Cours - Informatique

Création d'applications mobiles sous Android



© HOFMANS Pierre
Août 2026

Table des Matières

PARTIE 1 : DECOUVRIR KODULAR

1. Les bases du développement d'applications avec Kodular 1
   A. Objectifs de ce chapitre 1
   B. Bienvenue dans le monde des applications 1
   C. Qu'est-ce qu'une application Android ? 1
   D. Pourquoi Kodular ? 2
   E. L'histoire de Kodular 2
   F. Le développement visuel 3
   G. Kodular est-il réservé aux débutants ? 4
   H. Comment fonctionne une application ? 4
   I. Les avantages de Kodular 5
   J. Les limites de Kodular 5

PARTIE 2 : PREMIERS PAS AVEC KODULAR

1. Création du compte et découverte de l'interface 6
   A. Objectifs de ce chapitre 6
   B. Avant de commencer 6
   C. Le matériel nécessaire 6
   D. Créer un compte Kodular 6
   E. Créer ton premier projet 8
   F. Découverte de l'interface 9
   G. Le Viewer 9
   H. La Palette 9
   I. Les composants (Components) 10
   J. Les propriétés (Properties) 10
   K. Ajouter un composant 10
   L. Sauvegarde d'un projet 11
   M. Mini-projet : Ma première interface 11
2. Maîtriser le Designer et les composants visibles 12
   A. Objectifs de ce chapitre 12
   B. Qu'est-ce que le Designer ? 12
   C. Les composants visibles 13
   D. Les composants de mise en page (Layouts) 14
   E. Les propriétés importantes 17
   F. Nommer correctement ses composants 17
   G. Exercice guidé : Créer un écran d'accueil 18
   H. Comment tester son application ? 18
   I. Exercice 18
3. Les premiers Blocks : Donner vie à une application 18
   A. Objectifs de ce chapitre 18
   B. Du Designer aux Blocks 19
   C. Qu'est-ce qu'un block ? 19
   D. Les différentes catégories de blocks 19
   E. Comprendre les événements 20

F. Notre premier programme 20
G. Bonnes pratiques 21
H. Exercice : Application de bienvenue 21
I. Atelier pratique : Ton premier défi 22
J. Les principales erreurs des débutants 22
K. Bonnes pratiques des développeurs 23
L. Lexique du chapitre 24

PARTIE 3 : MAITRISER LE DESIGNER

1. Construire une interface professionnelle 25
A. Objectifs de ce chapitre 25
B. Introduction 25
C. le plan de ton application 25
D. Les éléments du Designer 25
E. Screen1 26
F. Les différentes tailles d'écran 26
G. Le système de coordonnées 27
H. Atelier pratique 27
I. Astuce de développeur 27
J. Les erreurs fréquentes 28
2. Les composants de mise en page (Layouts) 28
A. Objectifs de ce chapitre 28
B. Introduction 28
C. Les différents Arrangements 28
D. Le Vertical Arrangement 29
E. Le Horizontal Arrangement 29
F. Le Table Arrangement 29
G. Le Vertical Scroll Arrangement 30
H. Les propriétés importantes des Arrangements 31
I. Imbriquer les Arrangements 31
J. Les marges et les espacements 31
K. Concevoir une interface responsive 32
L. Atelier pratique 32
M. Conseils 33
N. Exercice 33
3. Les composants visibles de Kodular 34
A. Objectifs de ce chapitre 34
B. Introduction 34
C. Le Label 35
D. Le Button 35
E. La Text Box 35
F. L'Image 36
G. La Check Box 36
H. Le Switch 36
I. Le Slider 36
J. Le Spinner 37
K. Le List View 37
L. Le Date Picker 37
M. Le Time Picker 37
N. Le Notifier 38
O. Choisir le bon composant 38
P. Atelier pratique 38

Q. Exercice 39
4. Les composants invisibles : les véritables moteurs de l'application 40
A. Objectifs de ce chapitre 40
B. Introduction 40
C. Le composant Notifier 41
D. Le composant Clock 41
E. Le composant TinyDB 41
F. Le composant Sound 42
G. Le composant Player 42
H. Le composant Camera 43
I. Le composant Image Picker 43
J. Le composant Sharing 43
K. Le composant Activity Starter 44
L. Le composant Barcode Scanner 44
M. Le composant Location Sensor 45
N. Le composant Speech Recognizer 45
O. Le composant Text To Speech 46
P. Exercice : Mon assistant personnel 46
5. Projet complet, bonnes pratiques 48
A. Objectifs de ce chapitre 48
B. Introduction 49
C. Avant de programmer 49
D. Définition de l'interface 50
E. Penser comme un utilisateur 51
F. Vérifier l'ergonomie 51
G. Exercice : Projet *Ma bibliothèque* 51

PARTIE 4 : PROGRAMMER AVEC LES BLOCS

1. Comprendre la programmation événementielle 54
A. Objectifs de ce chapitre 54
B. Introduction 54
C. Qu'est-ce que la programmation événementielle ? 54
D. Le cycle « Événement → Traitement → Résultat » 54
E. Les principaux événements 55
F. Une action peut déclencher plusieurs actions 55
G. Les événements automatiques 56
H. Les erreurs fréquentes 56
I. Exercice 57
J. Exercice 57
K. Les événements les plus utilisés 57
2. Les variables : mémoriser et manipuler des données 57
A. Objectifs de ce chapitre 57
B. Qu'est-ce qu'une variable ? 57
C. Les différents types de données 58
D. Les variables globales 58
E. Les variables locales 59
F. Variables et composants 59
G. Les erreurs fréquentes 59
H. Exercice : Le compteur 59
I. Exercice : Calcul du total 59
3. Les conditions : apprendre à prendre des décisions 60
A. Objectifs de ce chapitre 60

B. Introduction 60
C. Qu'est-ce qu'une condition ? 60
D. Le bloc IF 61
E. Exemple concret 61
F. Les opérateurs de comparaison 61
G. Le bloc IF...ELSE 62
H. Les conditions multiples 63
I. Exercice : Calcul d'une remise 64
J. Exercice : Contrôle d'accès 64
4. Les boucles : répéter des actions intelligemment 65
A. Objectifs de ce chapitre 65
B. Introduction 65
C. La boucle FOR EACH number FROM … TO … 66
D. La boucle FOR EACH item IN LIST 67
E. La boucle WHILE … DO 67
F. Exemple concret : table de multiplication 68
G. Les boucles imbriquées 68
H. Exercice : Analyse des notes d'une classe 69
5. Les procédures : organiser son code comme un développeur professionnel .. 70
A. Objectifs de ce chapitre 70
B. Introduction 70
C. Pourquoi créer une procédure ? 71
D. Créer et appeler une procédure 71
E. Les procédures qui renvoient un résultat 72
F. Exercice : Vider un formulaire 73

PARTIE 5 : LES LISTES ET LES DICTIONNAIRES

1. Comprendre les listes : stocker plusieurs informations dans une variable 75
A. Objectifs de ce chapitre 75
B. Introduction 75
C. Qu'est-ce qu'une liste ? 75
D. Les éléments d'une liste 75
E. Les types de données dans une liste 76
F. Créer une liste dans Kodular 76
G. Combien d'éléments contient une liste ? 76
H. Les listes dans la vie quotidienne 77
I. Exercice : Mes films préférés 77
2. Manipuler les listes : ajouter, modifier et supprimer des éléments 78
A. Objectifs de ce chapitre 78
B. Introduction 78
C. Ajouter un élément à une liste 79
D. Insérer un élément à une position précise 79
E. Modifier un élément 80
F. Supprimer un élément 80
G. Vider complètement une liste 80
H. Vérifier si une liste est vide 80
I. Copier une liste 80
J. Exercice : Gestionnaire de tâches 81
3. Rechercher, trier et parcourir les listes 82
A. Objectifs de ce chapitre 82
B. Introduction 82
C. Rechercher un élément 83

D. Trouver la position d'un élément 83
E. Parcourir une liste 83
F. Trier une liste 84
G. Filtrer une liste 85
H. Les performances 85
I. Astuce de développeur 86
J. Les erreurs fréquentes 86
K. Exercice : Classement des joueurs 86
4. Les dictionnaires : associer des informations intelligemment 87
A. Objectifs de ce chapitre 87
B. Introduction 88
C. Qu'est-ce qu'un dictionnaire ? 88
D. Pourquoi utiliser un dictionnaire ? 88
E. Créer un dictionnaire dans Kodular 88
F. Rechercher une valeur 89
G. Ajouter une nouvelle entrée 89
H. Modifier une valeur 90
I. Supprimer une entrée 90
J. Parcourir un dictionnaire 90
K. Exercice : Répertoire téléphonique 90
L. Astuces de développeur 92
M. Exercice : Annuaire intelligent 92
N. Remplacer tous les boutons par un menu dans le bas de l'écran 95
O. Ajouter un tri alphabétique 98
P. Exercice de dépassement : Ajouter une recherche instantanée 99
Q. Sauvegarder les contacts 100

PARTIE 6 : LE STOCKAGE DES DONNEES

1. Conserver les données avec TinyDB 101
A. Objectifs de ce chapitre 101
B. Introduction 101
C. Qu'est-ce que TinyDB ? 101
D. Où se trouve TinyDB ? 101
E. Comment fonctionne TinyDB ? 102
F. Que se passe-t-il si la clé n'existe pas ? 102
G. Modifier une donnée 102
H. Supprimer une donnée 102
I. Quels types de données peut-on enregistrer ? 102
J. Que peut-on sauvegarder ? 103
K. Exercice : Mémoriser un prénom et un score 103
2. Sauvegarder des listes et des dictionnaires avec TinyDB 105
A. Objectifs de ce chapitre 105
B. Introduction 105
C. Sauvegarder une liste 105
D. Restaurer une liste 106
E. Modifier une liste sauvegardée 106
F. Sauvegarder un dictionnaire 107
G. Charger les données au démarrage 107
H. Vérifier si une sauvegarde existe 107
I. Exercice : Liste de courses permanente 107
J. Exercice : Carnet de contacts 108
K. Les erreurs fréquentes 109

L. Bonnes pratiques 109
3. Les fichiers : enregistrer et lire des données 110
A. Objectifs de ce chapitre 110
B. Introduction 110
C. Qu'est-ce qu'un fichier ? 110
D. TinyDB ou fichier ? 111
E. Le composant File 111
F. créer, écrire et lire un fichier texte 111
G. Modifier un fichier texte 113
H. Supprimer un fichier texte 113
I. Les fichiers CSV 113
J. Partager un fichier 117
K. Exercice : Bloc-notes 117
L. Exercice : Bloc-notes CSV 119
M. Bonnes pratiques 122
N. Résumé 122
O. Ce que nous verrons dans la Partie 4 122
4. Découvrir SQLite : la base de données intégrée à Android 122
A. Objectifs de ce chapitre 122
B. Introduction 122
C. Qu'est-ce que SQLite ? 123
D. Qu'est-ce qu'une base de données ? 123
E. Notion de table 123
F. Notion d'enregistrement et de champ 124
G. Notion de liens entre les tables 125
H. Notion de clé primaire 125
I. Notion d'indexation des données 126
J. Les opérations principales 126
K. Méthodologie à suivre pour la conception d'une base de données 126
L. Analyse des souhaits, besoins et situation actuelle du client 126
M. Identifier les entités 127
N. Les attributs des entités 127
O. Les identifiants des entités 128
P. Les liens entre les entités, le schéma relationnel 129
Q. Les optimisations possibles 129
R. TinyDB, fichiers ou SQLite ? 130
S. Les commandes SQL 131
T. Exercice : Conception d'une base de données pour une bibliotheque 137
5. Supabase : La base de données en ligne 143
A. Objectifs de ce chapitre 143
B. Introduction 143
C. Comprendre le principe 144
D. Première étape : créer ton projet Supabase 144
E. Créer notre première table : livres 145
F. Préparer les autorisations 145
G. Construire l'écran dans Kodular 146
H. Relier Kodular à Supabase 146
I. Comprendre les blocs avec trois opérations 146
J. Ce qu'il faut retenir 149
K. Que faire si la liste reste vide ? 150
L. Sécuriser les données 150
M. Bonnes pratiques 150

N. Exercice : Créer une liste de tâches collaborative..... 150

Découvrir Kodular

1. Les bases du developpement d'applications avec Kodular

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre ce qu'est une application Android ;
- Découvrir ce qu'est Kodular et pourquoi il a été créé ;
- Comprendre le principe du développement visuel ;
- Connaître les avantages et les limites de Kodular ;
- Découvrir le fonctionnement général d'une application Android.

B. BIENVENUE DANS LE MONDE DES APPLICATIONS

Aujourd'hui, les applications mobiles font partie de notre quotidien. Que ce soit pour consulter la météo, envoyer des messages, écouter de la musique, réserver un voyage ou gérer son compte bancaire, nous utilisons des dizaines d'applications sans même y penser.

Mais derrière chacune de ces applications se cache un travail de conception et de programmation.

Pendant longtemps, créer une application était réservé aux développeurs maîtrisant des langages comme **Java**, **Kotlin** ou **C++**. Il fallait apprendre une syntaxe complexe, connaître les bibliothèques Android et passer des semaines à résoudre des erreurs de programmation.

Heureusement, les outils ont évolué.

Aujourd'hui, il est possible de créer des applications professionnelles grâce à des plateformes de développement visuel, dont **Kodular** est l'une des plus populaires.

C. QU'EST-CE QU'UNE APPLICATION ANDROID ?

Une application Android est un programme conçu pour fonctionner sur un smartphone, une tablette ou tout autre appareil utilisant le système d'exploitation Android.

Une application peut être très simple, comme une calculatrice, ou extrêmement complexe, comme un réseau social ou une plateforme de commerce en ligne.

Dans tous les cas, une application repose sur trois éléments essentiels :

- **Une interface utilisateur** : ce que l'utilisateur voit à l'écran (boutons, images, textes, listes, ...).
- **Une logique** : ce qui se passe lorsque l'utilisateur interagit avec l'application.

- **Des données** : les informations que l'application affiche, modifie ou enregistre.

À retenir
Une application est avant tout une combinaison entre une interface, des règles de fonctionnement et des données.

D. POURQUOI KODULAR ?

Imagine que tu souhaites construire une maison.

Tu pourrais fabriquer toi-même les briques, les fenêtres, les portes, les tuiles et les poutres. Ce serait très long et demanderait des compétences très spécialisées.

Une autre solution consiste à utiliser des éléments déjà fabriqués et à les assembler intelligemment.

Kodular applique exactement ce principe au développement d'applications.

Au lieu d'écrire des centaines de lignes de code, tu construis ton application en assemblant des composants graphiques.

Cette approche est appelée **développement visuel**.

