

## Langage de programmation cible :

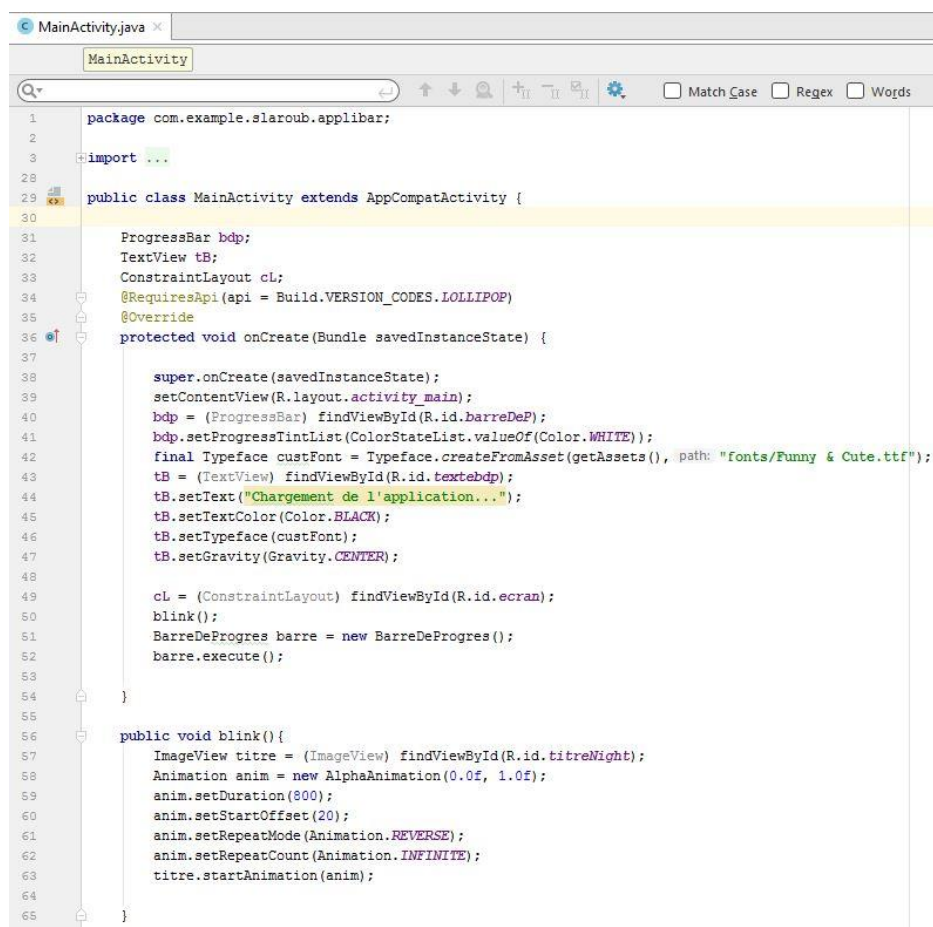
Afin de répondre aux besoins énoncés pour le développement de cette application, nous avons décidé de nous orienter vers le développement en ANDROID. En effet, il nous semblait d'une part judicieux pour la planification d'une soirée entre amis et un souci de convivialité de pouvoir avoir directement l'application à portée de main sur son téléphone ou sa tablette.

Les utilisateurs pourront être derrière le même téléphone et pourront ainsi fournir les différentes informations nécessaires et s'ils le souhaitent, apporter des modifications au plan de la soirée obtenue.

D'autre part, nous avons donné la priorité à une application ANDROID par rapport à un site web par exemple, car une application ne nécessite pas d'hébergement sur un serveur, le .APK est directement disponible sur téléphone ou tablette une fois téléchargé.

Un autre point qui a orienté notre choix vers le développement en ANDROID, est qu'Android Runtime ou ART (avant Android 5.0 « LOLLIPOP », Dalvik) est une machine virtuelle qui exécute du code JAVA, un langage que nous avons déjà pu aborder au cours de notre formation et auquel nous sommes plus ou moins familiers. Pour développer notre projet, nous avons décidé d'utiliser l'IDE AndroidStudio, développé par JetBrains, qui selon nous est plus intuitif et facile d'utilisation qu'Eclipse.

