



# Programmation mobile

## Compte-rendu TP1

### Les bases d'Android

Louis Jean  
Master 1 IMAGINE  
Université de Montpellier  
N° étudiant : 21914083  
<https://github.com/louis-jean0/HAI811I-mobile>

3 mars 2024

## Table des matières

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>2</b>  |
| <b>2</b> | <b>Une première application - Interface simple</b>              | <b>3</b>  |
| 2.1      | Vue XML . . . . .   | 3         |
| 2.2      | Vue Java . . . . .  | 5         |
| 2.3      | Internationalisation des interfaces . . . . .                   | 7         |
| 2.4      | Événements associés aux objets graphiques d'une vue . . . . .   | 9         |
| 2.5      | Intent explicite . . . . .                                      | 11        |
| 2.6      | Intent implicite . . . . .                                      | 13        |
| <b>3</b> | <b>Application simple pour consulter les horaires de trains</b> | <b>16</b> |
| <b>4</b> | <b>Application simple d'agenda</b>                              | <b>17</b> |
| <b>5</b> | <b>Conclusion</b>   | <b>18</b> |

# 1 Introduction

Ce travail pratique vise à introduire les bases du développement d'applications mobiles sur la plateforme Android. À travers une série d'exercices progressifs, de la création d'une simple application "Hello World" à la mise en place d'applications plus complexes permettant la gestion d'un agenda et la consultation d'horaires de trains, ce TP offre une exploration concrète de l'environnement de développement Android Studio, incluant la manipulation d'intents explicites et implicites, l'internationalisation des interfaces, et la gestion d'événements associés aux objets graphiques.

**N.B :** les deux premiers exercices ne figurent pas dans ce compte-rendu car ils n'ont pour but que de mettre en place l'environnement Android Studio.

## 2 Une première application - Interface simple

Pour des raisons de praticité, j'ai regroupé les exercices 3 à 7 en un seul et même projet Android Studio, car chacun de ces exercices itère sur le précédent.

### 2.1 Vue XML

Dans un premier temps, j'ai réalisé la vue uniquement en XML, en ne modifiant que `activity_main.xml`. J'ai gardé le code Java de base, et j'ai ajouté mes éléments comme ceci :

```
1 <LinearLayout
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:orientation="vertical"
5     android:padding="16dp">
6
7     <TextView
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content"
10        android:text="Nom"/>
11    <EditText
12        android:id="@+id/nom"
13        android:layout_width="match_parent"
14        android:layout_height="wrap_content"/>
15
16    . . .
```

J'ai ensuite créé le fichier `ids.xml` dans `res` pour déclarer des identifiants uniques pour les éléments de l'UI.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <item name="nom" type="id"/>
4     <item name="prenom" type="id"/>
5     <item name="age" type="id"/>
6     <item name="domaine" type="id"/>
7     <item name="telephone" type="id"/>
8     <item name="valider" type="id"/>
9 </resources>
```



Figure 1: Vue intégralement XML

## 2.2 Vue Java

Pour créer la vue entièrement en Java, j'ai modifié le code de **MainActivity.java**.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4
5      LinearLayout layout = new LinearLayout(this);
6      layout.setOrientation(LinearLayout.VERTICAL);
7      layout.setLayoutParams(new LinearLayout.LayoutParams(
8          LinearLayout.LayoutParams.MATCH_PARENT, LinearLayout.LayoutParams.MATCH_PARENT));
9      layout.setPadding(16, 16, 16, 16);
10
11     // Nom
12     TextView nomLabel = new TextView(this);
13     nomLabel.setText("Nom");
14     layout.addView(nomLabel);
15     EditText nomInput = new EditText(this);
16     nomInput.setId(R.id.nom);
17     layout.addView(nomInput);
18
19     // Prénom
20     TextView prenomLabel = new TextView(this);
21     prenomLabel.setText("Prénom");
22     layout.addView(prenomLabel);
23     EditText prenomInput = new EditText(this);
24     prenomInput.setId(R.id.prenom);
25     layout.addView(prenomInput);
26
27     ...
28
29     setContentView(layout);
30 }
```

De cette manière, tout les éléments de l'UI sont directement créés dans le code Java, et on peut les manipuler à partir d'ici.



