre de privacité

- *Widgets*: contenant les classes représentant les widgets plus généraux contenus dans les widgets principaux du paquet *MainWidgets*:

Classe	Utilité
SignInFormWidget	Widget contenant le formulaire de connexion à l'application
SignUpFormWidget	Widget contenant le formulaire d'enregistrement à l'application
CustomBoxLayout	BoxLayout auquel on peut appliquer une couleur de fond
CustomFloatLayout	FloatLayout auquel on peut appliquer une couleur de fond
CustomScreen	Screen auquel on peut appliquer une couleur de fond
CustomScreenManager	ScreenManager auquel on peut appliquer une couleur de fond
CustomPopup	Popup auquel on peut assigner un message personnalisé
PublicationWidget	Widget contenant une publication
FeedWidget	Widget contenant une publication du fil d'actualité (commentable)
FriendRequestWidget	Widget contenant une demande d'ami
AboutLabel	Label contenant une information personnelle de l'utilisateur
TextSizeLabel	Label s'adaptant à la taille du texte qu'il contient
IconButton	Bouton auquel on peut appliquer un icône
ExtensibleScrollView	ScrollView s'adaptant à la taille cumulée des widgets contenus

4 Installation des plateformes et lancement du prototype

Décrivons dans cette section le travail qui a été effectué dans l'optique d'automatiser l'installation des plateformes *Kivy* et *Buildozer* ainsi que la marche à suivre pour le lancement du prototype sur n'importe quelle machine.

4.1 Installation des plateformes

Les installations de la plateforme *Kivy* et du logiciel *Buildozer* sont compliquées car elles demandent d'installer de nombreuses dépendances. De plus, chaque version de la plateforme *Kivy* dépend d'une version bien précise de la plateforme *Cython*.

Un script Bash a été écrit permettant de réaliser, en l'exécutant, l'installation des deux plateformes et de leurs dépendances automatiquement dans un système Ubuntu 16.04. Ce script et l'image d'une machine virtuelle créée en utilisant le logiciel VirtualBox dans laquelle l'installation a été réalisée sont disponibles dans le dossier *MiniMemoire*. Cette machine virtuelle contient un utilisateur auquel le mot *projet* a été assigné comme nom et comme mot de passe.

Pour pouvoir utiliser l'image de la machine virtuelle mise à disposition, il suffit d'installer le logiciel VirtualBox et d'y importer l'image.

Pour que la machine virtuelle reconnaisse un smartphone connecté via USB à la machine hôte, il est nécessaire de permettre à la machine virtuelle de détecter la connexion à l'appareil via USB en effectuant les opérations suivantes:

- Installer l'extension *Oracle VM VirtualBox Extension Pack*.
- Si le système hôte est un système Linux, l'utilisateur doit être ajouté au groupe vboxusers grâce à la commande suivante:

```
sudo adduser <user> vboxusers
```

Ensuite, redémarrer la machine ou se déconnecter et se reconnecter à sa session.

- Autoriser la détection de l'appareil connecté via USB, une fois la machine virtuelle démarrée, en cliquant sur l'onglet Devices, puis sur USB et enfin sur le nom de l'appareil que l'on souhaite autoriser.

Enfin, pour effectuer une nouvelle installation de *Kivy* et *Buildozer* dans un système Ubuntu 16.04 à l'aide su script Bash mis à disposition, les opérations suivantes doivent être effectuée:

- Exécuter le script
- Télécharger le fichier tar.xz contenant le NDK Crystax sur la page suivante: `https://www.crystax.net/en/download` et l'extraire dans le répertoire /home
- Remplacer le fichier Android.py contenu dans le dossier /home/<user>/buildozer/buildozer/targets/ par le fichier Android.py mis à disposition dans le dossier *MiniMemoire*.