Une classe en ANDROID se présente toujours de la façon suivante : **Un fichier .java**



```
1 package com.example.slaroub.applibar;
2
3 import ...
4
28
29 public class MainActivity extends AppCompatActivity {
30
31     ProgressBar bdp;
32     TextView tB;
33     ConstraintLayout cL;
34     @RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
35     @Override
36     protected void onCreate(Bundle savedInstanceState) {
37
38         super.onCreate(savedInstanceState);
39         setContentView(R.layout.activity_main);
40         bdp = (ProgressBar) findViewById(R.id.barreDeP);
41         bdp.setProgressTintList(ColorStateList.valueOf(Color.WHITE));
42         final Typeface custFont = Typeface.createFromAsset(getAssets(), "fonts/Funny & Cute.ttf");
43         tB = (TextView) findViewById(R.id.textebdp);
44         tB.setText("Chargement de l'application...");
45         tB.setTextColor(Color.BLACK);
46         tB.setTypeface(custFont);
47         tB.setGravity(Gravity.CENTER);
48
49         cL = (ConstraintLayout) findViewById(R.id.ecran);
50         blink();
51         BarreDeProgres barre = new BarreDeProgres();
52         barre.execute();
53     }
54
55
56     public void blink(){
57         ImageView titre = (ImageView) findViewById(R.id.titreNight);
58         Animation anim = new AlphaAnimation(0.0f, 1.0f);
59         anim.setDuration(800);
60         anim.setStartOffset(20);
61         anim.setRepeatMode(Animation.REVERSE);
62         anim.setRepeatCount(Animation.INFINITE);
63         titre.startAnimation(anim);
64     }
65 }
```

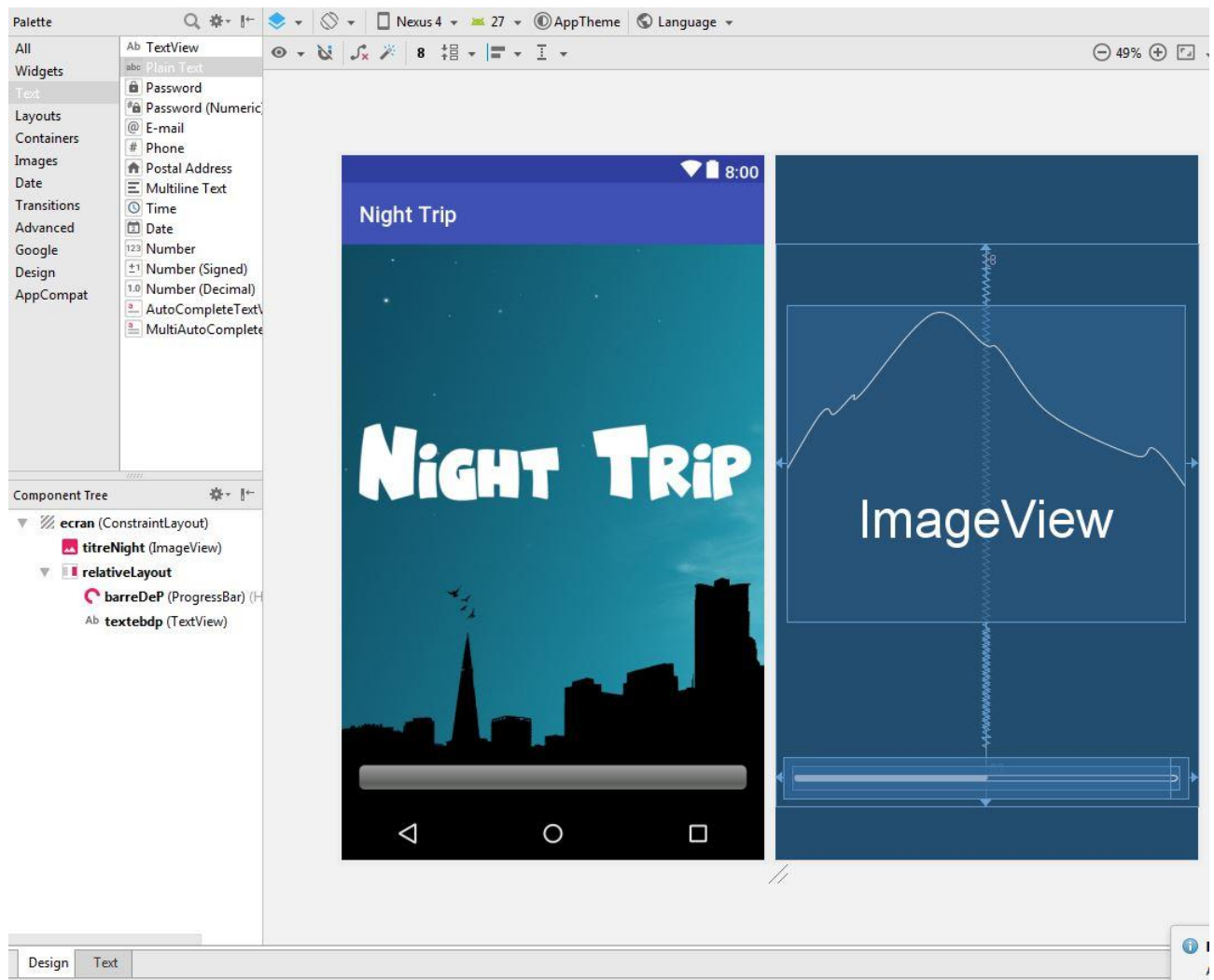
Celui-ci, définira le comportement de l'activité, ses méthodes, ses variables etc.