Figure 2: Vue intégralement Java

## 2.3 Internationalisation des interfaces

En créant un dossier **values-en** dans **res**, Android Studio reconnaît le code de la langue, et crée directement un fichier **strings.xml** dédié à l'anglais. Dans ce fichier, on remplace le texte en dur par des références aux ressources de chaînes. On fait de même mais en français pour le fichier **strings.xml** situé dans **values**.

```
1 <resources>
2     <string name="app_name">Exo3_4_5_6_7</string>
3     <string name="label_nom">Name</string>
4     <string name="label_prenom">First Name</string>
5     <string name="label_age">Age</string>
6     <string name="label_domaine">Field of Expertise</string>
7     <string name="label_telephone">Phone Number</string>
8     <string name="action_valider">Validate</string>
9 </resources>
```

On ajuste ensuite le layout pour changer le contenu des attributs **android:text** par les références aux chaînes, comme ceci par exemple :

```
1 <TextView
2     android:layout_width="wrap_content"
3     android:layout_height="wrap_content"
4     android:text="@string/label_prenom"/>
```

Par défaut, la langue de l'appareil émulé est l'anglais, donc désormais lorsque l'on lance l'application, on obtient le texte en anglais. Si l'on change la langue par français dans les paramètres, on retombe bien sur l'application vue plus haut.

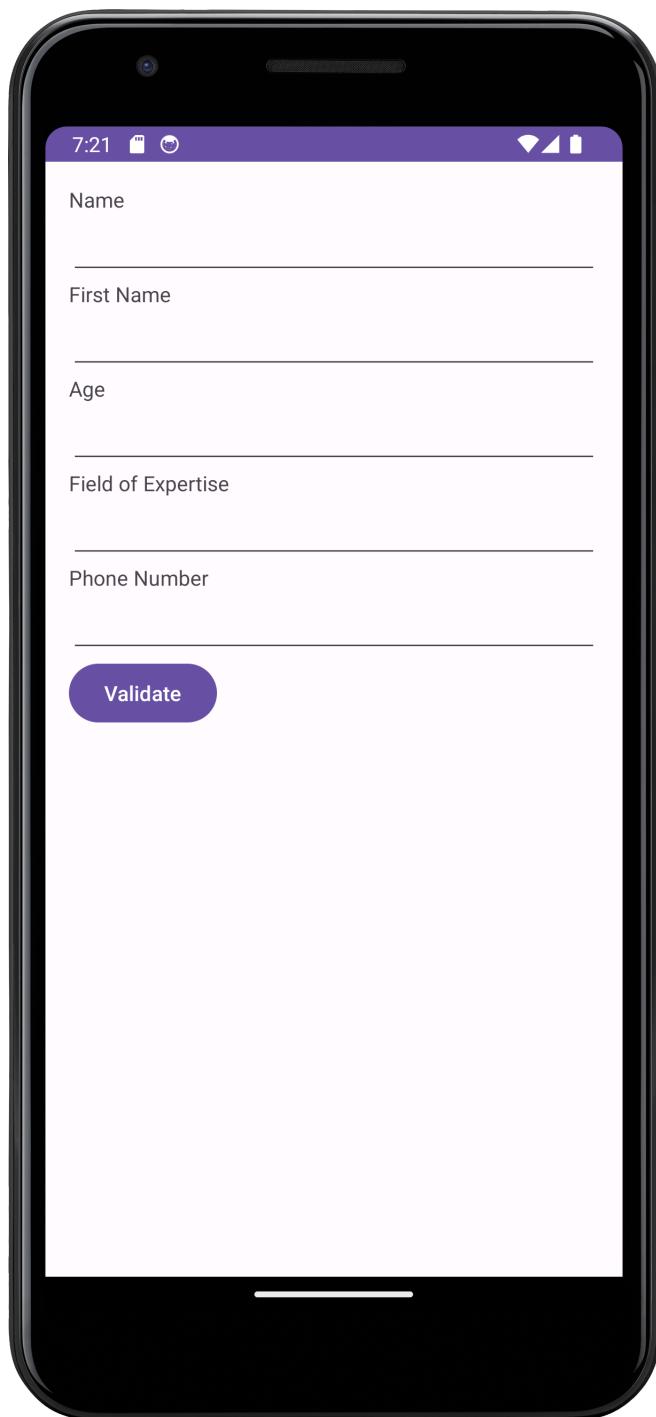


Figure 3: Internationalisation de l'application

## 2.4 Événements associés aux objets graphiques d'une vue

L'objectif ici est associer au bouton de validation un événement qui permet d'ouvrir une fenêtre de dialogue invitant l'utilisateur à confirmer ou à annuler la validation, puis qui change la couleur du fond des zones d'édition.