Une comparaison simple

Prenons l'exemple d'un bouton. Dans un langage classique, créer un bouton demande de nombreuses lignes de code :

- Création de l'objet ;
- Choix de la couleur ;
- Définition de la taille ;
- Positionnement ;
- Gestion du clic ;
- Modification de son apparence.

Avec Kodular :

1. Tu fais glisser un composant **Bouton** sur l'écran.
2. Tu modifies son texte.
3. Tu lui indiques ce qu'il doit faire lorsqu'on clique dessus.

En quelques secondes, ton bouton est opérationnel.

C'est ce qui rend Kodular particulièrement accessible aux débutants.

E. L'HISTOIRE DE KODULAR

Kodular est basé sur **MIT App Inventor**, un projet lancé par le Massachusetts Institute of Technology (MIT) afin de démocratiser la création d'applications Android.

L'objectif était simple :
Permettre à toute personne, même sans expérience en programmation, de créer sa propre application. Au fil du temps, des développeurs indépendants ont enrichi cette base pour proposer davantage de composants, de possibilités de personnalisation et de fonctionnalités avancées. C'est ainsi qu'est né Kodular.

Aujourd'hui, Kodular permet de réaliser des applications modernes intégrant notamment :

- Des cartes interactives ;
- Le GPS ;
- Le Bluetooth ;
- Des lecteurs audio et vidéo ;
- Des bases de données ;
- Des services Web (API) ;
- Des notifications ;
- Des publicités ;
- Des animations.

F. LE DEVELOPPEMENT VISUEL

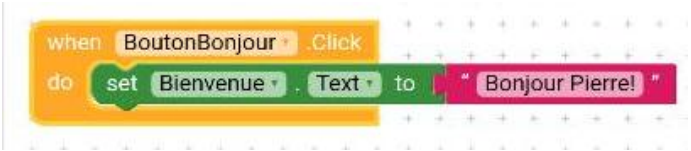
Le développement visuel repose sur une idée simple. Au lieu d'écrire du code, on assemble des blocs représentant des instructions.

Cette méthode présente plusieurs avantages:

- **Une prise en main rapide**
  Tu peux créer une première application en quelques minutes seulement.
- **Moins d'erreurs**
  Les blocs ne peuvent être assemblés que s'ils sont compatibles, ce qui réduit fortement les erreurs de syntaxe.
- **Une logique plus facile à comprendre**
  Les blocs représentent des actions concrètes :
  - Afficher un message ;
  - Ouvrir un écran ;
  - Jouer un son ;
  - Enregistrer une donnée.

Cette représentation est souvent plus intuitive qu'un langage de programmation textuel.

Exemple
Imaginons que tu souhaites afficher le message "Bonjour !" lorsqu'un utilisateur clique sur un bouton. En programmation traditionnelle, plusieurs lignes de code sont nécessaires. Avec Kodular, tu relies simplement des blocs :



Lorsque BoutonBonjour est cliqué, tu changes le texte de la zone "Bienvenue" en "Bonjour Pierre!".

Même sans connaître un langage de programmation, tu peux comprendre ce qui se passe.

G. KODULAR EST-IL RESERVE AUX DEBUTANTS ?

C'est une question fréquente. La réponse est **non**. Kodular est souvent utilisé pour apprendre la programmation, mais il est également capable de produire des applications complètes et performantes.

De nombreuses applications créées avec Kodular sont disponibles sur le Google Play Store.

Grâce aux extensions, il est possible d'ajouter des fonctionnalités avancées :

- Reconnaissance vocale ;
- Lecture de QR Codes ;
- Connexion à Supabase ;
- Intelligence artificielle ;
- Communication Bluetooth ;
- Capteurs du téléphone.

À retenir
Kodular n'est pas un jouet. C'est un environnement de développement sérieux, adapté aussi bien aux débutants qu'à des projets ambitieux.

H. COMMENT FONCTIONNE UNE APPLICATION ?

Toutes les applications, quel que soit le langage utilisé, suivent le même principe :

1. L'utilisateur effectue une action.
2. L'application détecte cette action.
3. L'application exécute une ou plusieurs instructions.
4. L'écran est mis à jour.

Prenons un exemple très simple :

Tu appuies sur un bouton intitulé **Calculer**.

L'application :

- Récupère les valeurs saisies ;
- Effectue un calcul ;
- Affiche le résultat.

Ce mécanisme est appelé **programmation événementielle**.

Un **événement** est une action qui déclenche un comportement.

Les événements les plus courants sont :

- Un clic sur un bouton ;
- L'ouverture d'un écran ;
- La saisie d'un texte ;
- L'arrivée d'un message ;
- L'obtention de la position GPS.

I. LES AVANTAGES DE KODULAR

Voici les principaux atouts de Kodular :

- **Rapidité de développement**
  Quelques heures suffisent pour créer une première application fonctionnelle.
- **Aucune connaissance en programmation n'est nécessaire**
  Les blocs remplacent les lignes de code.
- **Interface intuitive**
  Le glisser-déposer facilite énormément la création des écrans.
- **Evolutif**
  Il est possible de réaliser des projets très avancés grâce aux composants supplémentaires et aux extensions.
- **Gratuit**
  La plateforme est accessible (en partie) gratuitement, ce qui permet de se former sans investissement important.

J. LES LIMITES DE KODULAR

Comme tout outil, Kodular possède également quelques limites.

- Certaines fonctionnalités très spécifiques nécessitent des extensions externes.
- Les performances peuvent être légèrement inférieures à celles d'une application développée entièrement en Java ou Kotlin pour des usages très exigeants.
- Certaines mises à jour d'Android peuvent demander un temps d'adaptation avant d'être prises en charge.

Ces limites restent cependant mineures pour la grande majorité des projets.

Premiers pas avec Kodular

1. CREATION DU COMPTE ET DECOUVERTE DE L'INTERFACE

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Créer un compte sur Kodular ;
- Comprendre comment fonctionne un projet ;
- Découvrir les différentes zones de travail ;
- Créer ton premier projet ;
- Personnaliser l'écran principal de ton application.

B. AVANT DE COMMENCER

Avant de créer ta première application, tu dois disposer de quelques outils. La bonne nouvelle est que tu n'as rien à installer sur ta tablette, Kodular fonctionne directement dans un navigateur Internet.

Les navigateurs recommandés sont :

- Google Chrome
- Microsoft Edge
- Mozilla Firefox

C. LE MATERIEL NECESSAIRE

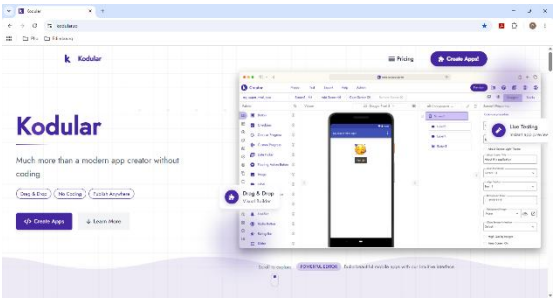
Pour suivre ce cours, il te faut :

- Une tablette sous Android ;
- Une connexion Internet ;
- Un compte Google ;
- Un smartphone Android afin de te permettre de tester ton application en temps réel grâce à **Kodular Companion**.

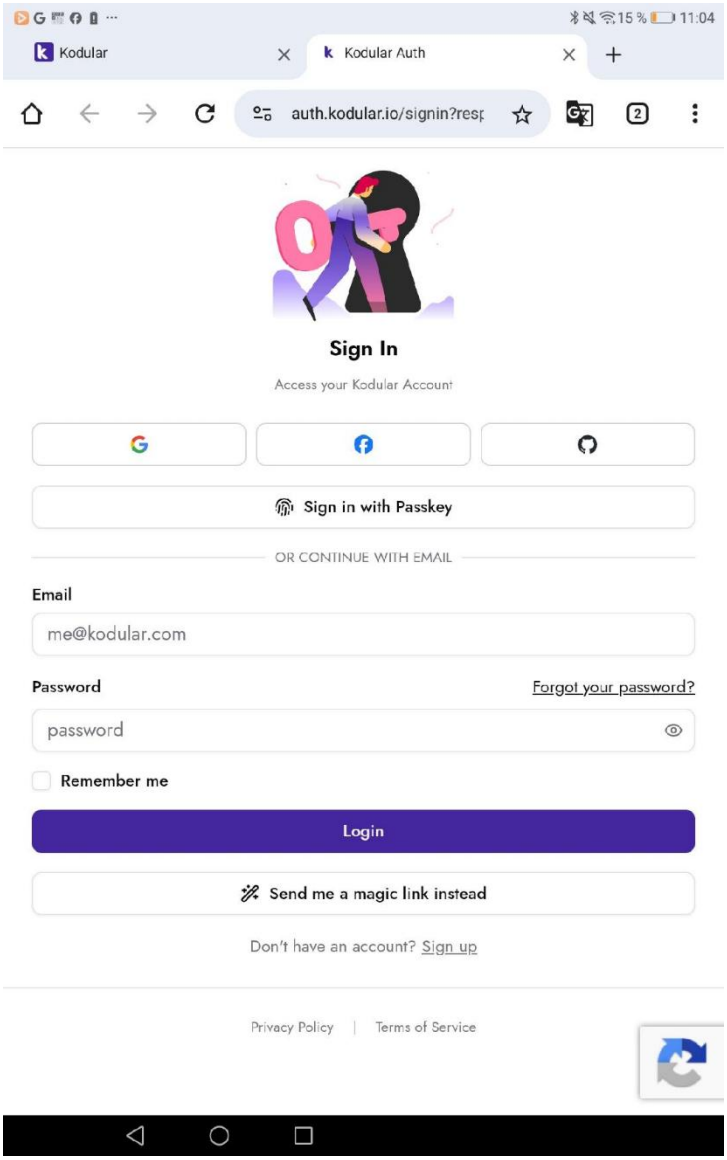
D. CREER UN COMPTE KODULAR

- Rends-toi sur le site officiel de Kodular **https://www.kodular.io/**.
  Tu arrives sur la page d'accueil.

- Au centre de l'écran se trouve un bouton **</> Create Apps** permettant de te connecter. Clique dessus.



- Kodular te demande ensuite de choisir un compte Google.

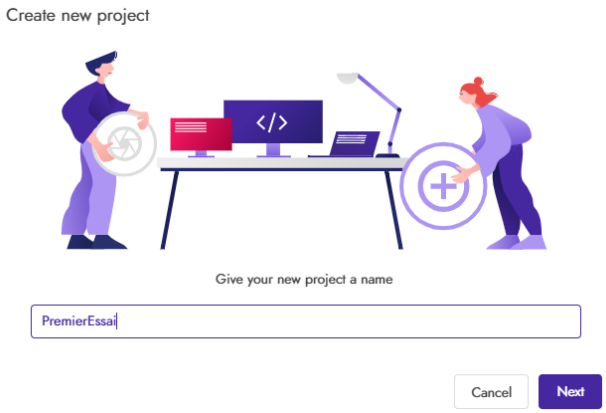


- Sélectionne ton compte et accepte les autorisations demandées. Quelques secondes plus tard, tu arrives sur ton espace de travail.

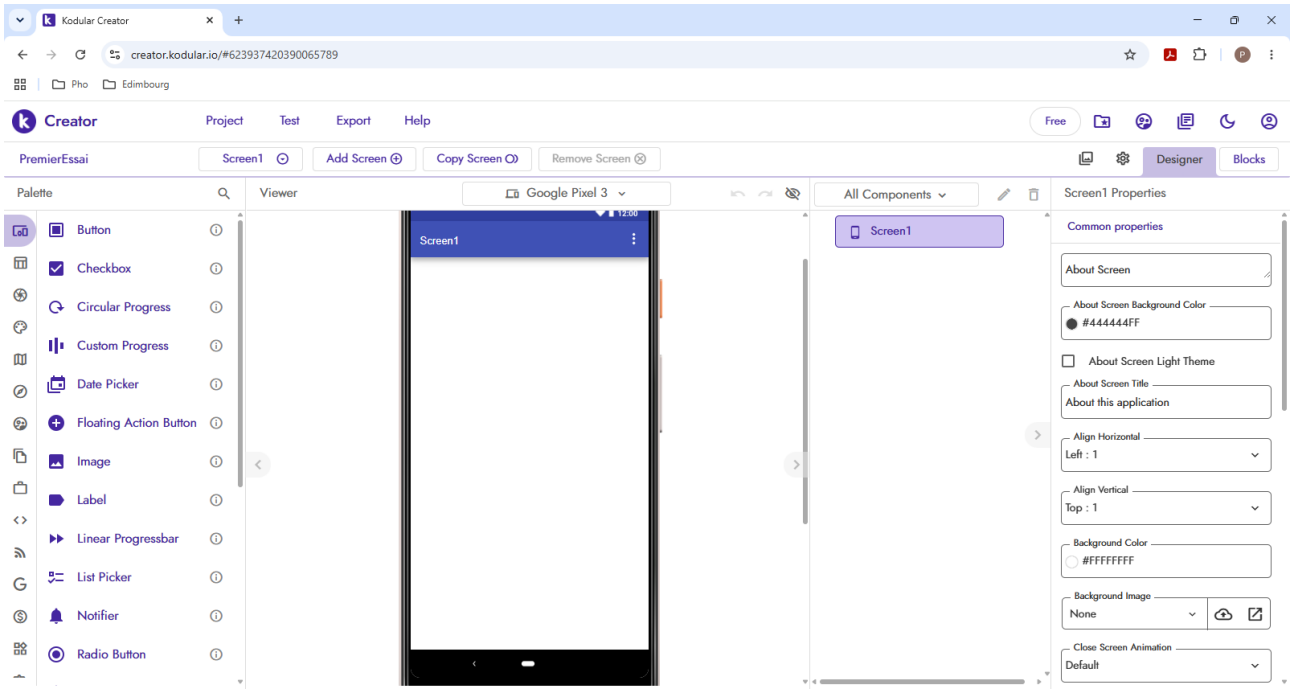
- Félicitations ! Tu viens de créer ton environnement de développement. Le tableau de bord affiche la liste de tous tes projets. Lors de ta première connexion, cette liste est vide. Plus tard, elle pourra contenir plusieurs dizaines d'applications.

E. CREER TON PREMIER PROJET

- Clique sur le bouton **Create Project**
- Encode le nom de ton projet dans la fenêtre qui apparaît.
  Attention, le nom ne peut contenir que des lettres non accentuées, des chiffres et le caractère Souligné (Underscore). De plus, il doit commencer par une lettre.
  → Pour cet exemple, on donnera le nom PremierEssai



- Clique sur le bouton **Next**
- Kodular te permet ensuite de définir certains paramètres de configuration de ton projet, mais tu pourras le faire plus tard également.
- Clique sur le bouton **Finish**
  → Kodular affiche à présent l'écran de gestion de ton projet



À retenir

Le nom du projet est différent du nom affiché sur le smartphone. Tu pourras modifier ce dernier plus tard.