## Un fichier .xml

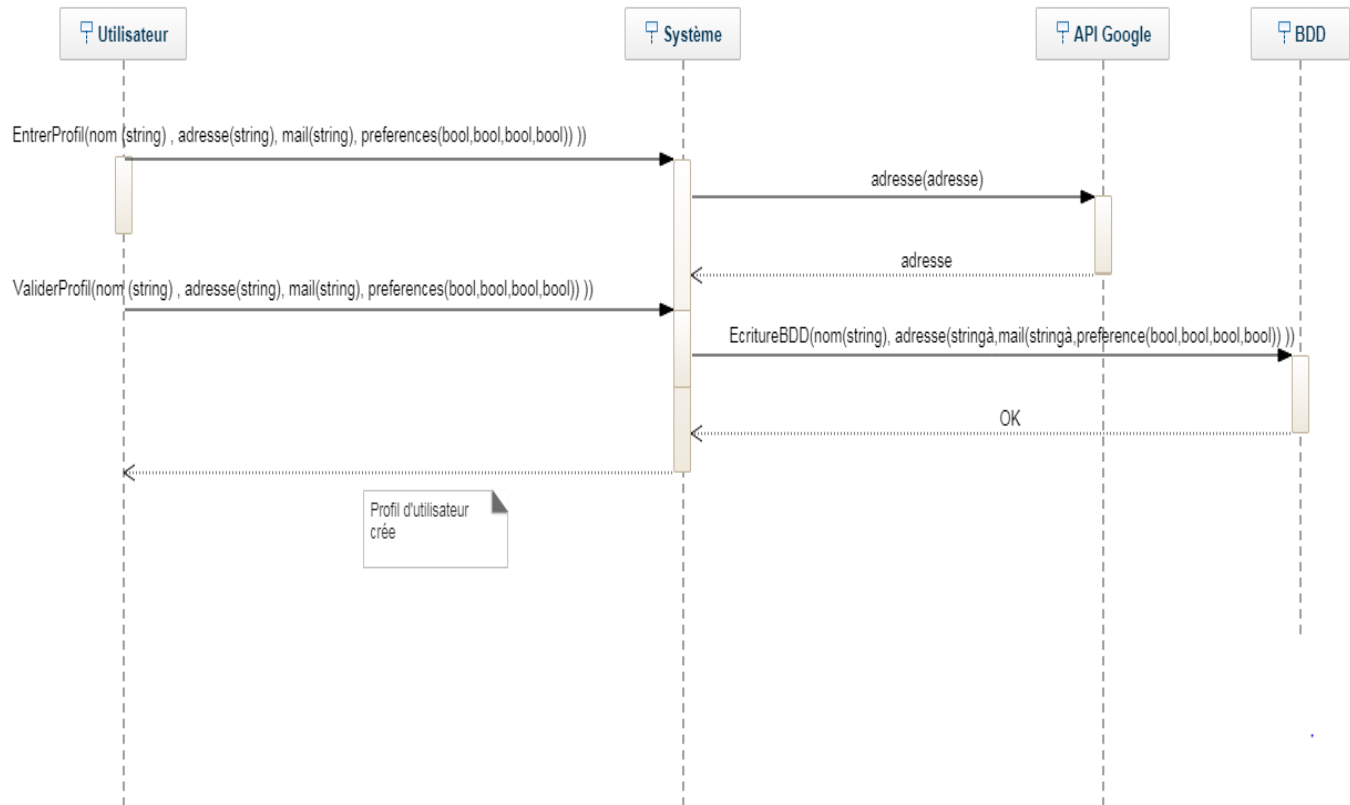
```
activity_main.xml x
android.support.constraint.ConstraintLayout
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:id="@+id/ecran"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      android:background="@drawable/ville"
9
10     tools:context="com.example.slaroub.applibar.MainActivity">
11
12
13     <ImageView
14         android:id="@+id/titreNight"
15         android:layout_width="362dp"
16         android:layout_height="288dp"
17         android:layout_alignLeft="@+id/progres"
18         android:layout_alignParentBottom="true"
19         android:layout_alignStart="@+id/progres"
20         android:layout_marginBottom="63dp"
21         app:layout_constraintBottom_toBottomOf="parent"
22         app:layout_constraintEnd_toEndOf="parent"
23         app:layout_constraintStart_toStartOf="parent"
24         app:layout_constraintTop_toTopOf="parent"
25         app:layout_constraintVertical_bias="0.342"
26         app:srcCompat="@drawable/tinight" />
27
28
29     <RelativeLayout
30         android:id="@+id/relativeLayout"
31         android:layout_width="368dp"
32         android:layout_height="38dp"
33         android:layout_marginBottom="8dp"
34         android:layout_marginEnd="8dp"
35         android:layout_marginStart="8dp"
36         android:layout_marginTop="8dp"
37         app:layout_constraintBottom_toBottomOf="parent"
38         app:layout_constraintEnd_toEndOf="parent"
39         app:layout_constraintHorizontal_bias="0.0"