On commence par ajouter un écouteur d'événements (**OnClickListener**) au bouton de validation, dans la méthode **onCreate**. Cet écouteur va déclencher l'affichage d'une fenêtre de dialogue.

```
1 Button validerButton = findViewById(R.id.valider);
2 validerButton.setOnClickListener(new View.OnClickListener() ←
3     ↪ {
4         @Override
5         public void onClick(View v) {
6             afficherDialogueConfirmation();
7         }
8     });
9 }
```

Ensuite, on écrit la méthode **afficherDialogueConfirmation** qui va créer et afficher une fenêtre de dialogue.

```
1 private void afficherDialogueConfirmation() {
2     AlertDialog.Builder builder = new AlertDialog.Builder(←
3         ↪ MainActivity.this);
4     builder.setTitle("Confirmation");
5     builder.setMessage("Êtes-vous sûr de vouloir valider ?");
6     builder.setPositiveButton("Confirmer", new ←
7         ↪ DialogInterface.OnClickListener() {
8             @Override
9             public void onClick(DialogInterface dialog, int ←
10                ↪ which) {
11                 changerCouleurFondEditTexts(Color.GREEN);
12             }
13         });
14     builder.setNegativeButton("Annuler", new DialogInterface←
15         ↪ .OnClickListener() {
16             @Override
17             public void onClick(DialogInterface dialog, int ←
18                ↪ which) {
19                 dialog.dismiss();
20             }
21         });
22     AlertDialog dialog = builder.create();
23     dialog.show();
24 }
```

Enfin, on code la méthode **changerCouleurFondEditTexts** qui va parcourir tous les **EditText** et changer leur couleur de fond.

```
1 private void changerCouleurFondEditTexts(int couleur) {  
2     EditText nom = findViewById(R.id.nom);  
3     EditText prenom = findViewById(R.id.prenom);  
4     EditText age = findViewById(R.id.age);  
5     EditText domaine = findViewById(R.id.domaine);  
6     EditText telephone = findViewById(R.id.telephone);  
7  
8     nom.setBackgroundColor(couleur);  
9     prenom.setBackgroundColor(couleur);  
10    age.setBackgroundColor(couleur);  
11    domaine.setBackgroundColor(couleur);  
12    telephone.setBackgroundColor(couleur);  
13 }
```