F. DECOUVERTE DE L'INTERFACE

La première impression peut être déroutante. L'écran semble rempli de panneaux. Pourtant, l'interface est très logique.

Elle est composée de plusieurs grandes zones :

- Le **Viewer**, au centre de l'écran
- A gauche se trouve la **Palette**
- A la droite du Viewer se trouve la liste des composants (**All Components**)
- A droite de l'écran se trouve le panneau des propriétés (**Properties**)

G. LE VIEWER

Au centre de l'écran se trouve le **Viewer**. C'est la représentation d'un smartphone. Tout ce que tu ajoutes apparaît immédiatement.

Imagine que tu construises une maison. Le Viewer correspond au terrain. C'est ici que tu vas placer les différents éléments.

H. LA PALETTE

À gauche se trouve la Palette. C'est une immense boîte à outils. Elle contient tous les composants disponibles. Ils sont classés par catégories.

Par exemple :

- User Interface
- Layout
- Media
- Drawing and Animation
- Maps
- Sensors
- Social
- Storage
- Utilities
- Dynamic Components
- Connectivity
- Google
- Extension

Chaque catégorie contient plusieurs composants.

Les ***composants visibles*** qui apparaîtront à l'écran dont notamment :

- Button
- Label
- Image
- List View
- Text Box

Les ***composants invisibles*** qui travaillent en arrière-plan dont notamment :

- Camera
- Player
- Sound

- TinyDB
- Clock
- Web
- Bluetooth
- Location Sensor

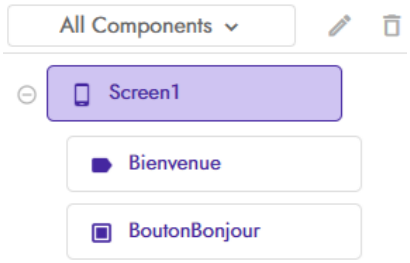
Ils sont essentiels mais n'apparaissent jamais sur l'écran du téléphone. Beaucoup de débutants pensent qu'un composant invisible ne sert à rien. C'est exactement le contraire. Les applications professionnelles utilisent énormément de composants invisibles.

I. LES COMPOSANTS (COMPONENTS)

A la droite de Viewer se trouve la liste des composants. Chaque fois que tu ajoutes un bouton, une image ou un lecteur audio, il apparaît dans cette liste.

Cette zone représente l'arborescence de ton application.

Par exemple : Screen1 / Bienvenue / BoutonBonjour



Cette liste est très importante.

Elle permet notamment de :

- Sélectionner un composant ;
- Le supprimer ;
- Le renommer ;
- Connaître sa position.

J. LES PROPRIETES (PROPERTIES)

A droite de l'écran se trouve le panneau des propriétés.

Lorsque tu sélectionnes un composant, toutes ses caractéristiques apparaissent. Si tu prends pour exemple un bouton, tu peux modifier :

- Le texte
- La couleur
- La largeur
- La hauteur
- La police
- La taille des caractères
- Les coins arrondis
- L'ombre
- L'icône
- La visibilité

Tout cela sans écrire une seule ligne de code.

K. AJOUTER UN COMPOSANT

Nous allons maintenant créer notre première interface.

- Dans la Palette, sélectionnons **User Interface**

- Glissons le composant **Label** au centre de **Viewer**
  Le label apparaît immédiatement dans les zones **Viewer**, **Components** et **Properties**
- Modifions la zone **Properties**/**Text** en remplaçant *Text for Label1* par *Bienvenue dans Kodular !*
- Ajoutons ensuite un **Button**
- Ajoutons une **Image**
- Observe leur apparition dans Components. Tu comprends maintenant la relation entre Palette → Viewer → Components → Properties.

L. SAUVEGARDE D'UN PROJET

Contrairement à certains logiciels, Kodular sauve automatiquement ton projet. Cependant il est conseillé d'effectuer régulièrement une sauvegarde locale.

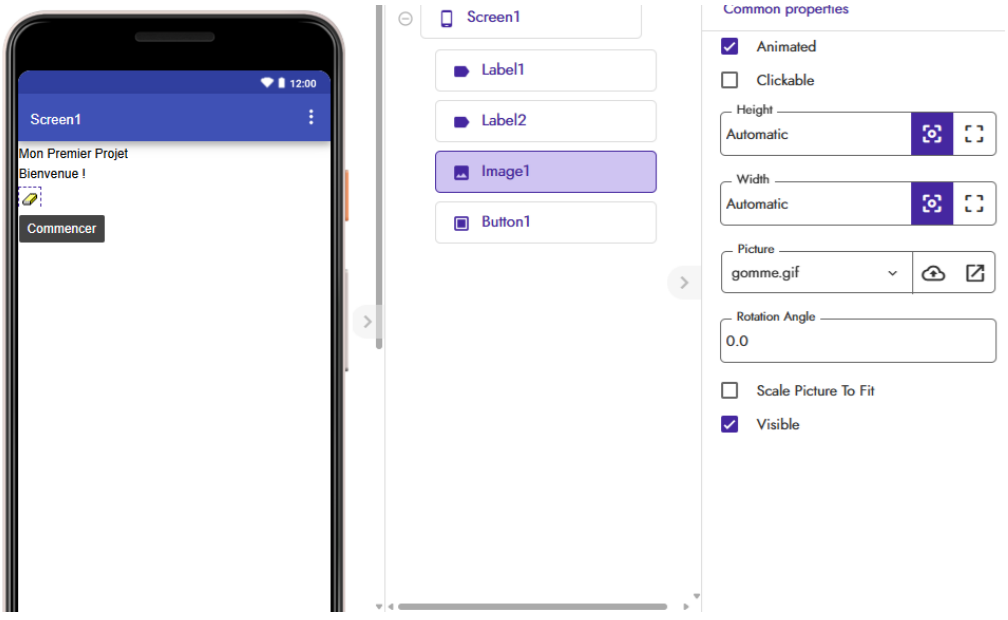
Pour cela :

- **Projects**
- **Export Selected Project (.aia) to my computer**
  Tu obtiens ainsi un fichier que tu pourras conserver sur ton ordinateur.

Bonne pratique Prends l'habitude d'exporter ton projet avant chaque modification importante. Si une erreur survient, tu pourras toujours revenir à une version précédente.

M. MINI-PROJET : MA PREMIERE INTERFACE

Reproduis l'écran suivant :



Consignes :

- Ajoute un Label pour le titre (Mon Premier Projet).
- Ajoute un second Label pour le message (Bienvenue !).
- Insère une Image (un logo ou une photo de ton choix).

- Ajoute un Button intitulé (Commencer).
  Ne t'occupe pas encore du comportement du bouton : nous verrons la programmation dans les prochaines parties.

2. MAITRISER LE DESIGNER ET LES COMPOSANTS VISIBLES

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre le rôle du **Designer** ;
- Distinguer les différents types de composants visibles ;
- Organiser correctement une interface ;
- Créer une mise en page professionnelle ;
- Eviter les erreurs de conception les plus fréquentes.

B. QU'EST-CE QUE LE DESIGNER ?

Le **Designer** est l'espace de travail dans lequel tu construis l'apparence de ton application. On peut le comparer à l'architecte d'une maison. Avant de penser à l'électricité ou à la plomberie, il faut dessiner les pièces. Le Designer joue exactement ce rôle.

C'est ici que tu décides :

- Où placer les boutons ;
- Où afficher les textes ;
- Quelles images utiliser ;
- Quelles couleurs choisir ;
- Comment les éléments seront organisés.

Le Designer ne contient aucune programmation. Il sert uniquement à créer l'interface graphique.

Interface graphique ou logique ?
Une application est composée de deux parties complémentaires.

L'**interface graphique**, ce que voit l'utilisateur :

- Les boutons ;
- Les images ;
- Les champs de saisie ;
- Les listes ;
- Les menus.

La **logique**, ce que l'utilisateur ne voit pas :

- Les calculs ;
- Les conditions ;
- Les variables ;
- Les traitements ;
- Les accès aux bases de données.

Dans Kodular :

- Designer = interface graphique
- Blocks = logique

Cette séparation rend les projets plus faciles à comprendre et à maintenir.

C. LES COMPOSANTS VISIBLES

Kodular propose de nombreux composants visibles.

Voici les principaux :

- **Button**
  Le bouton permet à l'utilisateur d'effectuer une action. Mais il ne fait rien tant qu'on ne lui associe pas un comportement dans les blocks.
  Exemples : Valider, Envoyer, Calculer, Enregistrer.

- **Checkbox**
  La case à cocher permet de répondre par Oui ou Non.

- **Date Picker**
  Permet de choisir une date.

- **Image**
  Une image améliore souvent l'apparence d'une application, mais il faut éviter d'en abuser.
  Exemples : une photo, un logo, une illustration, une icône.

- **Label**
  Le Label sert à afficher un texte. Il n'est généralement pas modifiable par l'utilisateur mais peut l'être par une commande liée à un bouton,…
  Exemples : un titre, une information, un résultat, une consigne.

- **Slider**
  Le Slider permet de sélectionner une valeur au moyen d'un curseur que l'on glisse sur une ligne dans un certain intervalle de valeurs.

- **Switch**
  Le Switch est un interrupteur. Il est très utilisé dans les paramètres.
  Modes ON/OFF

- **Text Box**
  La Text Box permet à l'utilisateur de saisir du texte.
  Exemples : un nom, une adresse e-mail, un mot de passe, une recherche.

- **Time Picker**
  Permet de sélectionner une heure.

D. LES COMPOSANTS DE MISE EN PAGE (LAYOUTS)

Une erreur très fréquente chez les débutants est le fait de déposer les composants directement sur l'écran. Résultat, une interface difficile à modifier. Une bonne application commence toujours par une bonne organisation.

Pour cela, Kodular fournit des composants appelés **Layouts** répartis en quatre catégories :

- **General**
- **Lists**
- **Navigation**
- **Views**

Les layouts (ou conteneurs de mise en page) servent à organiser les composants de ton application. Sans eux, tous les composants seraient simplement placés les uns sous les autres.

Les layouts permettent :

- De créer des interfaces modernes ;
- D'organiser les écrans ;
- D'obtenir des applications adaptables à toutes les tailles d'écran ;
- De faciliter la navigation.

GENERAL

Les composants de cette catégorie servent principalement à organiser les éléments visibles de l'application.

- ***GENERAL / Horizontal Arrangement***
  Dispose tous les composants sur une seule ligne, de gauche à droite.
  Utilisations : barre d'outils, ligne de boutons, formulaire, menu horizontal
  Principales propriétés : Align Horizontal, Align Vertical, Width, Height, BackgroundColor, Visible, Clickable
  Événements : Click, Long Click
  Avantages : Très simple, Très rapide, Peu gourmand en mémoire
  Inconvénients : Peu adapté lorsqu'il y a beaucoup de composants.

- ***GENERAL / Horizontal Scroll Arrangement***
  Même principe que le Horizontal Arrangement mais un défilement horizontal apparaît lorsque les composants dépassent la largeur de l'écran.
  Utilisations : galerie photos, produits, films, menus

- ***GENERAL / Vertical Arrangement***
  Dispose les composants les uns sous les autres. C'est probablement le composant le plus utilisé dans Kodular.
  Utilisations : formulaire, pages de paramètres, listes, écrans principaux

- ***GENERAL / Vertical Scroll Arrangement***
  Très utilisé. Il permet de créer une page beaucoup plus longue que l'écran.

- ***GENERAL / Table Arrangement***
  Dispose les composants sous forme de tableau. On choisit le nombre de lignes et le nombre de colonnes.
  Utilisations : tableaux, planning, jeux, calculatrices

- ***GENERAL / Card View***
  Le Card View est une carte Material Design. Elle possède une ombre, une élévation et éventuellement des coins arrondis.
  Utilisations : fiches produits, profils, recettes, actualités

- ***GENERAL / Grid View***
  Le Grid View affiche les éléments sous forme de grille.
  Utilisations : galerie, boutique, album photo, catalogue
  Attention, son utilisation est un peu particulière et dépasse le cadre de ce cours.

- ***GENERAL / Swipe Refresh Layout***
  Permet d'actualiser une page en tirant celle-ci vers le bas. Très utilisé dans Gmail, Facebook, Instagram

- ***GENERAL / Space***
  Composant invisible. Il sert uniquement à ajouter un espace entre deux composants. Très pratique pour aérer une interface.

LISTS

Cette catégorie est consacrée aux listes de données.

- ***LISTS / List View***
  Affiche une liste verticale.
  Fonctionnalités : recherche, sélection, défilement, clic, clic long, personnalisation des couleurs, hauteur des éléments, barre de recherche
  Événements : AfterPicking, LongClick
  Méthodes : Clear List, Scroll To Position
  Utilisations : contacts, pays , fichiers, menus

- ***LISTS / List View Image And Text***
  Ce composant est semblable à List View mais permet d'afficher une image et un texte à chaque ligne de la liste. De ce fait, son initialisation diffère fortement d'une simple liste.
  Attention, son utilisation est un peu particulière et dépasse le cadre de ce cours.

NAVIGATION

Ces composants permettent de passer facilement d'une page ou d'une section à une autre.

- ***NAVIGATION / Bottom Navigation***
  Barre de navigation située au bas de l'écran.
  Très utilisée dans : YouTube, Spotify, Instagram, Google Photos.
  Avantages : Très intuitive, Navigation rapide.

- ***NAVIGATION / Side Menu Layout***
  Aussi appelé Navigation Drawer.
  Menu qui apparaît depuis le côté gauche.
  Très utilisé lorsque l'application possède beaucoup de pages.

- ***NAVIGATION / Tab Layout***
  Affiche plusieurs onglets.
  L'utilisateur peut toucher un onglet ou faire glisser l'écran.
  Fonctions principales : ajout dynamique d'onglets, icônes facultatives, sélection par nom ou position, mode fixe ou défilant.
  Utilisations : paramètres, catégories

- ***NAVIGATION / View Pager***
  Le View Pager permet de faire défiler plusieurs pages horizontalement.
  Très utilisé pour les tutoriels

VIEWS

Cette catégorie regroupe des composants destinés à afficher des contenus spécialisés dont :

- ***VIEWS / Web View***
  Affiche une page Web directement dans l'application.
  Peut servir à afficher : un site Internet, une aide en ligne

- ***VIEWS / Surface View***
  Composant spécialisé permettant d'effectuer un affichage graphique performant. Il est particulièrement adapté aux applications nécessitant des rafraîchissements rapides.
  Utilisations : jeux, animations, graphiques temps réel, visualisations

LES BONNES PRATIQUES

- Utilise Vertical Arrangement comme conteneur principal dans la majorité des écrans.
- Évite de multiplier les niveaux d'imbrication des layouts : une hiérarchie trop profonde peut rendre l'interface plus difficile à maintenir et moins performante.
- Préfère Vertical Scroll Arrangement pour les écrans dont le contenu peut dépasser la hauteur de l'appareil.
- Réserve Bottom Navigation aux applications comportant 3 à 5 sections principales. Au-delà, un Side Menu Layout est souvent plus approprié.
- Utilise Card View pour mettre en valeur des informations importantes (produits, articles, profils).