40         app:layout_constraintStart_toStartOf="parent"
41         app:layout_constraintTop_toTopOf="parent"
42         app:layout_constraintVertical_bias="1.0">
43
44     <ProgressBar
45         android:id="@+id/barreDeP"
46         style="@android:style/Widget.ProgressBar.Horizontal"
```

Ce fichier représente la vue de notre activité, il nous permet de placer des containers, qui viendront structurer cette vue ainsi que des éléments tels qu'un champ de saisie ou bien une liste déroulante ou encore des boutons à placer dans ces containers.

De plus, AndroidStudio offre la possibilité d'utiliser le Drag and Drop afin de faciliter le développement et la structuration des différents écrans de notre application. Si l'on souhaite par exemple un champ de saisie au milieu de notre écran, il suffira de sélectionner dans la liste un « EditText » de glisser celui-ci sur l'interface « Design » d'AndroidStudio, et l'IDE se chargera de générer le code xml correspondant.



## Création d'un profil d'utilisateur correct.



Afin de créer un profil d'utilisateur, l'utilisateur va donner les informations relatifs au profil tel que le nom en string, l'adresse mail en string également, ses préférences en restaurant correspondant à des booléens ( Avec des types de restaurants par défaut qu'il faut cocher) et son adresse en string.

Au moment de l'entrée de l'adresse, le système va faire appel à l'API Google afin de faire correspondre le début de l'adresse rentrée avec une adresse qui existe et la finir par auto-complétion.