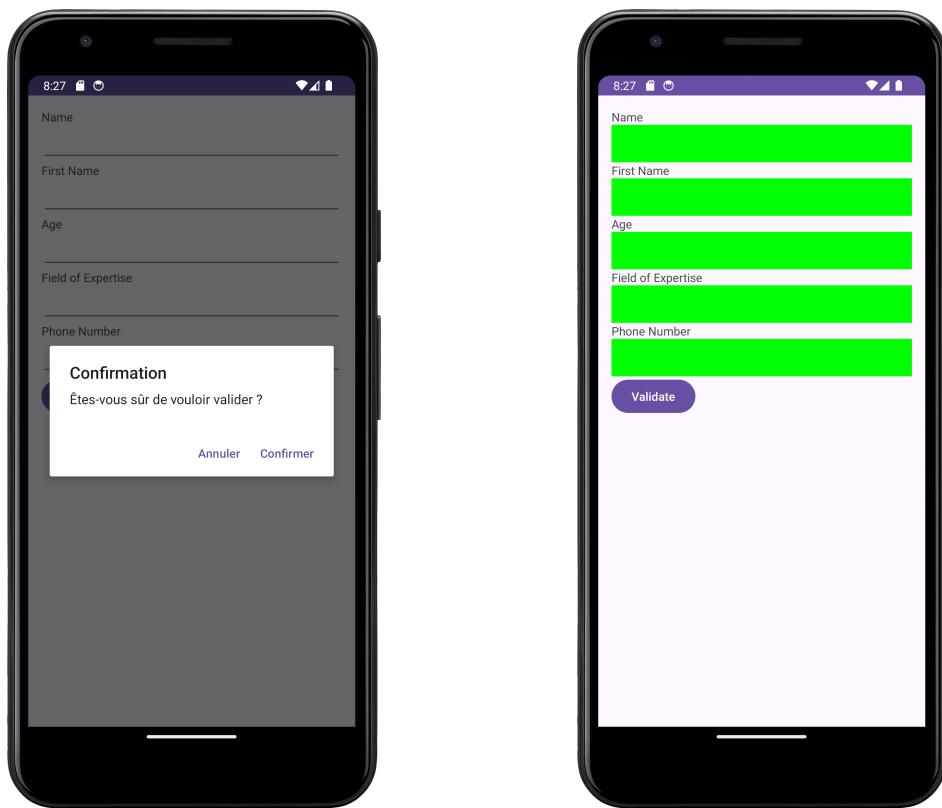


Figure 4: Évènement sur le bouton de validation

## 2.5 Intent explicite

Le but de cet exercice est de créer un **Intent** permettant de récupérer toutes les informations des champs saisis et de lancer une nouvelle activité via cet Intent. Cette nouvelle activité récupère les données saisies, les affiche et affiche deux boutons. Le premier bouton "OK" doit lancer une troisième activité, et le deuxième bouton "Retour" doit permettre de revenir à l'activité précédente.

Premièrement, j'ai créé une nouvelle activité nommée **AffichageActivity**. Dans le layout de cette activité, j'ai créé un bouton "OK", un bouton "Retour", et un champ "informations" qui servira à afficher les informations récupérées. J'ai ensuite créé une troisième activité que j'ai appelée **TroisiemeActivity**, que j'ai simplement configurée pour qu'elle affiche "Bienvenue sur la troisième activité" lorsqu'elle est lancée. Dans la méthode **onCreate** de **AffichageActivity**, voici ce que j'ai écrit pour récupérer et afficher les informations.

```
1 TextView textViewInfos = findViewById(R.id.textViewInfos);
2
3     Button btnOK = findViewById(R.id.btnOK);
4     Button btnRetour = findViewById(R.id.btnRetour);
5     // Récupérer les données de l'intent
6     Intent intent = getIntent();
7     String informations = "Nom: " + intent.getStringExtra("nom") + "\n" +
8         "Prénom: " + intent.getStringExtra("prenom") + "\n" +
9         "Âge: " + intent.getStringExtra("age") + "\n" +
10        "Domaine: " + intent.getStringExtra("domaine") + "\n" +
11        "Téléphone: " + intent.getStringExtra("telephone");
12
13     textViewInfos.setText(informations);
```

Si l'on appuie sur le bouton "OK", alors on passe à la troisième activité.

```
1 btnOK.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View view) {
4         Intent intent = new Intent(AffichageActivity.this, ←
5             TroisiemeActivity.class);
6         startActivity(intent);
7     }
8});
```

Si l'on appuie sur le bouton "Retour", alors on revient à l'activité précédente.

```
1 btnRetour.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View view) {
4         finish();
5     }
6 }) ;
```