- Teste systématiquement l'affichage sur plusieurs tailles d'écran pour vérifier que les layouts restent adaptés.

CONSEIL DE DÉVELOPPEUR

Utilise les Arrangements comme des boîtes de rangement. Chaque groupe d'éléments doit être placé dans un conteneur logique. Tu obtiendras une interface plus propre et plus facile à modifier.

E. LES PROPRIETES IMPORTANTES

Chaque composant possède des propriétés.

Voici les principales :

- **Width (Largeur)**
  Détermine la largeur.
  Trois choix sont courants : automatique ; remplir le parent (Fill Parent) ; largeur personnalisée.
- **Height (Hauteur)**
  Même principe.
- **Visible**
  Permet de masquer un composant.
  Il pourra être affiché plus tard grâce aux blocks.
- **Enabled**
  Si cette propriété est désactivée, le bouton apparaît mais ne peut plus être utilisé.
- **Background Color**
  Définit la couleur d'arrière-plan.
- **Font Size**
  Détermine la taille du texte.
- **Text Color**
  Choisit la couleur du texte.
- **Corner Radius**
  Arrondit les coins d'un bouton.
  Les applications modernes utilisent généralement des coins légèrement arrondis.

F. NOMMER CORRECTEMENT SES COMPOSANTS

Kodular attribue automatiquement des noms comme : Button1 Button2 Button3. Ce n'est pas une bonne pratique. Imagine une application contenant 150 boutons. Comment retrouver Button87 ? Il vaut mieux renommer immédiatement les composants.

Par exemple : btnConnexion btnCalcul btnEnvoyer lblTitre lblResultat txtNom imgLogo

Cette habitude te fera gagner énormément de temps lorsque tu commenceras à programmer.

G. EXERCICE GUIDE : CREER UN ECRAN D'ACCUEIL

Nous allons réaliser une page d'accueil simple :

- Ajoute un **Vertical Arrangement**.
- Règle sa largeur sur **Fill Parent**.
- Ajoute un **Label**.
  Texte : Bienvenue
  Police : Taille : 24
  Gras : Oui
- Alignement : Centre
- Ajoute une **Image**.
  Choisis un logo ou une illustration.
- Ajoute un second Label.
  Texte : Bienvenue dans votre première application créée avec Kodular.
- Ajoute deux boutons : Commencer et Quitter

À ce stade, les boutons ne réalisent encore aucune action. Nous leur donnerons un comportement dans les prochains chapitres.

H. COMMENT TESTER SON APPLICATION ?

- Installe **Kodular companion** sur ton smartphone
- Dans Kodular, quand ton projet est ouvert, exécute la commande **Test** / **Connect to companion**
- Sur ton smartphone, démarre **Kodular companion**
- Clique sur **Scanner le code QR**
- Scanne le code QR de Kodular

I. EXERCICE

Crée une interface intitulée **Mon Carnet d'Adresses** comprenant :

- Une organisation réalisée uniquement avec des Vertical Arrangement et des Horizontal Arrangement.
- Un titre centré ;
- Une image représentant un carnet ;
- Deux champs de saisie : Nom et Téléphone ;
- Trois boutons : Ajouter, Rechercher et Supprimer.

3. LES PREMIERS BLOCKS : DONNER VIE A UNE APPLICATION

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre ce qu'est un block ;
- Distinguer un événement d'une action ;

- Créer ton premier programme ;
- Modifier les propriétés d'un composant par programmation ;
- Tester ton application sur un téléphone Android.

B. DU DESIGNER AUX BLOCKS

Jusqu'à présent, nous avons travaillé dans le **Designer**. Nous avons créé une belle interface composée de boutons, d'images, de textes et de zones de saisie. Pourtant, si tu testes ton application maintenant, tu constateras qu'il ne se passe absolument rien. Les boutons sont présents, mais ils ne réagissent pas.

Pourquoi ? Parce qu'une interface seule ne suffit pas. Il faut maintenant indiquer à l'application comment elle doit réagir lorsque l'utilisateur effectue une action. C'est le rôle de l'éditeur **Blocks**.

C. QU'EST-CE QU'UN BLOCK ?

Un block est une instruction graphique.

Au lieu d'écrire en Java `if (buttonClicked) { label.setText("Bonjour"); }`, Kodular représente cette logique avec des pièces qui s'assemblent.

Chaque block possède une fonction précise.

Par exemple :

- Afficher un texte ;
- Effectuer un calcul ;
- Comparer deux valeurs ;
- Jouer un son ;
- Ouvrir un écran.

Les blocks sont conçus comme des pièces de puzzle. Ils ne peuvent être assemblés que si leur forme est compatible. Cette conception limite fortement les erreurs de programmation.

D. LES DIFFERENTES CATEGORIES DE BLOCKS

Lorsque tu ouvres l'éditeur **Blocks**, tu remarques une colonne située à gauche. Elle contient plusieurs catégories.

Les plus importantes sont :

- Les blocks des composants
  Chaque composant possède ses propres blocks.
  Exemple : Button1 Label1 Image1 TextBox1
  En cliquant sur Button1, tu obtiens les blocks liés à ce bouton.
- **Control**
  Cette catégorie contient les structures de contrôle.
  Exemples : si..., sinon..., répéter..., attendre...
  Nous les étudierons en détail plus tard.

- **Logic**
  On y trouve les valeurs : Vrai (**True**) et Faux (**False**)
  Ainsi que les opérateurs logiques : ET (**And**), OU (**Or**) et NON (**Not**)
- **Math**
  Tous les calculs, Addition, Racine carrée, Nombre aléatoire, etc.
- **Text**
  Manipulation des chaînes de caractères, Concaténation, Longueur, Recherche
- **Lists**
  Création et manipulation de listes.
  Très utile pour les carnets d'adresses ou les listes de courses.
- **Variables**
  Création de variables. Nous y reviendrons dans un chapitre entier.
- **Procedures**
  Création de fonctions personnalisées.

E. COMPRENDRE LES EVENEMENTS

Les événements sont au cœur de la programmation avec Kodular. Un événement est une action détectée par l'application.

Par exemple :

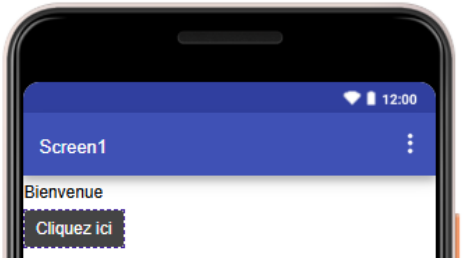
- L'utilisateur clique sur un bouton ;
- L'écran s'ouvre ;
- Une image est sélectionnée ;
- Un SMS est reçu ;
- La position GPS change.

Chaque événement peut déclencher une ou plusieurs actions.

F. NOTRE PREMIER PROGRAMME

Nous allons maintenant programmer notre premier comportement.

Dans le **Designer**, crée l'interface suivante :



- Le texte est affiché dans un **Label**.
- Le texte du **Button** est *Cliquez ici*.

Passe maintenant dans l'onglet **Blocks** :

- Clique sur **Button1** et choisis **when Button1.Click** qui signifie *Lorsque l'utilisateur clique sur le bouton*. À lui seul, il ne fait encore rien. Il attend que nous ajoutions des instructions.



- Clique maintenant sur **Label1** et glisse le block **set Label1.Text to** à

l'intérieur du block précédent.

- Ajoute ensuite un block **Text** et écris-y le texte *Bonjour !*

Tu obtiens ainsi la logique suivante : Lorsque le Button1 est cliqué, modifie le libellé de la zone Text par Bonjour !

Ton premier programme est terminé.

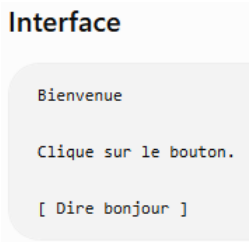
Il te reste à présent à tester ton application comme décrit précédemment en utilisant **Kodular companion**.

G. BONNES PRATIQUES

- Programme une seule chose à la fois. Teste chaque nouvelle fonctionnalité avant d'en ajouter une autre.
- Sauvegarde régulièrement. Exporte ton projet au format **.aia**.
- Renomme tes composants. Préfère : btnValider lblBienvenue à Button1 Label1

H. EXERCICE : APPLICATION DE BIENVENUE

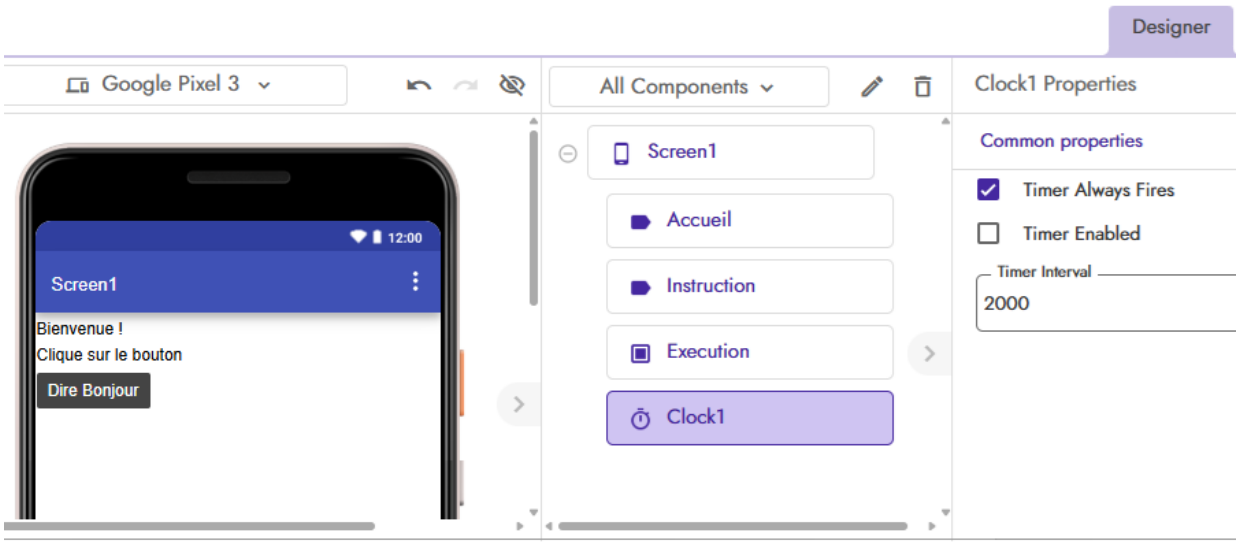
Crée une petite application :

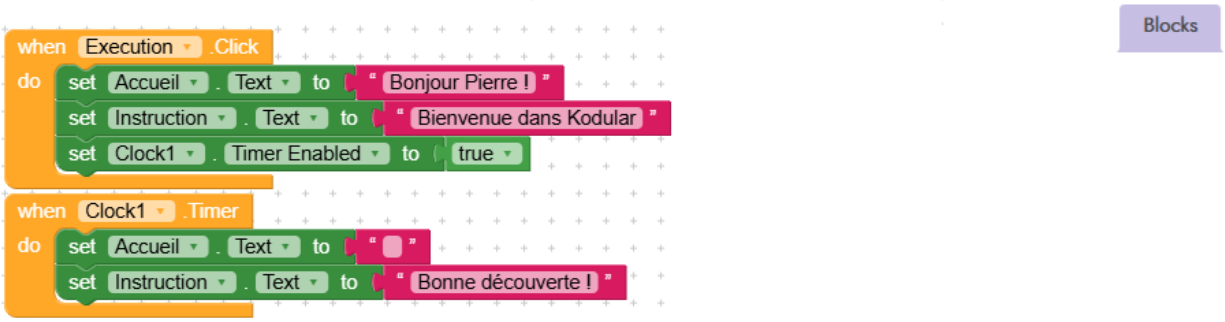


- **Interface** : voir ci-contre
- **Fonctionnement** : Lorsque l'utilisateur clique sur le bouton, les textes deviennent *Bonjour Pierre !* et *Bienvenue dans Kodular*.

Bonus : Après quelques secondes, remplace le texte par *Bonne découverte !*
Remarque : Cette dernière fonctionnalité utilisera le composant (**Sensors**/)**Clock**, que nous découvrirons plus en détail dans un chapitre ultérieur.

Réponse :





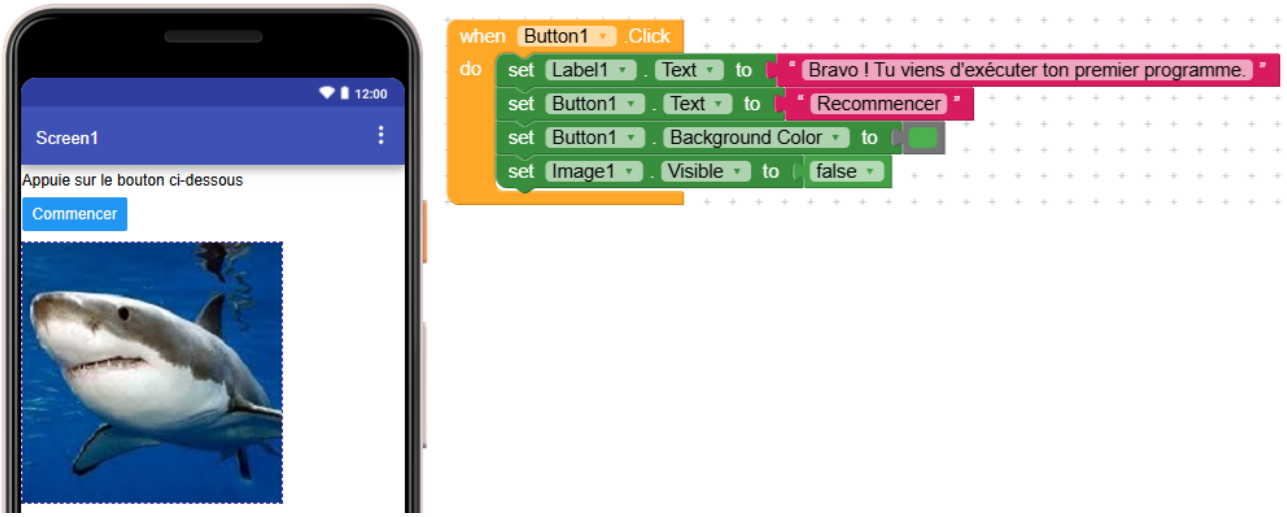
I. ATELIER PRATIQUE : TON PREMIER DEFI

Objectif : créer une application interactive.