Ensuite l'utilisateur va appuyer sur le bouton de validation ce qui causera la vérification des valeurs entrées. Si ces valeurs sont correctes, le système va les envoyer vers la base de données afin de les stocker et ainsi le profil d'utilisateur sera créé.

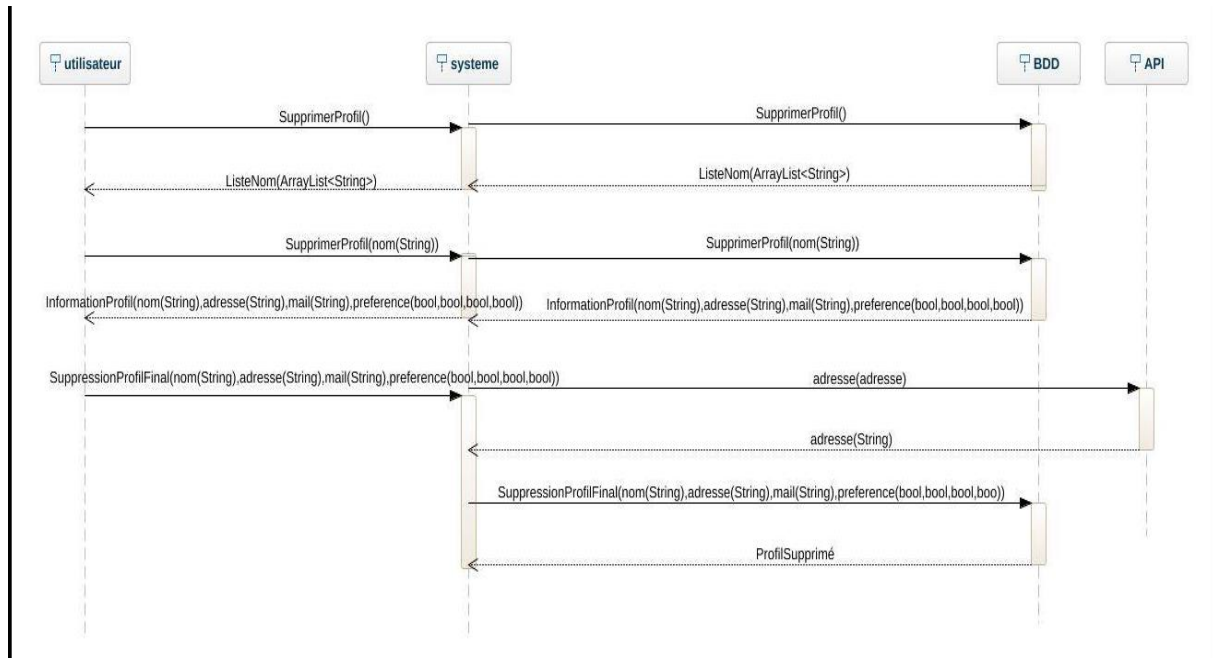
## Création d'un profil d'utilisateur incorrect.



La création d'un profil incorrecte reprend les mêmes bases que précédemment, l'utilisateur va entrer les informations et c'est en appuyant sur la touche valider que le système va regarder si les champs entrés sont bons.

Dans le cas contraire, cela renvoie un message d'erreur à l'utilisateur.

