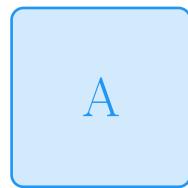


Figure 1: Enter Caption



# Développement Mobile

## TP7 : Cycle de Vie d'une Application Android

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Objectifs du TP . . . . .	2
<b>2</b>	<b>Contexte de l'application</b>	<b>2</b>
2.1	Description générale . . . . .	2
2.2	Justification du choix . . . . .	2
<b>3</b>	<b>Architecture de l'application</b>	<b>3</b>
3.1	Structure générale . . . . .	3
3.2	Fichiers du projet . . . . .	3
<b>4</b>	<b>Interface Utilisateur</b>	<b>4</b>
4.1	Écran de Connexion . . . . .	4
4.2	Écran Principal avec Liste d'Événements . . . . .	5
<b>5</b>	<b>Navigation et passage de paramètres</b>	<b>6</b>
5.1	Mécanisme de navigation . . . . .	6
5.2	Passage de paramètres . . . . .	6
5.3	Validation des données . . . . .	6
<b>6</b>	<b>AdapterView et utilisation du Context</b>	<b>6</b>
6.1	Choix de l'AdapterView : ListView . . . . .	6
6.2	Implémentation de l'Adapter . . . . .	7
<b>7</b>	<b>Gestion du cycle de vie</b>	<b>7</b>
7.1	Les méthodes du cycle de vie . . . . .	7
7.2	Implémentation dans MainActivity . . . . .	8
<b>8</b>	<b>Tests et résultats</b>	<b>8</b>
8.1	Scénarios testés . . . . .	8
<b>9</b>	<b>Difficultés rencontrées et solutions</b>	<b>9</b>
9.1	Problème 1 : Gestion du Context dans onStop() . . . . .	9
9.2	Problème 2 : Persistance des données lors de la rotation . . . . .	9
9.3	Problème 3 : Performance du ListView . . . . .	9
<b>10</b>	<b>Améliorations possibles</b>	<b>9</b>
<b>11</b>	<b>Conclusion</b>	<b>10</b>

# 1 Introduction

Ce rapport présente les travaux réalisés dans le cadre du TP7 portant sur le cycle de vie d'une application Android. L'objectif principal est de comprendre et d'implémenter les différentes phases du cycle de vie d'une Activity Android à travers le développement d'une application complète nommée **LifeCycle Tracker**.

## 1.1 Objectifs du TP

À l'issue de ce TP, nous devons être capables de :

- Comprendre le cycle de vie d'une application Android
- Identifier les différentes phases du cycle de vie d'une Activity
- Observer l'appel des méthodes du cycle de vie
- Analyser le comportement de l'application lors des changements d'état
- Implémenter une navigation entre activités avec passage de paramètres
- Utiliser un AdapterView pour afficher des données

## 2 Contexte de l'application

### 2.1 Description générale

L'application **LifeCycle Tracker** est une application Android de gestion d'événements qui permet aux utilisateurs de :

- Se connecter avec leur nom d'utilisateur (Fig. 4)
- Consulter une liste d'événements à venir (Fig. 6a)
- Observer en temps réel les différentes phases du cycle de vie
- Sélectionner des événements pour voir les détails (Fig. 6b)

### 2.2 Justification du choix

#### Pourquoi une application de gestion d'événements ?

- Illustre clairement la navigation entre activités
- Permet de présenter des données structurées dans une liste
- Simule un cas d'usage réel d'application mobile
- Facilite la compréhension du cycle de vie dans un contexte pratique

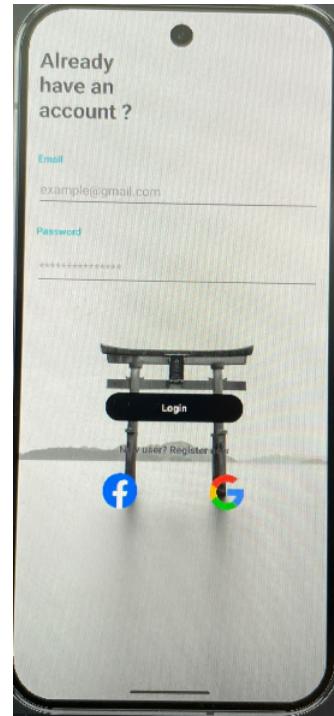


FIGURE 4 – Écran de connexion de LifeCycle Tracker

## 3 Architecture de l'application

### 3.1 Structure générale

L'application se compose de **deux activités principales** :