- **Interface**
  Un **Label** affichant *Appuie sur le bouton ci-dessous*.
  Un **Button** intitulé *Commencer*.
  Une **Image** (par exemple un logo ou un smiley).

- **Fonctionnement**
  Lorsqu'on clique sur le bouton :
  - Le texte du Label devient *Bravo ! Tu viens d'exécuter ton 1er programme.*
  - Le texte du bouton devient *Recommencer*.
  - La couleur du bouton passe du bleu au vert.
  - L'image devient invisible.

Réponse :



J. LES PRINCIPALES ERREURS DES DEBUTANTS

1. **Oublier de renommer les composants**
   Au bout de plusieurs écrans, on ne sait plus quel bouton est concerné.
2. **Utiliser trop de couleurs**
   Une belle interface reste sobre.

3. **Ecrire tout en MAJUSCULES**
   Difficile à lire.
4. **Utiliser des images trop lourdes**
   Une application devient lente.
5. **Ne jamais tester**
   Teste après chaque modification.
6. **Tout programmer d'un seul coup**
   Programme une fonctionnalité.
   Teste.
   Puis passe à la suivante.
7. **Ne jamais commenter son projet**
   Même si Kodular est graphique, prends l'habitude d'ajouter des commentaires dans les blocks lorsque la logique devient plus complexe (voir ci-après)
8. **Ne pas faire de sauvegarde**
   Exporte régulièrement ton projet via la commande **Project** / **Export selected project (.aia) to my computer**.
9. **Vouloir créer une grande application dès le début**
   Commence toujours par de petits projets.

Dans Kodular, tu peux ajouter un commentaire directement dans la partie Blocks pour expliquer ce que fait une partie de ton programme :

1. Fais un **clic droit** sur le block que tu veux commenter.
2. Dans le menu qui apparaît, sélectionne **Add Comment**.
3. Un petit espace de commentaire apparaît près du block. Si ce n'est pas le cas, il faut cliquer sur le bouton **"?"** apparu sur le block.
4. Clique dans cet espace et écris ton commentaire.

On peut demander à Kodular d'afficher ou de masquer tous les commentaires de la partie Blocks en cliquant sur le bouton droit de la souris sur une zone vide de la partie Blocks et d'exécuter la commande **Show All Comments** ou **Hide All Comments**.

K. BONNES PRATIQUES DES DEVELOPPEURS

Les développeurs expérimentés suivent presque toujours la même méthode :

1. Réfléchir au projet.
2. Dessiner l'interface.
3. Nommer correctement les composants.
4. Construire les écrans.
5. Programmer progressivement.
6. Tester.
7. Corriger.
8. Optimiser.
9. Sauvegarder.
10. Publier.

Cette méthode permet de gagner un temps considérable.

L. LEXIQUE DU CHAPITRE

Terme	Définition
Application	Programme destiné à être exécuté sur un appareil Android.
Designer	Espace où l'on construit l'interface graphique.
Blocks	Espace où l'on programme les comportements de l'application.
Composant	Élément utilisé dans une application (bouton, image, texte, etc.).
Evénement	Action détectée par l'application (clic, ouverture d'écran, saisie...).
Propriété	Caractéristique d'un composant (couleur, texte, taille, visibilité...).
Label	Composant qui affiche du texte.
Button	Composant permettant à l'utilisateur de déclencher une action.
Arrangement	Conteneur servant à organiser les composants dans l'interface.
APK	Fichier permettant d'installer une application Android.
AIA	Fichier de projet Kodular contenant tous les éléments de l'application.

Maîtriser le Designer

1. CONSTRUIRE UNE INTERFACE PROFESSIONNELLE

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre le rôle du Designer dans un projet Kodular ;
- Distinguer une interface amateur d'une interface professionnelle ;
- Configurer correctement un écran (Screen) ;
- Utiliser les principaux paramètres de Screen1 ;
- Concevoir une interface adaptée à différentes tailles d'écran.

B. INTRODUCTION

Au chapitre précédent, tu as découvert les bases de Kodular et créé ta première application. Même si cette application fonctionnait correctement, son apparence restait très simple.

Dans ce chapitre, nous allons apprendre à créer des interfaces agréables, modernes et professionnelles. En effet, une application n'est pas jugée uniquement sur ses fonctionnalités. Les utilisateurs décident souvent en quelques secondes s'ils souhaitent continuer à l'utiliser. Une interface soignée inspire confiance. Une interface mal conçue donne immédiatement une impression d'amateurisme.

Le rôle du Designer est précisément de créer cette première impression.

C. LE PLAN DE TON APPLICATION

Avant de programmer quoi que ce soit, tu dessines ton application.

Tu décides :

- Où placer les boutons ;
- Où afficher les informations ;
- Quelles images utiliser ;
- Quelles couleurs choisir ;
- Quelle disposition adopter.

Un développeur professionnel commence presque toujours par concevoir l'interface avant d'écrire le moindre bloc.

D. LES ELEMENTS DU DESIGNER

Comme expliqué précédemment, lorsque tu ouvres un projet Kodular, plusieurs panneaux apparaissent : La **Palette**, le **Viewer**, le **Components** qui reprend l'arborescence des composants et les **Properties**. Ils travaillent ensemble.

E. SCREEN1

Lorsqu'un nouveau projet est créé, un premier écran apparaît. Il s'appelle Screen1. C'est l'écran principal de l'application. Tous les composants visibles sont placés à l'intérieur. Plus tard, une application pourra contenir plusieurs écrans.

Les propriétés de Screen1

Sélectionne **Screen1**. Tu observes une longue liste de propriétés. Certaines sont particulièrement importantes :

- **App Name** (accessible via le bouton **Roue dentée**)
  Correspond au nom affiché sous l'icône de l'application. Il ne faut pas le confondre avec le nom du projet.

- **About Screen** …
  Ces propriétés sont liées à l'écran de l'application affiché lorsque l'utilisateur sélectionne l'option "A propos de l'application"

- **Title**
  Texte affiché dans la barre supérieure de l'application. Certaines applications masquent complètement cette barre (au moyen de l'option **Title Visible**).

- **Background Color**
  Détermine la couleur de fond. Par défaut : Blanc.

- **Scrollable**
  Cette propriété est très importante.
  False : L'écran reste fixe. Aucun défilement.
  True : L'utilisateur peut faire défiler la page.

- **Align Horizontal**
  Détermine l'alignement horizontal des composants.
  Valeurs possibles : Gauche, Centre et Droite.

- **Align Vertical**
  Détermine la position verticale : Haut, Centre et Bas.

- **Show Status Bar**
  Affiche ou masque la barre d'état Android.
  La masquer permet de gagner quelques pixels d'affichage, mais l'utilisateur ne voit plus l'heure ni le niveau de batterie.

F. LES DIFFERENTES TAILLES D'ECRAN

Un des défis du développement Android est la diversité des appareils.

Ton application pourra être utilisée sur :

- Un smartphone de 5 pouces ;
- Une tablette de 11 pouces ;
- Un écran pliable ;

- Un téléviseur Android (dans certains cas).

Une interface bien conçue doit rester lisible partout.

Une mauvaise pratique consiste à créer une interface en positionnant les composants "à l'œil" car sur certains téléphones différents du nôtre, les boutons se chevaucheront, les textes seront coupés, …

Une bonne pratique sera de toujours utiliser des Arrangements (conteneurs de mise en page). Ils adaptent automatiquement l'organisation des composants. C'est l'une des clés d'une application professionnelle.

G. LE SYSTEME DE COORDONNEES

Contrairement à certains logiciels de dessin, Kodular ne fonctionne pas avec des coordonnées X et Y pour placer chaque composant. Il utilise une approche basée sur des conteneurs.

Cette méthode présente plusieurs avantages :
- Adaptation automatique aux différentes tailles d'écran ;
- Interface plus simple à maintenir ;
- Meilleure compatibilité avec les mises à jour Android.

Tu verras dans les prochains chapitres comment exploiter pleinement les **Vertical Arrangement**, **Horizontal Arrangement** et **Table Arrangement**.

H. ATELIER PRATIQUE

Objectif : Créer un écran d'accueil

- Crée un nouveau projet intitulé *MonInterface*

- Configure Screen1 avec les paramètres suivants :
  - **Title** Mon Interface
  - **Background Color** Blanc
  - **Align Horizontal** Center
  - **Align Vertical** Top
  - **Scrollable** False

- Ajoute ensuite :
  - Un **Label** intitulé *Bienvenue* ;
  - Une **Image** (un logo ou une illustration) ;
  - Un **Button** nommé *Continuer*.

- Ne programme aucun comportement mais teste le résultat

I. ASTUCE DE DEVELOPPEUR

Avant de commencer un projet important, dessine toujours ton application sur papier ou avec un logiciel de maquettage.

Un simple croquis permet souvent de détecter des problèmes d'organisation avant même de créer le premier composant.

J. LES ERREURS FREQUENTES

- Ajouter immédiatement des dizaines de composants.
  Construis ton interface progressivement.
- Modifier toutes les propriétés sans savoir à quoi elles servent.
  Teste chaque modification individuellement.
- Oublier les marges et les espaces.
  Une interface aérée est plus agréable à utiliser.
- Utiliser des couleurs trop vives partout.
  Choisis une palette de deux ou trois couleurs principales.

2. LES COMPOSANTS DE MISE EN PAGE (LAYOUTS)

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre le rôle des composants de mise en page ;
- Utiliser correctement les Arrangements de Kodular ;
- Créer des interfaces qui s'adaptent à tous les écrans ;
- Organiser proprement tes composants ;
- Construire une interface moderne et facile à maintenir.

B. INTRODUCTION

Les composants doivent être rangés dans des conteneurs. Ceux-ci sont appelés **Layouts** ou **Arrangements**. Ils sont parmi les composants les plus importants de Kodular. Pourtant, ce sont aussi les plus mal utilisés par les débutants.

Sans Arrangement, chaque composant serait placé individuellement. Ce qui les rend difficiles à déplacer et/ou à adapter à une tablette. Avec un Arrangement, tous les composants restent organisés. Si tu déplaces le conteneur, tout son contenu suit automatiquement.

Un Arrangement est une boîte invisible qui sert à regrouper et organiser des composants.

C. LES DIFFERENTS ARRANGEMENTS

Kodular propose plusieurs types de conteneurs.

Les quatre principaux sont :

- Vertical Arrangement
- Horizontal Arrangement
- Table Arrangement
- Vertical Scroll Arrangement

Nous allons les découvrir un par un.

D. LE VERTICAL ARRANGEMENT

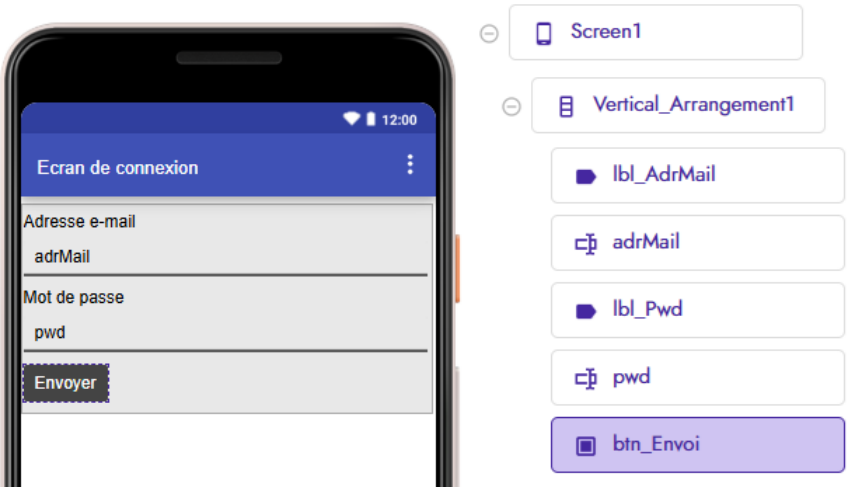
C'est le conteneur le plus utilisé. Tous les composants sont placés les uns sous les autres. Chaque nouvel élément apparaît sous le précédent.

Le Vertical Arrangement est idéal pour :

- Une page d'accueil ;
- Un formulaire ;
- Un menu ;
- Une fiche produit ;
- Une page de paramètres.

En pratique, la majorité des écrans d'une application utilisent au moins un Vertical Arrangement.

Exemple :
Construisons un écran de connexion.



Tous les éléments sont naturellement empilés. Le Vertical Arrangement est donc le meilleur choix.

E. LE HORIZONTAL ARRANGEMENT

Ce conteneur place les composants côte à côte.

Très utile pour :

- Deux boutons ;
- Une icône et un texte ;
- Une photo et une description ;
- Une ligne de formulaire.

F. LE TABLE ARRANGEMENT

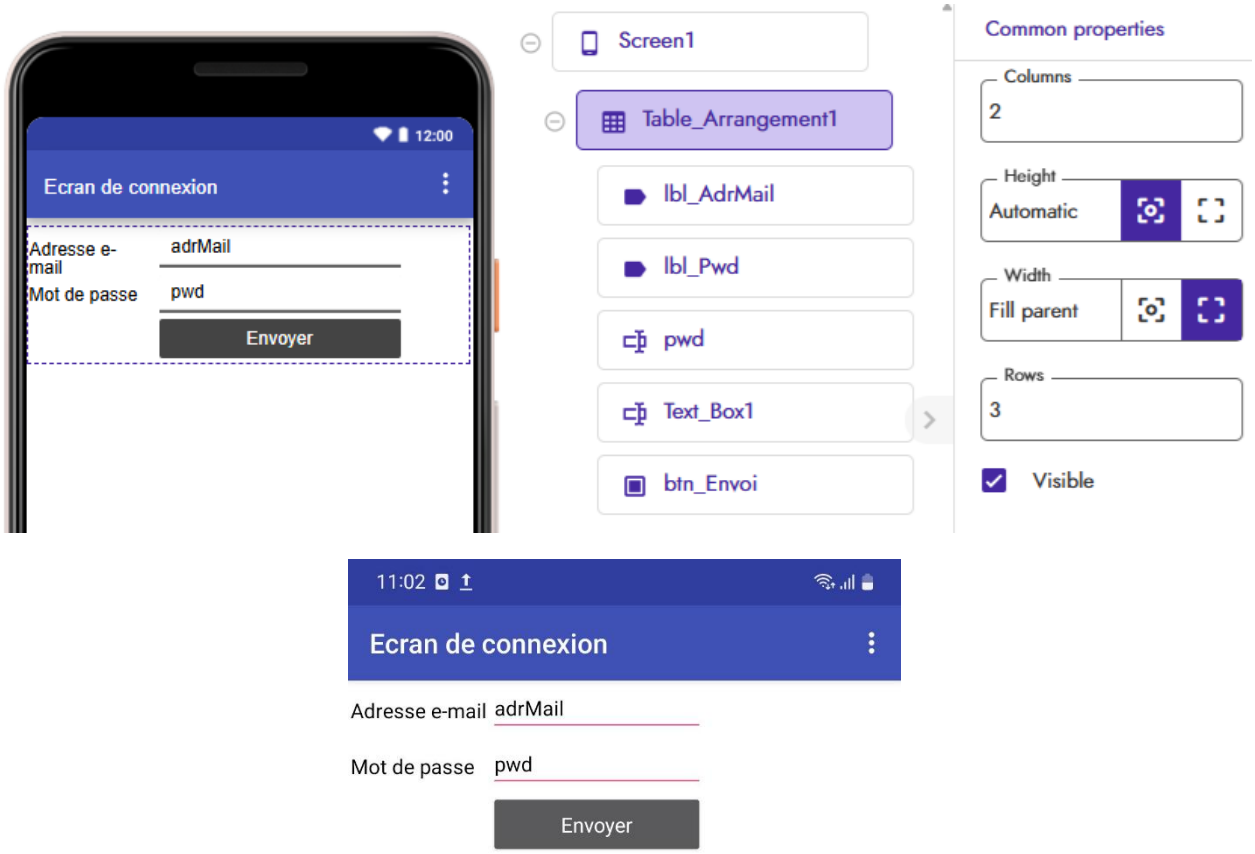
Le Table Arrangement organise les composants sous forme de tableau. Chaque ligne possède plusieurs colonnes.