4.2 Lancement du prototype

4.2.1 Lancement du serveur

Comme la partie serveur de l'application n'utilise que des modules standard de Python, elle peut être lancée directement en ligne de commande sur un système Linux, en utilisant des privilèges élevés car le serveur écoute sur le port 80 (Il est obligatoire d'utiliser un port plus petit ou égal à 1024 pour que le serveur fonctionne dans le réseau de l'ULB).

Il était prévu initialement de créer une machine virtuelle dans laquelle une adresse IP statique aurait été configurée. Malheureusement, comme le transfert d'adresse IP est bloqué dans le réseau de l'ULB, il est impossible d'utiliser le logiciel VirtualBox. En effet, le transfert d'adresse IP est utilisé lorsqu'on configure une interface réseau en mode Bridge pour une machine virtuelle en utilisant ce logiciel et ce mode est le seul permettant la création d'une adresse IP statique avec laquelle on pourrait communiquer depuis une autre machine. Une autre solution aurait été de configurer une adresse IP statique dans un conteneur Docker. Malheureusement, par manque de temps, cette configuration n'a pas pu être réalisée. Il est donc nécessaire d'indiquer manuellement l'adresse IP de la machine sur laquelle il est lancé lors du lancement du côté client de l'application.

4.2.2 Lancement du client sur ordinateur

La partie client de l'application peut être lancée directement en ligne de commande dans la machine virtuelle Ubuntu 16.04 présentée ci-dessus. Il est possible d'ajouter des paramètres à la commande pour lancer l'application dans une fenêtre de la taille d'un écran de smartphone. Par exemple:

```
python3 main.py -m screen:phone_samsung_galaxy_s6,portrait,scale=.4
```

4.2.3 Lancement du client sur smartphone

Le lancement de l'application sur un smartphone nécessite de l'empaqueter et de la déployer avec le logiciel *Buildozer*.

Premièrement, le débogage USB doit être autorisé dans les options de développement du smartphone. Les options de développement ne sont pas accessibles par défaut, il faut réaliser une manipulation propre à chaque version des systèmes Android et iOS, cette manipulation doit être trouvée en effectuant une recherche en fonction du système installé dans le smartphone.

Ensuite, il est nécessaire de créer un fichier buildozer.spec grâce à la commande `buildozer init`. Cette commande créée un fichier buildozer.spec, à éditer, qui contient des informations concernant l'application telles que le nom, la version et bien d'autres. Il est nécessaire de préciser que l'on utilise Python 3, en ajoutant python3crystax==3.5 à la ligne *requirements* et en précisant le chemin du dossier contenant le NDK CrystaX et sa version aux lignes, respectivement, *android.ndk_path* et *android.ndk*. De plus, il est nécessaire d'ajouter INTERNET à la ligne *android.permissions*. Un tel fichier pré-configuré est disponible dans le sous-dossier *SocialNetwork*.

Pour que Buildozer puisse reconnaître le fichier contenant le lancement de l'application, ce dernier doit obligatoirement être nommé main.py et se trouver dans le même dossier que le fichier buildozer.spec, dans le cas de ce prototype il s'agit du fichier contenant la classe **SocialNetworkApp**. L'empaquetage de l'application est réalisé en exécutant la commande `buildozer android debug` ou `buildozer ios debug`, en fonction du type de système d'exploitation du smartphone. Il est important de noter que la commande `buildozer ios debug` ne peut être exécutée que sur un système OSX, il est donc malheureusement impossible de l'utiliser dans le cadre de ce projet.

Enfin, une fois l'application empaquetée, elle peut être déployée sur le smartphone. Le smartphone doit être connecté à l'ordinateur via USB et la lecture/écriture de données doit être acceptée (un dialogue permettant de confirmer cette autorisation apparaît lors de la connexion du smartphone à l'ordinateur). Le déploiement est effectué en exécutant la commande `buildozer android deploy` ou être déployée et lancée directement en utilisant la commande `buildozer android deploy run`.

Lorsque l'application a été correctement déployée dans le smartphone, celle-ci peut être exécutée à tout moment, le smartphone ne doit plus être connecté à l'ordinateur.