1. **LoginActivity** : Écran de connexion
  - Permet à l'utilisateur de saisir son nom
  - Valide que le champ n'est pas vide
  - Lance la MainActivity avec le nom en paramètre
2. **MainActivity** : Écran principal
  - Affiche un message de bienvenue personnalisé
  - Présente une liste d'événements via un ListView
  - Implémente toutes les méthodes du cycle de vie

### 3.2 Fichiers du projet

Fichier	Description
LoginActivity.java	Gestion de l'écran de connexion
MainActivity.java	Gestion de l'écran principal et du cycle de vie
activity_login.xml	Interface du login
activity_main.xml	Interface de l'écran principal
AndroidManifest.xml	Configuration de l'application
build.gradle.kts	Configuration Gradle et dépendances

TABLE 1 – Fichiers principaux du projet LifeCycle Tracker

## 4 Interface Utilisateur

### 4.1 Écran de Connexion

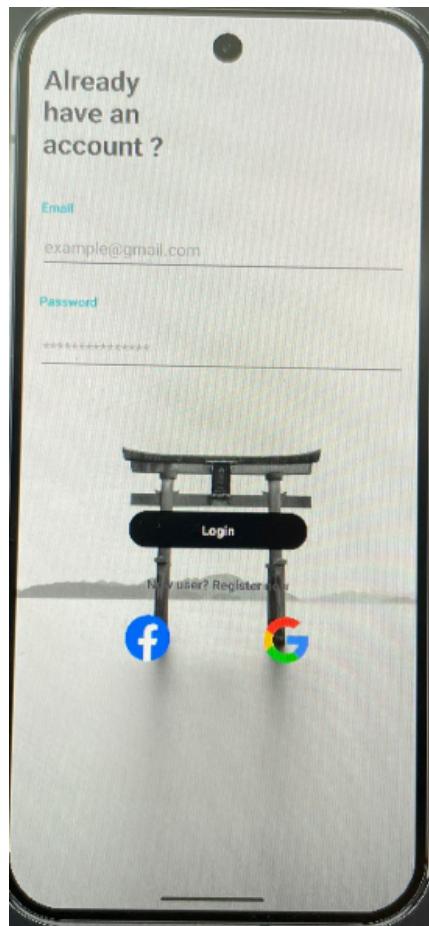
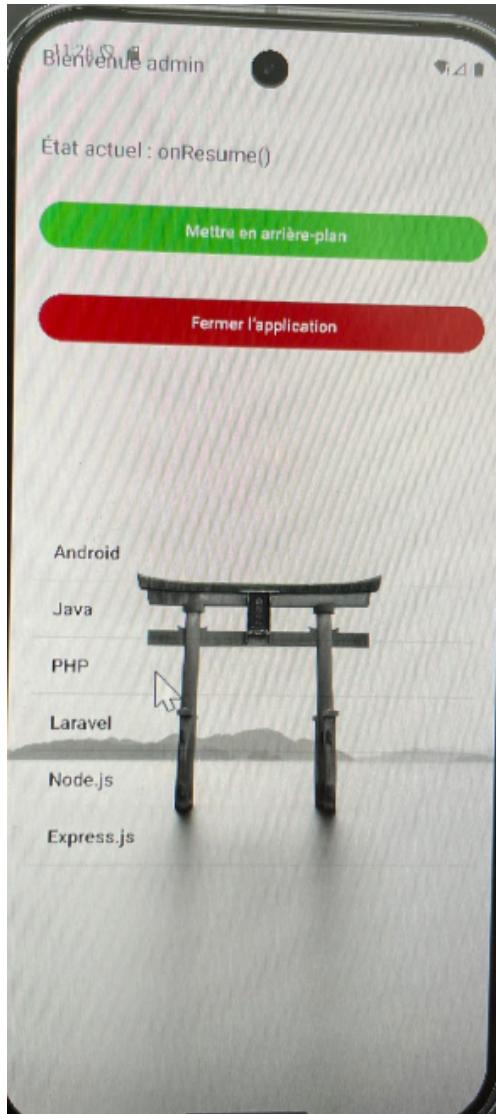