Très pratique pour :

- Une fiche d'informations ;

- Un tableau de résultats ;
- Une grille de boutons ;
- Un formulaire ;
- Un calendrier.

Exemple :



G. LE VERTICAL SCROLL ARRANGEMENT

Imagine une application contenant une longue notice. Elle ne tient pas sur un seul écran. Le Vertical Scroll Arrangement permet alors de faire défiler le contenu au sein de cette zone indépendamment de la propriété Scrollable de l'écran. L'utilisateur fait simplement glisser son doigt vers le haut.

Applications typiques :

- Conditions générales.
- Recettes de cuisine.
- Articles.
- Guides.
- Catalogues.

H. LES PROPRIETES IMPORTANTES DES ARRANGEMENTS

Les Arrangements possèdent plusieurs propriétés essentielles dont :

- **Width**
  Largeur.
  Trois valeurs sont les plus utilisées :
  - **Automatic** -> La largeur dépend du contenu.
  - **Fill Parent** -> Le conteneur occupe toute la largeur disponible. C'est souvent la meilleure solution.
  - **Pourcentage** -> Permet de définir une largeur en pourcentage de l'écran. (En tapant, par exemple, 50% à la place du mot Automatic).

- **Height**
  Même principe. Attention à ne pas donner une hauteur fixe inutilement. Une hauteur automatique est souvent préférable.

- **Align Horizontal**
  Permet d'aligner les composants contenus dans le conteneur.

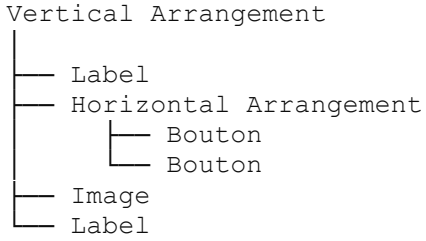
- **Align Vertical**
  Détermine leur position verticale.

- **Background Color**
  Très utile pour créer des zones visuellement distinctes.

I. IMBRIQUER LES ARRANGEMENTS

Un Arrangement peut contenir un autre Arrangement. C'est une notion fondamentale. Cette technique permet de créer des interfaces complexes tout en restant organisées. Cette méthode est utilisée dans pratiquement toutes les applications professionnelles.

Exemple :



J. LES MARGES ET LES ESPACEMENTS

Une erreur très fréquente consiste à coller tous les composants. Quand tout est serré, la lecture devient difficile. Quelques pixels d'espace suffisent à améliorer considérablement le rendu.

Pour ce faire, on peut soit ajouter des marges aux composants (généralement via des instructions Blocks), soit insérer un **Layout** / **General** / **Space** dont on redéfinit les propriétés **Height** et **Width** en précisant les valeurs souhaitées en pixels (**px**) ou en pourcentage.

K. CONCEVOIR UNE INTERFACE RESPONSIVE

Le terme **responsive** signifie que l'interface s'adapte automatiquement à la taille de l'écran. Grâce aux Arrangements et aux largeurs en pourcentage, Kodular adapte automatiquement la présentation.

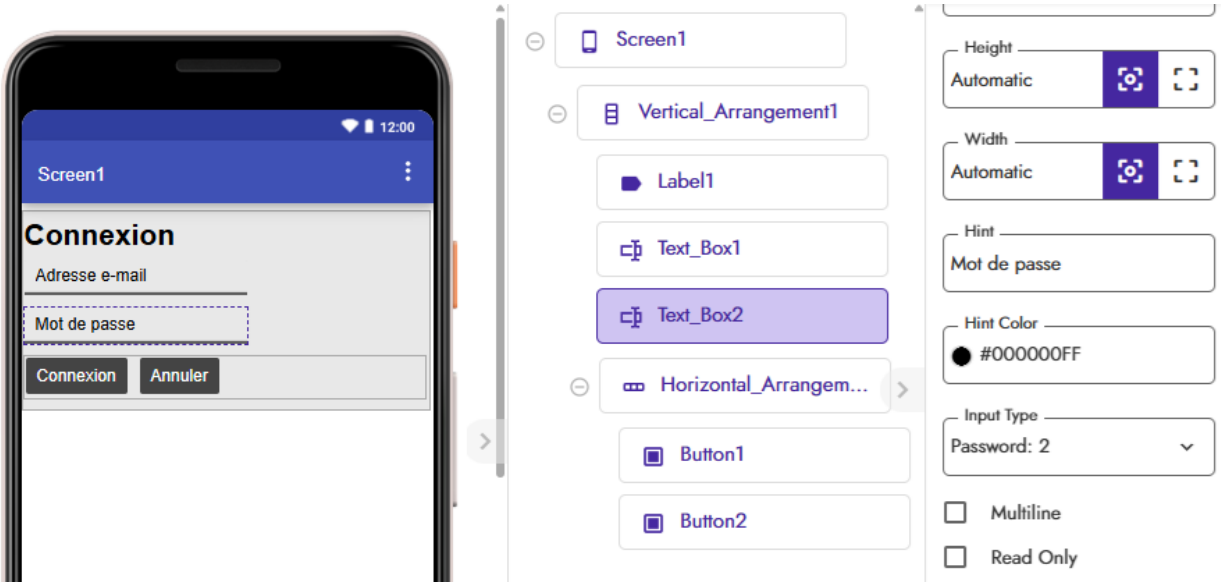
Ne fixe pas systématiquement une largeur ou une hauteur en pixels, mais privilégie une des options **Fill Parent**, **Automatic** ou un pourcentage.

Ton application sera compatible avec davantage d'appareils.

L. ATELIER PRATIQUE

Créons une page de connexion :

- Étape 1 - Ajoute un **Vertical Arrangement**.
  Largeur : **Fill Parent**.
- Étape 2 - Ajoute un **Label**.
  Texte : Connexion
  Police : 24
  Gras : Oui
- Étape 3 - Ajoute un **Text Box**.
  Hint : Adresse e-mail
- Étape 4 - Ajoute un second **Text Box**.
  Hint : Mot de passe
  Active la propriété **Input Type** / **Password** afin de masquer les caractères saisis
- Étape 5 - Ajoute un **Horizontal Arrangement**.
  À l'intérieur :
  - Bouton Connexion
  - Bouton Annuler



M. CONSEILS

Lorsque tu réalises une interface complexe, commence toujours par dessiner les **Arrangements**. Seulement ensuite, ajoute les boutons, les images et les textes. Les développeurs expérimentés pensent d'abord à la structure, puis au contenu.

Les erreurs fréquentes :

- Tout placer directement sur Screen1.
  Utilise des Arrangements.
- Créer un Horizontal Arrangement pour un seul composant.
  C'est inutile.
- Utiliser plusieurs dizaines d'Arrangements imbriqués sans raison.
  Une structure trop profonde complique la maintenance.
- Oublier les espaces.
  Une interface aérée est plus agréable.

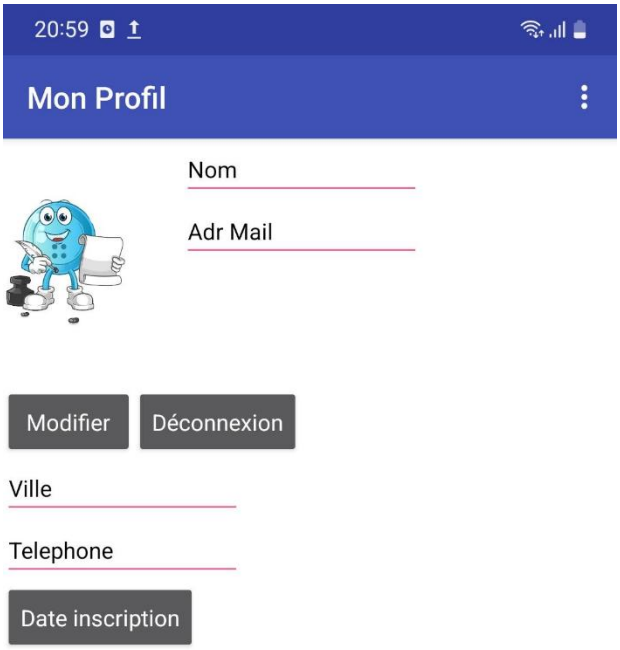
Dans cette partie, tu as appris à :

- Utiliser les principaux Arrangements de Kodular ;
- Choisir le conteneur adapté à chaque situation ;
- Organiser proprement une interface ;
- Créer une mise en page responsive ;
- Imbriquer des conteneurs pour construire des écrans plus élaborés.

Ces notions sont fondamentales. Une bonne architecture d'interface te fera gagner beaucoup de temps lorsque tu commenceras à développer des applications plus complexes.

N. EXERCICE

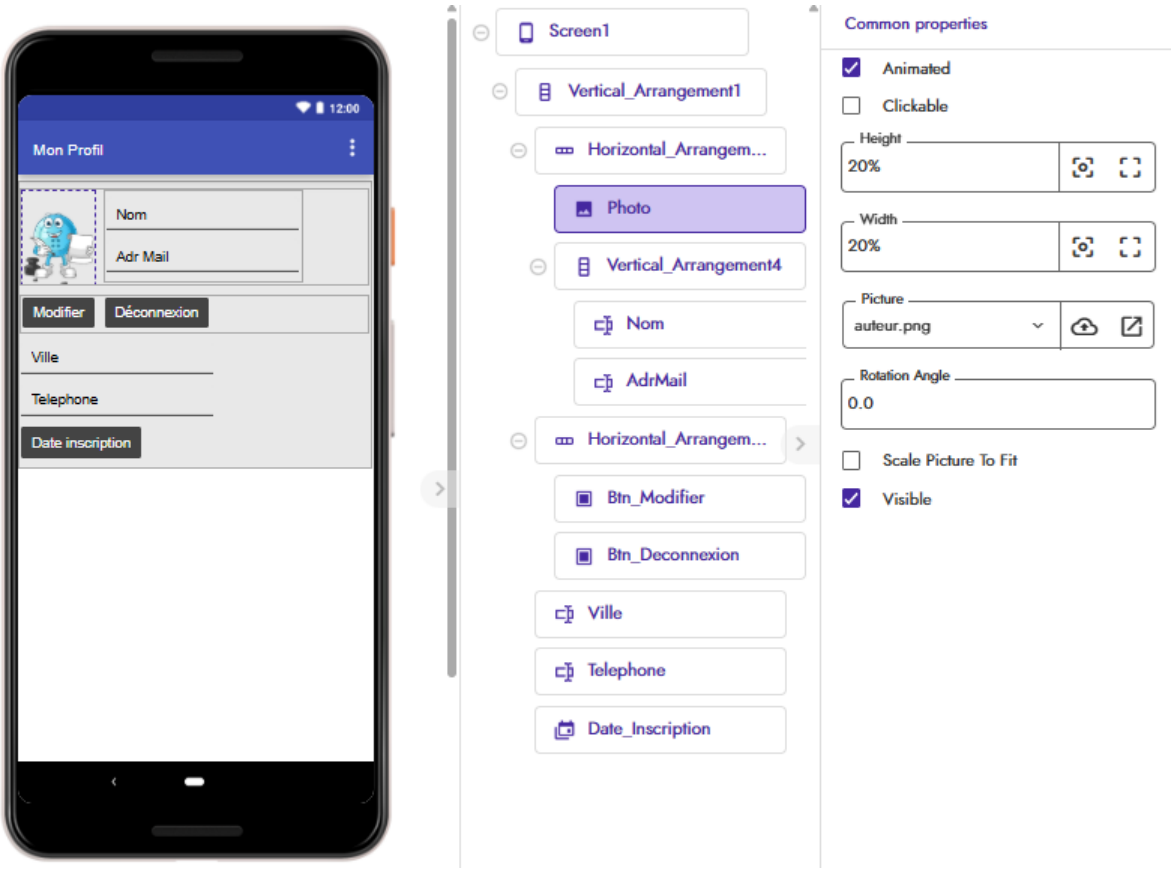
Réalise une interface intitulée **Mon Profil** contenant notamment :



- Un **Vertical Arrangement** principal ;
- Une photo (hauteur et largeur à 20%) ;
- Un nom ;
- Une adresse e-mail ;
- Les boutons *Modifier* et *Déconnexion* ;
- Un second **Vertical Arrangement** contenant trois informations (ville, téléphone, date d'inscription).

L'objectif est de manipuler plusieurs niveaux de conteneurs afin de créer une interface propre, évolutive et facile à maintenir.

Solution :



3. Les composants visibles de Kodular

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Connaître les principaux composants visibles de Kodular ;
- Choisir le composant le plus adapté à chaque situation ;
- Configurer leurs propriétés essentielles ;
- Comprendre leurs événements principaux ;
- Réaliser une interface ergonomique et agréable à utiliser.

B. INTRODUCTION

Les composants visibles sont les éléments que l'utilisateur voit et avec lesquels il interagit. Ils constituent l'interface graphique de ton application. Un bon choix de composants améliore considérablement l'expérience utilisateur.

Une bonne interface doit être :

- Claire ;
- Intuitive ;
- Cohérente ;
- Agréable.

Dans cette partie, nous allons découvrir les composants les plus utilisés.

C. LE LABEL

Le **Label** est probablement le composant le plus simple. Il sert uniquement à afficher un texte. L'utilisateur ne peut généralement pas le modifier.

Exemples d'utilisation : Titre d'une page, résultat d'un calcul, consigne, message d'erreur, score d'un jeu, nom d'un produit.