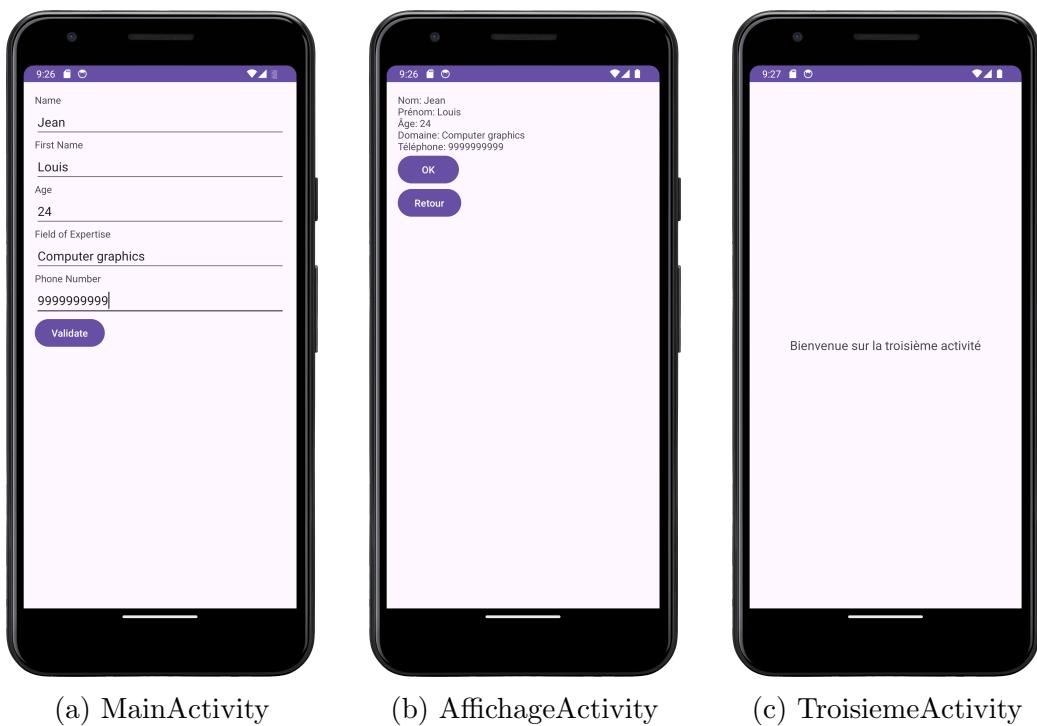


Figure 5: Navigation entre les activités à travers un Intent explicite

## 2.6 Intent implicite

Cet exercice vise à améliorer **TroisiemeActivity** grâce à un **Intent** implicite en y ajoutant une image de téléphone, le numéro de téléphone saisi depuis **MainActivity** et un bouton "Appeler". L'activation de ce bouton doit permettre de lancer un appel téléphonique vers ce numéro.

Pour ce faire, j'ai commencé par modifier le layout de **TroisiemeActivity**. J'ai rajouté une **ImageView** qui contient une image de téléphone directement fournie par Android Studio. J'ai aussi centré les éléments de la vue, et créé un bouton pour lancer l'appel.

```
1 <LinearLayout
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:orientation="vertical"
5     android:padding="16dp"
6     android:gravity="center">
7
8     <ImageView
9         android:id="@+id/phoneIcon"
10        android:layout_width="wrap_content"
11        android:layout_height="wrap_content"
12        android:src="@android:drawable/sym_action_call" />
13
14     <TextView
15         android:id="@+id/phoneNumberText"
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:textSize="18sp"
19         android:layout_marginTop="16dp"
20         android:text="Numéro de téléphone" />
21
22     <Button
23         android:id="@+id/callButton"
24         android:layout_width="wrap_content"
25         android:layout_height="wrap_content"
26         android:layout_marginTop="24dp"
27         android:text="Appeler" />
28 </LinearLayout>
```

Ensuite, j'ai accordé la permission d'appeler des numéros de téléphone à l'application, depuis le manifest.

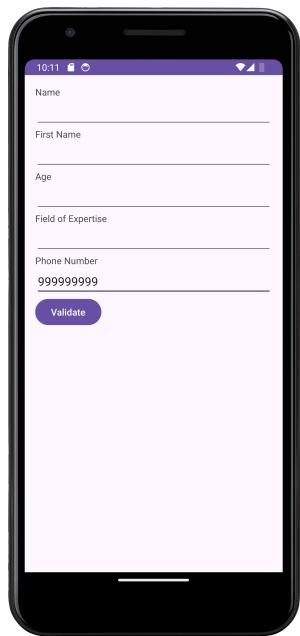
```
1 <uses-permission android:name="android.permission.CALL_PHONE" />
```

Par la suite, j'ai modifié le comportement du bouton "OK" depuis **AffichageActivity**, car désormais il doit passer le numéro de téléphone à **TroisiemeActivity**.