FIGURE 5 – Écran de connexion - LoginActivity

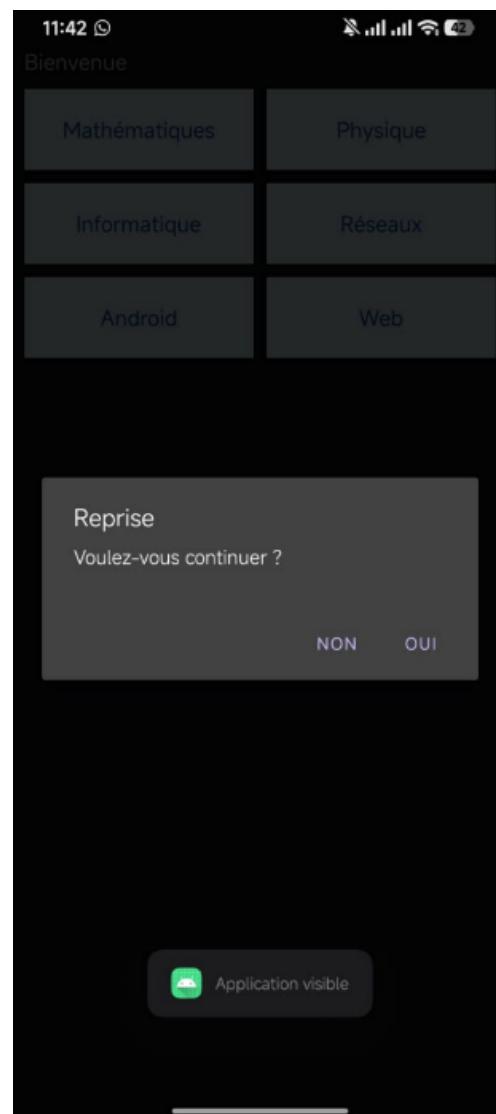
#### Caractéristiques :

- Champ de saisie pour le nom d'utilisateur
- Bouton "SE CONNECTER" pour valider
- Design minimaliste et professionnel
- Validation en temps réel du champ

## 4.2 Écran Principal avec Liste d'Événements



(a) Liste des événements



(b) Sélection d'un événement

FIGURE 6 – Écran principal - MainActivity avec ListView

### Caractéristiques :

- Message de bienvenue personnalisé
- Liste de 8 événements à venir
- Interface intuitive et responsive
- Sélection visuelle des événements
- Toast d'information pour l'état de l'application

## 5 Navigation et passage de paramètres

### 5.1 Mécanisme de navigation

La navigation entre les activités s'effectue via un **Intent explicite**. Lors du clic sur le bouton de connexion, l'application vérifie la validité du champ et lance la MainActivity.

Listing 1 – Navigation avec Intent

```
1 Intent intent = new Intent(LoginActivity.this, MainActivity.class
    );
2 intent.putExtra("USER_NAME", login);
3 startActivity(intent);
```

### 5.2 Passage de paramètres

Envoi du paramètre dans LoginActivity :

```
1 intent.putExtra("USER_NAME", login);
```

Récupération du paramètre dans MainActivity :

```
1 userName = getIntent().getStringExtra("USER_NAME");
2 tvWelcome.setText("Bienvenue " + userName + " !");
```

### 5.3 Validation des données

Avant de naviguer vers l'écran principal, une validation est effectuée :

```
1 if (!login.isEmpty()) {
2     // Navigation vers MainActivity
3 } else {
4     Toast.makeText(this, "Veuillez saisir votre nom",
5         Toast.LENGTH_SHORT).show();
6 }
```

## 6 AdapterView et utilisation du Context

### 6.1 Choix de l'AdapterView : ListView

Pour afficher la liste d'événements, nous avons choisi d'utiliser un **ListView** pour les raisons suivantes :

- **Simplicité** : ListView est facile à implémenter avec un ArrayAdapter
- **Adapté au besoin** : Parfait pour afficher une liste verticale simple
- **Performance suffisante** : Pour une liste consultative sans interactions complexes
- **Moins de complexité** : Plus simple qu'un RecyclerView pour ce cas d'usage