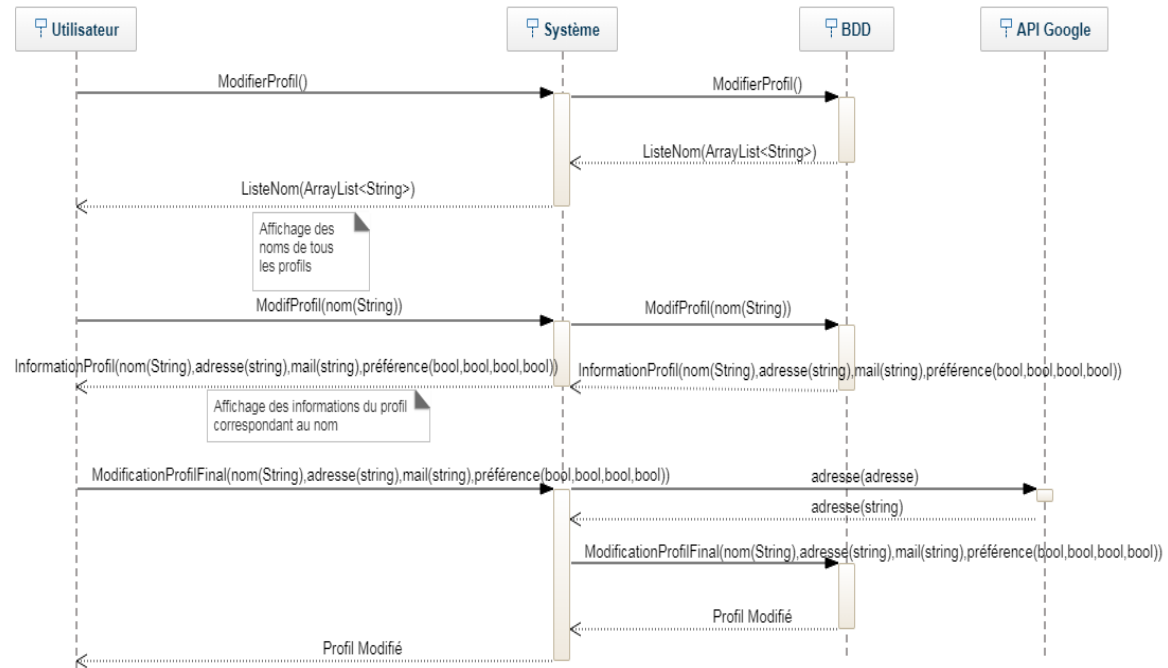
## Suppression d'un profil



Pour la suppression d'un profil, l'utilisateur va tout d'abord entrer dans l'écran de modification de profils ce qui a pour effet que le système demande à la base de donnée de lui sortir tous les noms se trouvant dans celle-ci sous forme de liste.

Ensuite l'utilisateur va choisir un profil à supprimer et ensuite valider ce qui va donner le signal au système pour effectuer la suppression du profil.