Les propriétés principales :

Propriété	Description
Background Color	Couleur du fond
Font Bold	Texte en gras
Font Italic	Texte en italique
Font Size	Taille de la police
Height	Hauteur
Text	Texte affiché
Text Alignment	Alignement du texte
Text Color	Couleur du texte
Visible	Afficher ou masquer le Label
Width	Largeur

D. LE BUTTON

Le **bouton** permet à l'utilisateur d'exécuter une action. C'est l'un des composants les plus importants.

Exemples : Connexion, Calculer, Supprimer, Valider, Envoyer.

Les propriétés principales :

Propriété	Description
Background Color	Couleur du bouton
Enabled	Actif ou non
Font Size	Taille des caractères
Image	Icône éventuelle
Shape	Forme
Text	Texte du bouton
Text Alignment	Alignement du texte au sein du bouton
Text Color	Couleur du texte
Visible	Visible ou caché

Evénements

L'événement le plus utilisé est **when Button.Click** qui est déclenché lorsque l'utilisateur appuie sur le bouton.

E. LA TEXT BOX

La **Text Box** permet à l'utilisateur de saisir une information.

Exemples : Nom, adresse, téléphone, e-mail, commentaire, recherche.

Les propriétés importantes

Propriété	Description
Enabled	Champ actif ou non (grisé)
Hint	Texte affiché lorsque le champ est vide
Input Type / Numbers Only	Accepte uniquement des chiffres
Input Type / Password	Masque les caractères
MultiLine	Autorise plusieurs lignes
Read Only	Uniquement en lecture
Text	Valeur saisie

Astuce

Utilise toujours un **Hint** afin d'indiquer à l'utilisateur ce qu'il doit saisir.

F. L'IMAGE

Le composant **Image** affiche une photo, un logo, une illustration, une icône.

Les propriétés importantes

Propriété	Description
Height	Hauteur
Picture	Image affichée
Rotation Angle	Rotation de l'image
Scale Picture to Fit	Adapter à la taille disponible
Width	Largeur

G. LA CHECK BOX

La **Check Box** permet de répondre **Oui** ou **Non** en cochant ou non la case.

Exemples :

☐ J'accepte les conditions.
☐ Recevoir la newsletter.
☐ Activer les notifications.

Evénement : when CheckBox.Changed

H. LE SWITCH

Le **Switch** fonctionne comme un interrupteur. Il est semblable aux cases à cocher mais est plus moderne.

I. LE SLIDER

Le **Slider** permet de choisir une valeur au moyen d'un curseur gradué.

Exemples : Volume sonore, luminosité, zoom

Propriétés : Minimum, maximum, position

Attention, si l'on veut qu'il n'accepte que des valeurs entières, il faut lui associer une commande Blocks **round** semblable à celle-ci-après.

Attention, si l'on veut afficher sa valeur à l'écran, il faut ajouter un **Label** ou une **Text Box** à l'écran et modifier sa valeur à chaque changement du **Slider** via une commande semblable à celle-ci après.



J. LE SPINNER

Le **Spinner** est une liste déroulante. Très utile lorsque plusieurs choix sont possibles.

Les valeurs disponibles sont ajoutées dans la propriété **Elements From String** en les séparant par une virgule.

K. LE LIST VIEW

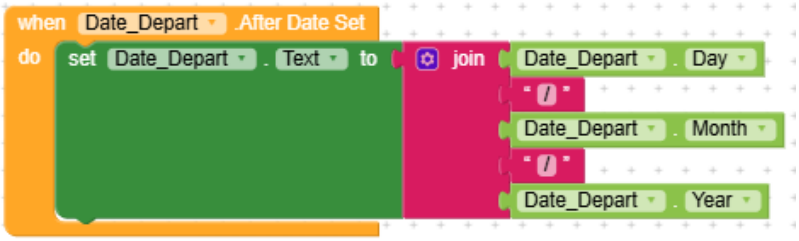
Le **List View** affiche une liste.

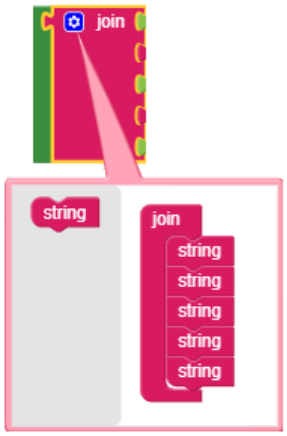
Evénement : AfterPicking qui est généré après la sélection d'un élément.

L. LE DATE PICKER

Le **Date Picker** affiche un calendrier qui permet à l'utilisateur de choisir une date.

Attention, si l'on veut qu'il retienne et affiche la date que l'utilisateur choisira, il faut y associer les commandes **Blocks** semblables à celles-ci :





Remarque :
Pour associer plus de 2 valeurs à la commande **Join**, il faut cliquer sur la **roue dentée bleue** associée et glisser l'icône **String** de gauche sur la partie droite autant de fois que souhaité.

M. LE TIME PICKER

Même principe, mais pour choisir une heure.

N. LE NOTIFIER

Le **Notifier** affiche des messages. Très utile pour dialoguer avec l'utilisateur.

Exemples : *Enregistrement réussi*, une confirmation, une question, une erreur.

O. CHOISIR LE BON COMPOSANT

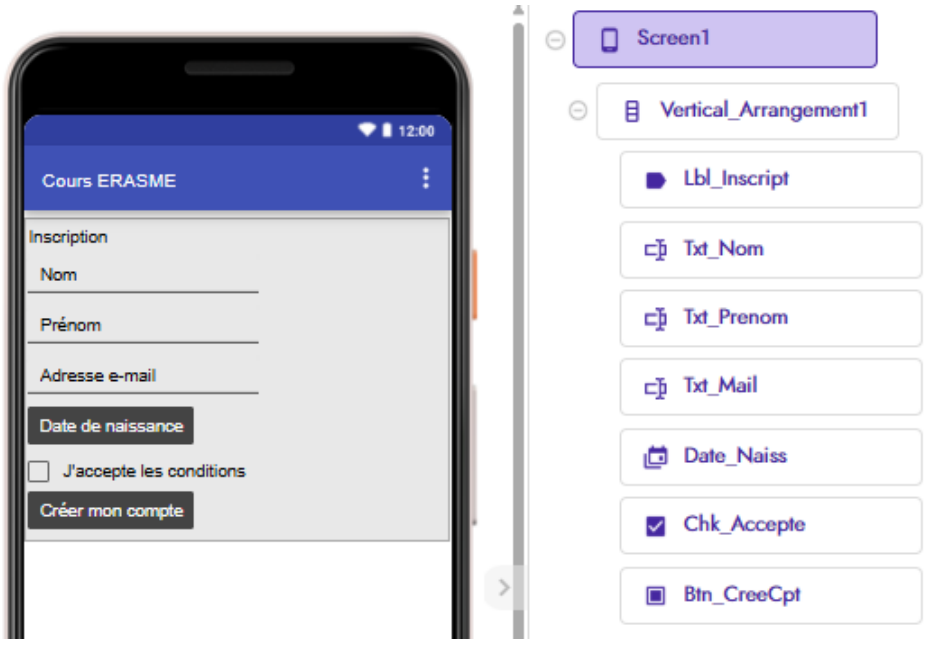
Voici quelques exemples.

Besoin	**Composant**
Activer une option	**Switch**
Afficher un titre	**Label**
Afficher une image	**Image**
Afficher une liste	**List View**
Choisir Oui/Non	**Check Box**
Choisir une date	**Date Picker**
Choisir une heure	**Time Picker**
Faire un choix dans une liste	**Spinner**
Lancer une action	**Button**
Régler une valeur	**Slider**
Saisir un texte	**Text Box**

P. ATELIER PRATIQUE

Créer un formulaire d'inscription en se basant sur l'image ci-après :

- Vertical Arrangement
- Modifier le libellé de l'écran
- Renommer les composants à l'écran
- Faire attention au type du composant *Adresse e-mail*
- La date de naissance est un composant "date"
- *Créer mon compte* est un composant "bouton"



Q. EXERCICE

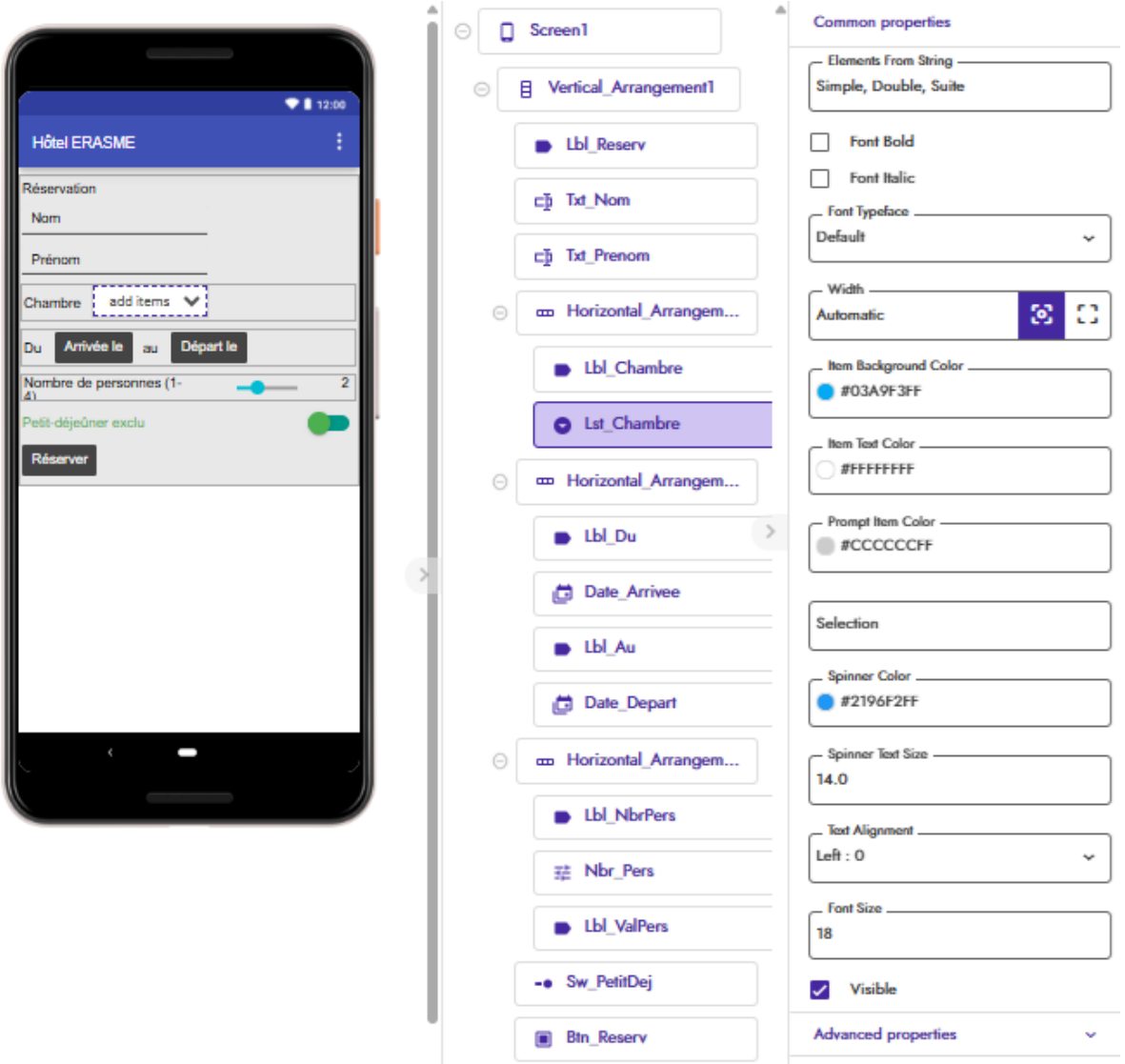
Conçois une interface de réservation d'hôtel comprenant notamment :

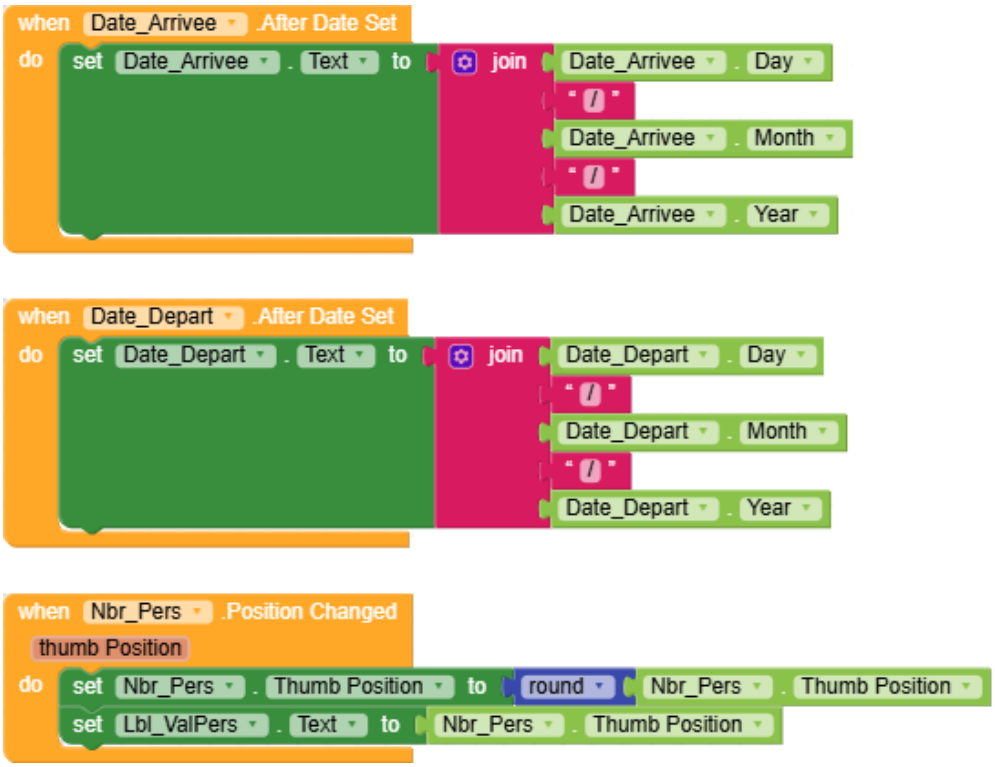
- Un titre « Réservation » ;
- Les champs **Nom** et **Prénom** ;
- Un **Spinner** pour choisir le type de chambre (Simple, Double, Suite) ;
- Deux **Date Picker** pour la date d'arrivée et celle de départ ;
- Un **Slider** pour indiquer le nombre de personnes (1 à 4) ;
- Un **Switch** intitulé « Petit-déjeuner inclus » ;
- Un bouton **Réserver**.