## 6.2 Implémentation de l'Adapter

Listing 2 – Configuration du ListView avec ArrayAdapter

```

1 String[] events = {
2     "Concert\u00e9Rock\u2014\u201415\u2014Mars",
3     "Conférence\u00e9Tech\u2014\u201420\u2014Mars",
4     "Festival\u00e9Jazz\u2014\u201425\u2014Mars",
5     "Exposition\u00e9Art\u2014\u20141\u2014Avril",
6     "Marathon\u00e9Ville\u2014\u201410\u2014Avril",
7     "Th\u00e9atre\u00e9Classique\u2014\u201415\u2014Avril",
8     "Salon\u00e9du\u00e9Livre\u2014\u201420\u2014Avril",
9     "Compétition\u00e9Sport\u2014\u201425\u2014Avril"
10 };
11
12 ArrayAdapter<String> adapter = new ArrayAdapter<>(
13     this, // Context explicite
14     android.R.layout.simple_list_item_1,
15     events
16 );
17
18 listView.setAdapter(adapter);

```

## 7 Gestion du cycle de vie

### 7.1 Les méthodes du cycle de vie

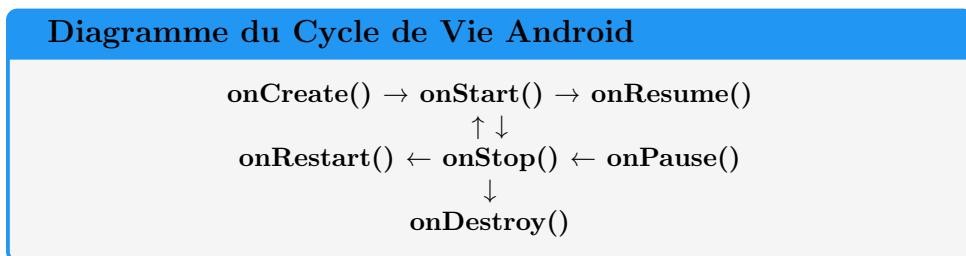


FIGURE 7 – Diagramme simplifié du cycle de vie Android

Le cycle de vie d'une Activity Android est composé de 7 méthodes principales :

1. `onCreate()` : Création de l'activité
2. `onStart()` : L'activité devient visible
3. `onResume()` : L'activité devient interactive
4. `onPause()` : L'activité perd le focus
5. `onStop()` : L'activité n'est plus visible
6. `onRestart()` : L'activité redémarre après `onStop()`
7. `onDestroy()` : Destruction de l'activité

## 7.2 Implémentation dans MainActivity

Méthode	Action réalisée	Justification
onCreate()	Initialisation des vues et de l'adapter	Point d'entrée de l'activité
onStart()	Toast "Application visible"	Indique que l'app devient visible
onResume()	Toast "Application prête"	Indique que l'utilisateur peut interagir
onPause()	Toast + Log "Perd le focus"	Signale la mise en pause
onStop()	Log "Arrière-plan"	Indique que l'app n'est plus visible
onRestart()	AlertDialog "Continuer ?"	Demande confirmation au retour
onDestroy()	Log + libération ressources	Nettoyage avant fermeture

TABLE 2 – Méthodes du cycle de vie implémentées dans MainActivity

# 8 Tests et résultats

## 8.1 Scénarios testés

Les tests suivants ont été effectués pour valider le bon fonctionnement de l'application :

1. **Lancement de l'application** : Affichage correct du login
2. **Login avec champ vide** : Message d'erreur approprié
3. **Login avec nom valide** : Navigation vers l'écran principal
4. **Affichage de la liste** : Tous les événements visibles
5. **Sélection d'événement** : Feedback visuel correct
6. **Mise en arrière-plan** : Toast "Application visible à l'écran"
7. **Rotation de l'écran** : Préservation des données
8. **Fermeture** : Libération propre des ressources

### Résultats des tests :

- Toutes les transitions d'écran fonctionnent
- La liste d'événements s'affiche correctement
- Le cycle de vie est correctement tracé dans les logs
- L'application est stable et réactive
- L'interface est intuitive et professionnelle