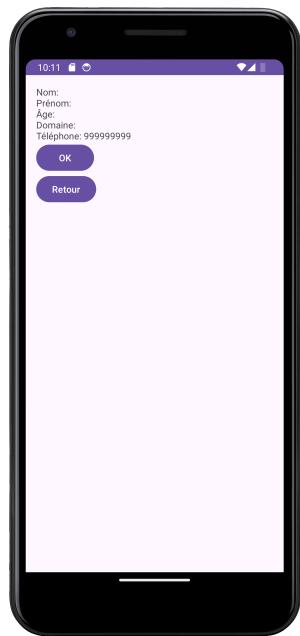
```
1 btnOK.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View view) {
4         Intent intent = new Intent(AffichageActivity.this, ↵
5             TroisiemeActivity.class);
6         String telephone = getIntent().getStringExtra("↵
7             telephone");
8         intent.putExtra("telephone", telephone);
9         startActivity(intent);
10    }
11 });
12 }
```

Enfin, j'ai modifié la méthode **onCreate** de **TroisiemeActivity**, afin de récupérer le numéro de téléphone puis d'utiliser le système du téléphone pour lancer l'appel.

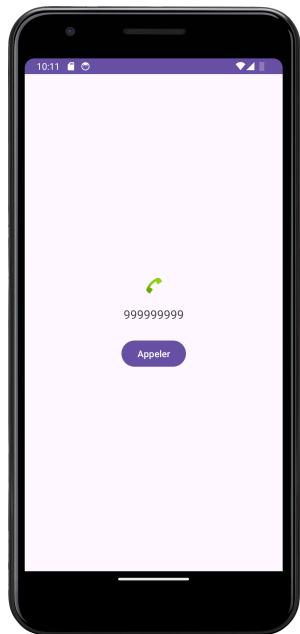
```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3     super.onCreate(savedInstanceState);
4     setContentView(R.layout.activity_troisieme);
5     String phoneNumber = getIntent().getStringExtra("↵
6         telephone");
7     System.out.println(phoneNumber);
8     TextView phoneNumberText = findViewById(R.id.↵
9         phoneNumberText);
10    phoneNumberText.setText(phoneNumber);
11    findViewById(R.id.callButton).setOnClickListener(view ->↵
12        {
13            Intent callIntent = new Intent(Intent.ACTION_CALL);
14            callIntent.setData(Uri.parse("tel:" + phoneNumber));
15            if (ContextCompat.checkSelfPermission(↵
16                TroisiemeActivity.this, Manifest.permission.CALL_PHONE↵
17                ) == PackageManager.PERMISSION_GRANTED) {
18                startActivity(callIntent);
19            } else {
20                ActivityCompat.requestPermissions(↵
21                    TroisiemeActivity.this, new String[]{Manifest.↵
22                        permission.CALL_PHONE}, 1);
23            }
24        });
25    }
26 }
```



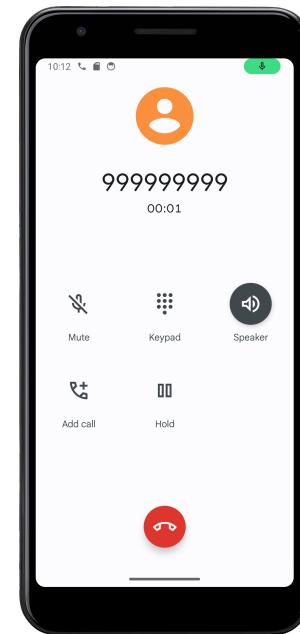
(a) Saisie du numéro de téléphone



(b) Affichage des informations



(c) Bouton pour appeler le numéro précédemment saisi



(d) Appel

Figure 6: Mise en œuvre de l'Intent implicite pour passer un appel

### 3 Application simple pour consulter les horaires de trains

Pour cette application, il fallait permettre à l'utilisateur de rentrer le nom de deux villes, pour ensuite afficher les horaires de train disponibles pour relier ces deux villes. J'ai fait très simple, avec une seule vue et une seule activité. Les horaires sont codées en dur et sont les mêmes quelles que soient les villes saisies.