## Modification d'un profil

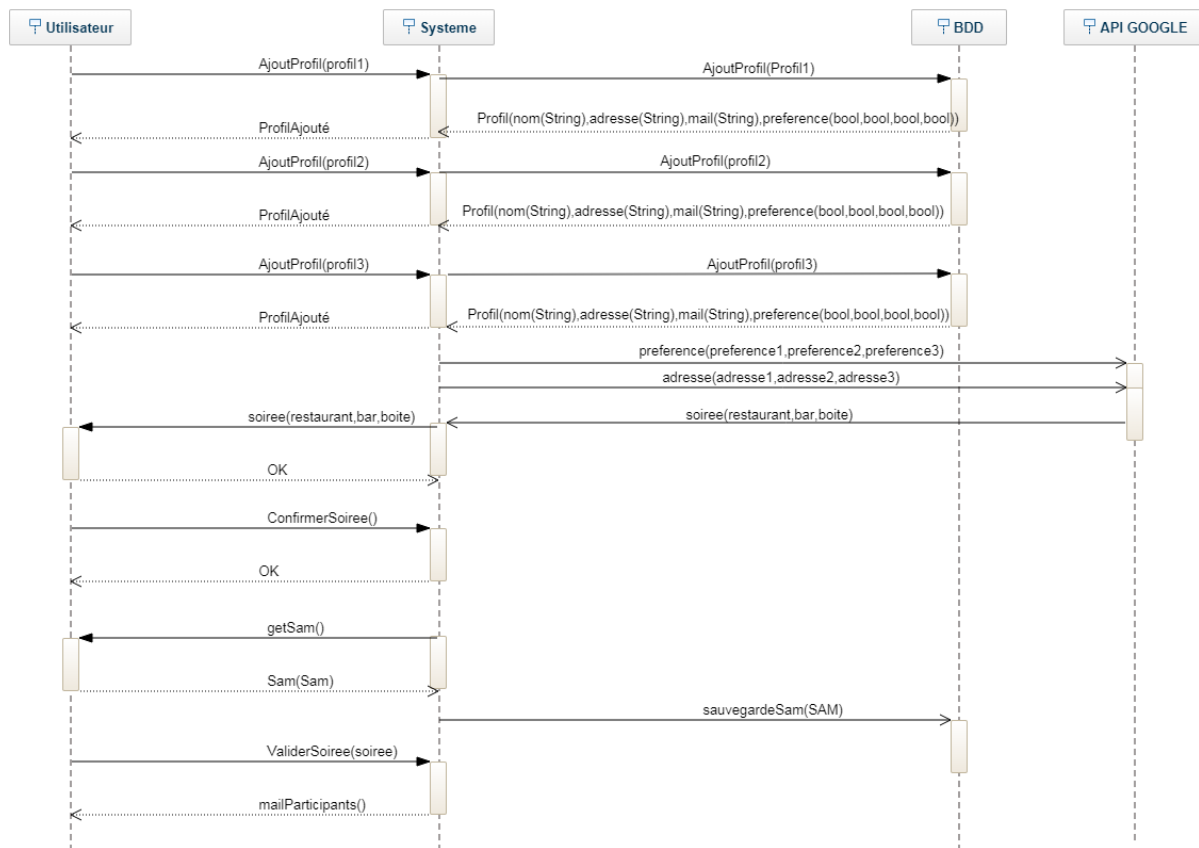


Pour la modification d'un profil, l'utilisateur va tout d'abord entrer dans l'écran de modification des profils ce qui va faire que le système demande à la base de données tous les noms se trouve dans celle-ci sous forme de liste.

Ensuite l'utilisateur va choisir un profil à modifier, ce qui va faire que le système va faire une requête à la base de donnée pour le profil choisi, celle-ci va lui donner toutes les informations qu'elle a sur le profil et tout ceci sera affiché à l'utilisateur.

L'utilisateur va donc ensuite modifier une des valeurs du profils, et similairement à la création de profil, le système va soumettre tout cela à la base de données. Le profil sera donc modifié.