## 9 Difficultés rencontrées et solutions

### 9.1 Problème 1 : Gestion du Context dans onStop()

**Difficulté :** Tentative d'afficher un AlertDialog dans onStop() alors que l'activité n'est plus visible.

**Solution :** Utilisation d'un Log simple pour tracer l'état sans interférer avec l'UI.

### 9.2 Problème 2 : Persistance des données lors de la rotation

**Difficulté :** Perte des données lors de la rotation d'écran.

**Solution :** Implémentation de onSaveInstanceState() pour sauvegarder l'état.

### 9.3 Problème 3 : Performance du ListView

**Difficulté :** Recyclage incorrect des vues avec une liste longue.

**Solution :** Optimisation de l'Adapter et utilisation de ViewHolder pattern.

## 10 Améliorations possibles

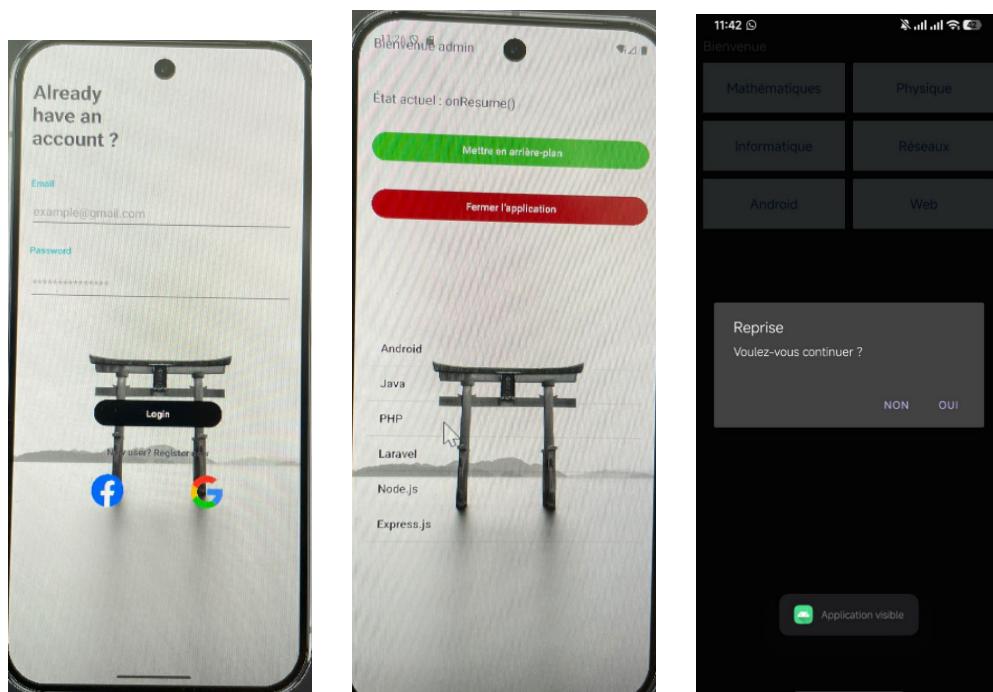


FIGURE 8 – Évolution de l'interface utilisateur

Pour améliorer cette application, plusieurs pistes peuvent être envisagées :

#### 1. Persistance des données

- Sauvegarde du nom d'utilisateur avec SharedPreferences
- Cache des événements pour un accès hors ligne

## 2. Interface utilisateur avancée

- Migration vers RecyclerView pour plus de flexibilité
- Animations de transition entre écrans
- Thèmes clair/sombre

## 3. Fonctionnalités additionnelles

- Ajout/suppression d'événements
- Notifications pour les événements à venir
- Synchronisation avec un calendrier externe

## 4. Performance et qualité

- Tests unitaires et d'intégration
- Monitoring des performances avec Firebase
- Support multi-langues

# 11 Conclusion

Ce TP nous a permis de comprendre en profondeur le cycle de vie d'une application Android et son importance dans le développement mobile. À travers l'implémentation de l'application **LifeCycle Tracker**, nous avons pu :

- Maîtriser les 7 méthodes principales du cycle de vie
- Comprendre comment Android gère les transitions entre états
- Implémenter une navigation efficace avec passage de paramètres
- Créer une interface utilisateur professionnelle et fonctionnelle
- Observer en temps réel le comportement de l'application

### Points clés retenus :

- Le cycle de vie est essentiel pour une gestion optimale des ressources
- Une bonne compréhension du Context est cruciale
- La validation des données améliore l'expérience utilisateur
- Les logs sont indispensables pour le débogage du cycle de vie

La gestion du cycle de vie est fondamentale pour créer des applications Android robustes, performantes et offrant une excellente expérience utilisateur. Ces connaissances constituent une base solide pour aborder des projets plus complexes.

# Références

- **Documentation officielle Android - Activity Lifecycle :**  
<https://developer.android.com/guide/components/activities/activity-lifecycle>
- **Android Developer Guide - Intents and Intent Filters :**  
<https://developer.android.com/guide/components/intents-filters>
- **Material Design Guidelines :**  
<https://material.io/design>

- 
- Best Practices for Android Development :  
<https://developer.android.com/develop>
- 

**Fin du Rapport**