5 Utilisation du code produit pour la rédaction d'un projet pour le cours de programmation

Le code produit lors de l'implémentation du prototype ne peut malheureusement pas être utilisé tel quel dans le but de rédiger un projet pour les étudiant en première année à cause de l'utilisation de classes. En restant dans l'optique d'utiliser le code pour la rédaction d'un projet, présentons, dans cette section, quelles parties du code pourrait être utilisé en le mettant à disposition des étudiants.

En premier lieu, les classes utilisées pour implémenter le modèle client-serveur de l'application devraient leur être fourni ainsi que les méthodes de la partie serveur permettant d'associer un socket au nom d'utilisateur correspondant au client pour lequel il a été ouvert. Il s'agit donc des classes **Client** et **MessageBuffer** pour la partie client et **Server** de laquelle les méthodes permettant de gérer les messages transférés auraient été retirées mis à part les méthodes correspondant aux opérations d'enregistrement et de connexion à l'application (il serait nécessaire de fournir ces deux méthodes car elles sont indispensables pour l'association d'un socket à un nom d'utilisateur), **ClientSocket** et **MessageBuffer** pour la partie serveur.

Comme les méthodes correspondant aux opérations d'enregistrement et de connexion à l'application devraient être fournies, le code de la partie client de l'application correspondante devrait l'être aussi. Il s'agit des écrans de connexion et d'enregistrement à l'application implémentés dans les classes **SignInScreen** et **SignUpScreen**. Similairement, le code correspondant à l'enregistrement des données correspondantes dans la base de données devrait aussi être fourni, ainsi que le code permettant de créer la base de données, il s'agit de méthodes de la classe **DTO**.

Enfin, pour que le code fourni puisse être fonctionnel, il serait alors obligatoire de fournir aux étudiant l'écran permettant d'indiquer l'adresse IP du serveur implémenté dans la classe **WelcomeScreen** ainsi que le code permettant de lancer l'application, de lancer la boucle du côté client dans un thread et d'instancier les trois écrans se trouvant dans la classe **SocialNetworkApp**.

Il pourrait donc être demandé aux étudiant, après leur avoir fourni cette base, d'implémenter les fonctionnalités restantes reprises dans la section 1.3.

Conclusion

En conclusion, il est important de noter qu'on a constaté, après de nombreuses recherches, qu'il n'existe, au moment de la rédaction de ce rapport, aucune plateforme plus facile d'utilisation ou plus complète que la plateforme *Kivy* permettant d'implémenter une interface graphique portable sur Android et iOS en Python. L'utilisation de cette plateforme comporte malgré tout certaines difficultés. En effet, bien qu'étant complète et bien conçue, elle est peu utilisée, ce qui rend son utilisation fastidieuse par manque d'explications et d'exemples disponibles sur l'Internet. Dans la plupart des cas, lors d'une question sur son utilisation, il n'est possible de se référer qu'à sa documentation officielle, qui peut prendre du temps à être comprise. Similairement, l'installation de cette plateforme, couplée à l'installation du logiciel *Buildozer* et, implicitement, de la plateforme *Cython*, pose des difficultés, comme on a pu le voir plus haut. Pour cette raison, la rédaction d'une script permettant d'automatiser ces installations nous a paru indispensable. De plus, on a constaté, après de nombreux essais, que la mise en place d'un mécanisme permettant de configurer une adresse IP statique, dans un réseau sécurisé tel que celui de l'ULB est, elle aussi, très fastidieuse. Malgré ces difficultés, un prototype complètement fonctionnel de réseau social portable sur Android et iOS écrit en Python 3 et comportant les fonctionnalités requises a été implémenté, l'adresse IP du serveur doit simplement être indiquée manuellement lors de son lancement. Comme ce prototype comporte tous les exemples d'utilisation de *Kivy* nécessaires pour le développement d'un réseau social, il pourra être utilisé ultérieurement comme modèle pour la rédaction d'un projet pour les étudiants du cours de programmation de la première année du grade de Bachelier en sciences informatique.