## Création d'une soirée

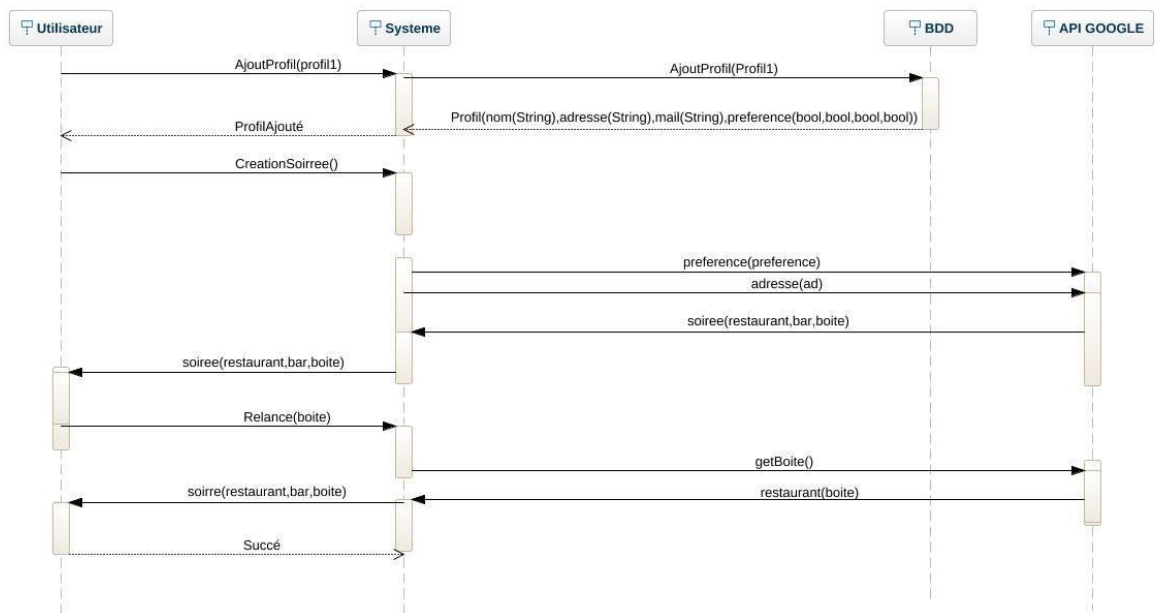
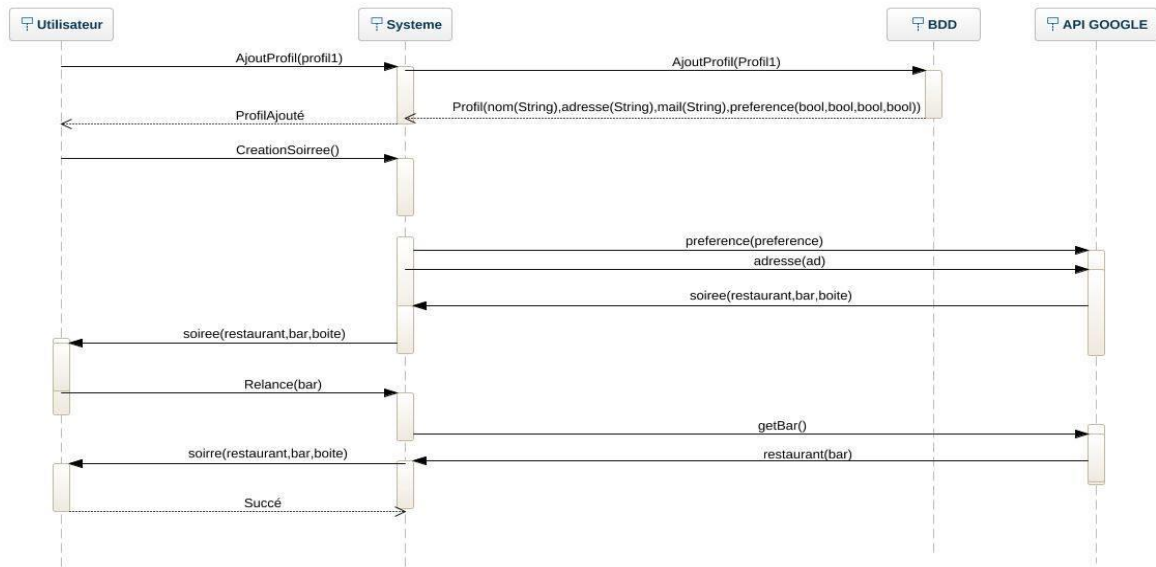


Pour la création d'une soirée, l'utilisateur va tout d'abord aller dans l'écran de création de soirée et choisir un profil. Le système va effectuer une requête à la base de donnée afin de récupérer toutes les informations de ce profil qui va ensuite ranger dans un tableau. Il y a ensuite un message de confirmation à l'utilisateur qui indique que le profil a été ajouté.

Cela va être effectué 1-2 ou 3 fois en fonction du nombre de participant à la soirée. Une fois tous les profils ajoutés, l'utilisateur va indiquer au système de générer la soirée ce qui aura pour effet de prendre en compte les paramètres des profils et d'envoyer une requête à l'API Google afin qu'elle nous ressorte un triplet contenant le restaurant, le bar et la boîte. L'utilisateur va ensuite confirmer la soirée et cela va ainsi désigner un Sam et effectuer l'envoi des mails.



## Relance de lieux





Voyons maintenant le cas où l'utilisateur n'est pas satisfait d'un des lieux proposé par le système.

On a donc toute la procédure qui va générer la soirée à partir des profils sélectionnés et ainsi le système nous génère un premier triplet de lieux.

C'est là que l'utilisateur va pouvoir soit valider et donc accepter le triplet (comme le cas précédent) ou ici effectuer une relance d'un des lieux proposés.

S'il décide d'effectuer une relance, le système va redemander à l'API Google un lieu différent qu'il va de nouveau proposer à l'utilisateur.

Ces relances peuvent s'effectuer autant de fois que l'utilisateur le veut ou jusqu'à que l'utilisateur décide de ne pas créer de soirée.

Une fois que l'utilisateur accepte un triplet de lieu proposé, la création de la soirée est un succès.