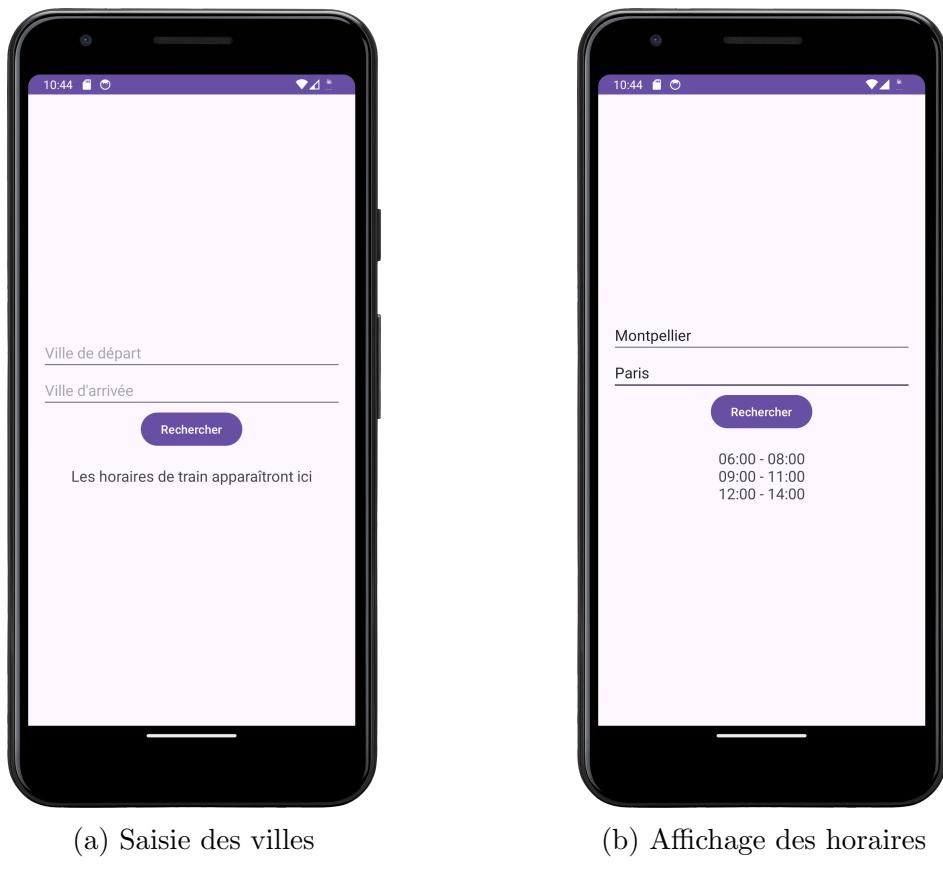


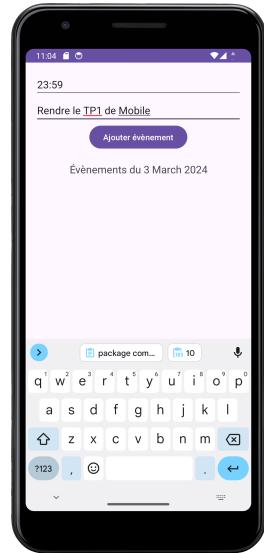
Figure 7: Simple application d'horaires de train

## 4 Application simple d'agenda

L'objectif final de ce TP était de réaliser une application simple d'agenda. L'application que j'ai réalisée récupère le jour actuel et permet d'y ajouter des évènements à des heures précises, puis de les visualiser.



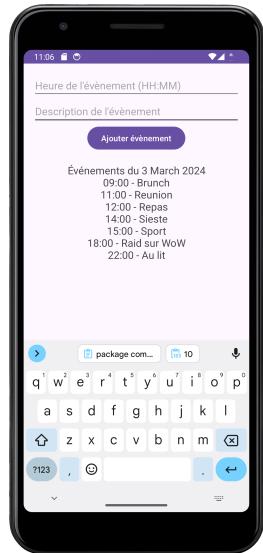
(a) Accueil de l'application



(b) Saisie d'un évènement



(c) Enregistrement d'un évènement



(d) Exemple avec une multitude d'évènements

Figure 8: Simple application d'agenda

## **5 Conclusion**

J'ai pris plaisir à me familiariser avec Android Studio et je suis désormais prêt à aborder le TP suivant pour perfectionner mes compétences et travailler sur des projets plus ambitieux et fonctionnels.

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.