Remarques :

- Il faut penser à ajouter des Labels à certaines zones afin que l'utilisateur sache à quoi elles correspondent.
- Il faut définir les commandes Blocks pour les dates et le Slider.
- Sélectionner le composant le plus adapté à chaque type d'information ;
- Organiser les composants dans un **Vertical Arrangement** ;
- Appliquer les bonnes pratiques de nommage et de présentation.

Réponse :





4. LES COMPOSANTS INVISIBLES : LES VERITABLES MOTEURS DE L'APPLICATION

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre ce qu'est un composant invisible ;
- Connaître les principaux composants invisibles de Kodular ;
- Choisir le composant adapté à chaque besoin ;
- Configurer leurs propriétés essentielles ;
- Réaliser des applications plus interactives et plus complètes.

B. INTRODUCTION

Depuis le début de ce cours, nous avons utilisé des composants visibles tels que Label, Button, Text Box, etc. L'utilisateur les voit à l'écran. Mais une application moderne fait beaucoup plus que simplement afficher des boutons.

Elle peut :

- Enregistrer des données ;
- Jouer de la musique ;
- Prendre une photo ;
- Vibrer ;
- Accéder à Internet ;
- Utiliser le GPS ;
- Envoyer une notification.

Toutes ces fonctions sont assurées par les composants invisibles. Ils n'apparaissent pas dans l'interface utilisateur mais sont indispensables au fonctionnement de l'application.

Ajoute un composant (**Sensors**/)**Clock** dans le Designer. Tu remarqueras qu'il n'apparaît pas sur l'écran du téléphone. À la place, il est affiché sous le Viewer, dans une zone réservée aux composants invisibles. Ils fonctionnent en arrière-plan. Ils sont comparables au moteur d'une voiture. Le conducteur ne le voit pas, mais sans lui, la voiture n'avance pas.

Ils sont souvent activés grâce aux Blocks.

C. LE COMPOSANT NOTIFIER

Le **User Interface** / **Notifier** permet d'afficher des messages. C'est l'un des composants les plus utilisés.

Utilisations :

- Confirmer une action ;
- Afficher une erreur ;
- Demander une confirmation ;
- Informer l'utilisateur.

Exemple pratique :
Lorsque l'utilisateur clique sur **Button1**, afficher *Ceci est une simple notification.* Cette simple confirmation améliore fortement l'expérience utilisateur.



D. LE COMPOSANT CLOCK

Le composant **Sensors** / **Clock** gère le temps.

Il permet :

- D'obtenir la date actuelle ;
- De connaître l'heure ;
- De déclencher une action après un délai ;
- De répéter automatiquement une tâche.

Exemple :
Créer un chronomètre (voir suite du cours car on devra utiliser des variables).

E. LE COMPOSANT TINYDB

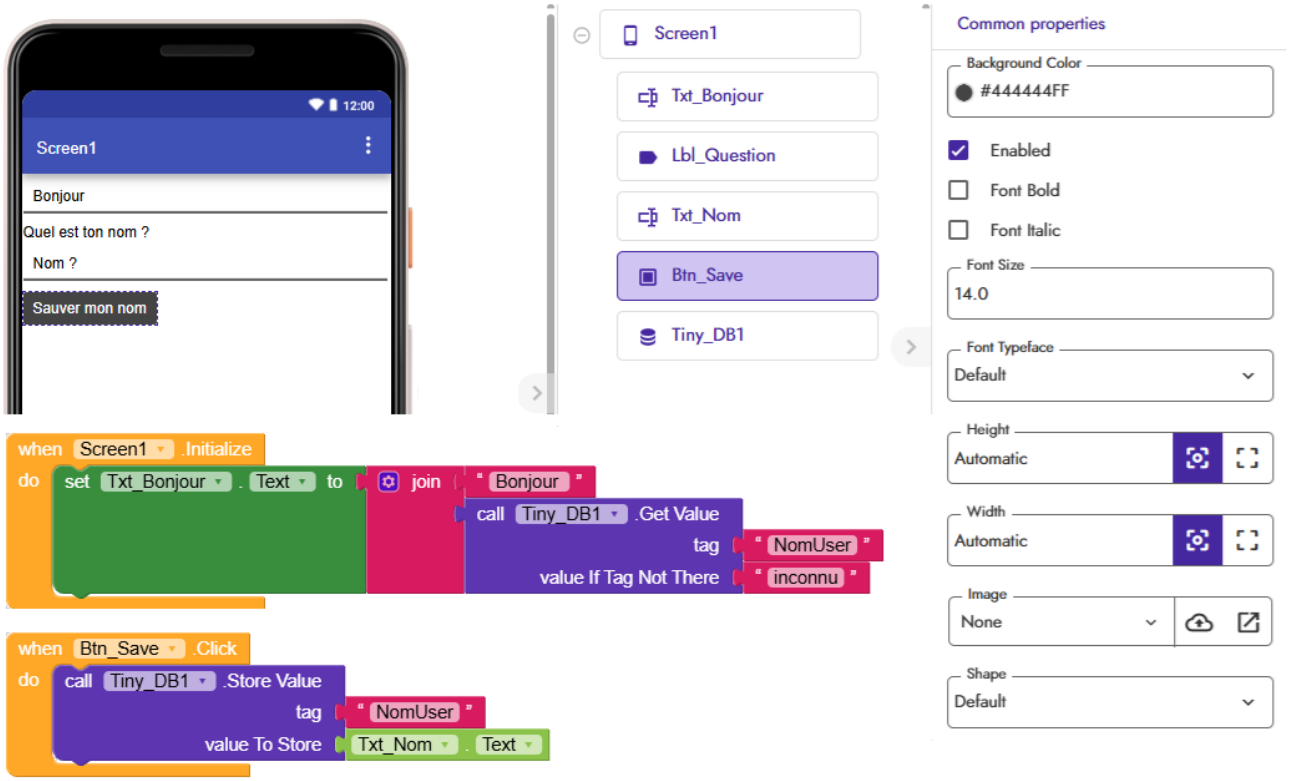
Storage / **TinyDB** est l'un des composants les plus importants de Kodular. Il permet d'enregistrer des informations directement dans le téléphone. Même si l'application est fermée, les données restent disponibles.

Exemples :

- Nom de l'utilisateur ;
- Score d'un jeu ;

- Paramètres ;
- Liste de tâches.

Dans l'exemple suivant, l'écran affiche la première fois *Bonjour inconnu*. Mais si l'utilisateur tape son nom et le sauve, l'application affichera *Bonjour nom* lors des utilisations ultérieures.



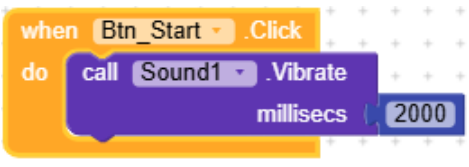
F. LE COMPOSANT SOUND

Le composant **Media** / **Sound** joue de courts effets sonores.

Exemples :

- Clic ;
- Bip ;
- Applaudissements ;
- Notification.

Dans les dernières versions Kodular, c'est également ce composant qui permet de faire vibrer l'appareil (smartphone/tablette) pendant un certain nombre de millisecondes.



G. LE COMPOSANT PLAYER

Le **Media** / **Player** lit des fichiers audio plus longs. Contrairement au composant **Sound**, il est prévu pour des lectures prolongées.



Exemples :

- Musique ;

- Podcast ;
- Livre audio ;
- Ambiance sonore.

H. LE COMPOSANT CAMERA

La **Media** / **Camera** permet :

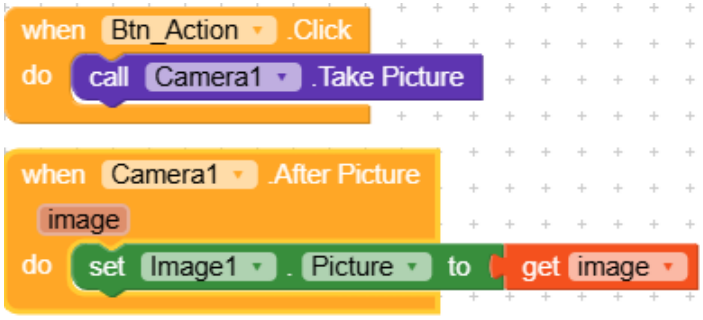
- De prendre une photo ;
- D'enregistrer une image.

Applications possibles :

- Carnet de recettes ;
- Inventaire ;
- Application immobilière ;
- Carnet de voyage.

Exemple :
L'utilisateur clique :

- Prendre une photo
- L'appareil photo s'ouvre.
- La photo est enregistrée.
- Elle peut ensuite être affichée dans un composant Image.



Remarque :
Le block **get(image)** a été ajouté en glissant le bouton **image** du block **When Camera1.After Picture.**

I. LE COMPOSANT IMAGE PICKER

L'Image **Media** / **Image Picker** permet de sélectionner une photo déjà présente dans la galerie du téléphone.

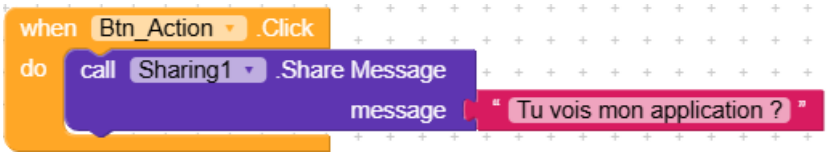


J. LE COMPOSANT SHARING

Le composant **Social** / **Sharing** partage du contenu avec d'autres applications.

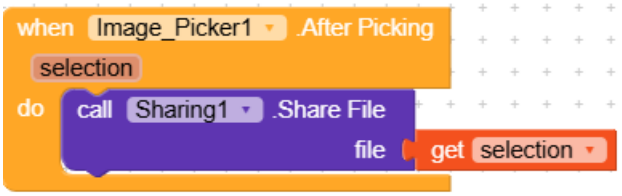
Exemples :

- Envoyer un texte ;
- Partager une photo ;
- Transmettre un lien.



Applications :

- WhatsApp ;
- e-mail ;
- SMS ;
- réseaux sociaux.



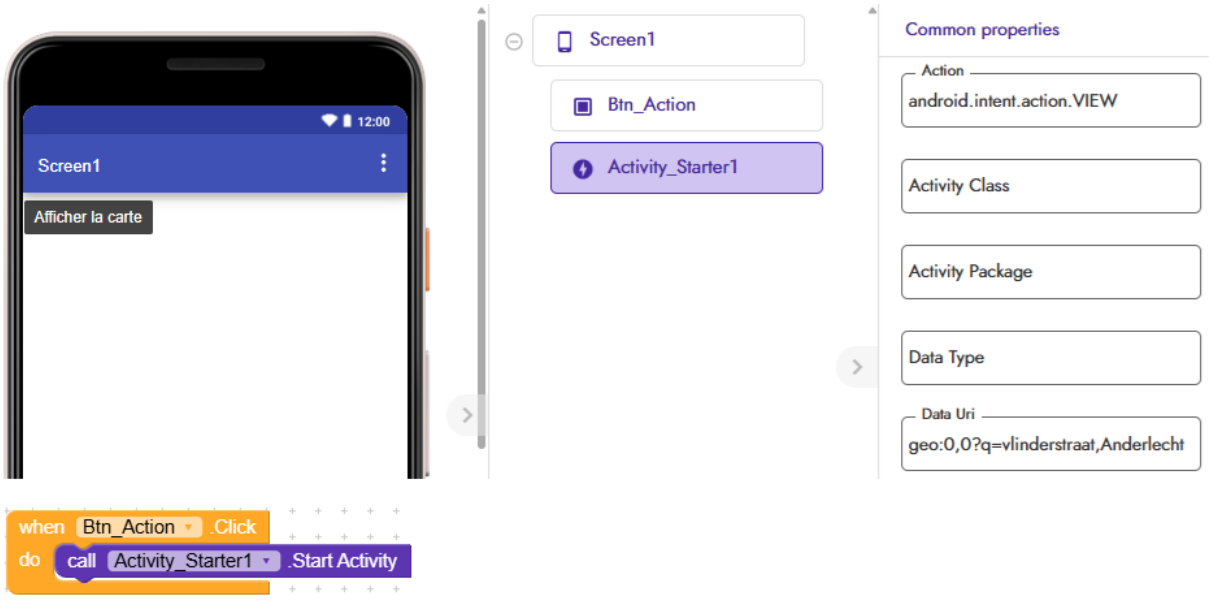
K. LE COMPOSANT ACTIVITY STARTER

Le composant **Connectivity** / **Activity Starter** lance une autre application installée sur le téléphone.

Exemples :

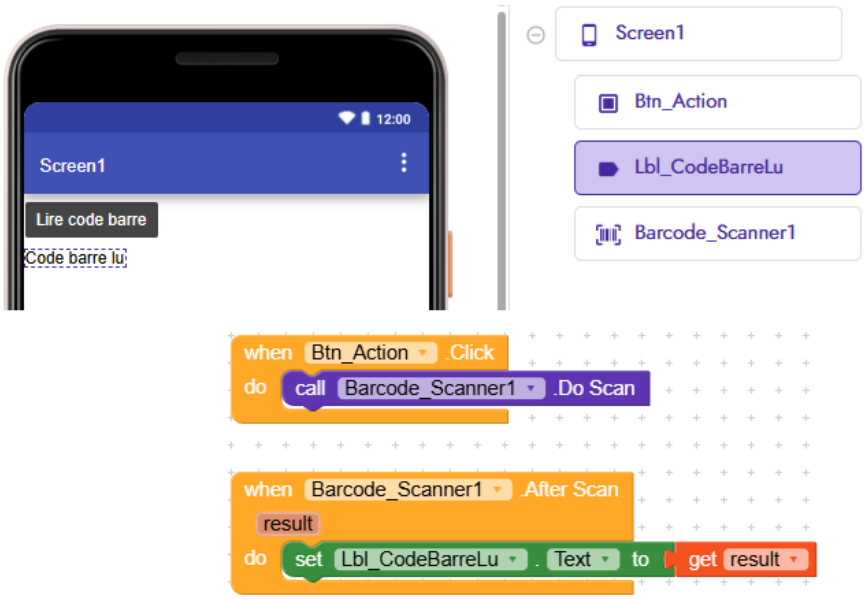
- Lancer Google Maps ;
- Afficher les contacts ;
- Ouvrir le navigateur.

Exemple :
Affichage de la rue Vlinderstraat à Anderlecht sur une carte



L. LE COMPOSANT BARCODE SCANNER

Le composant **Sensors** / **Barcode Scanner** permet de lire les codes-barres et les QR-Codes. Il peut donc servir dans le cadre d'inventaire, de gestion de stock, …



M. LE COMPOSANT LOCATION SENSOR

Le composant **Sensors** / **Location Sensor** permet de récupérer la position géographique.

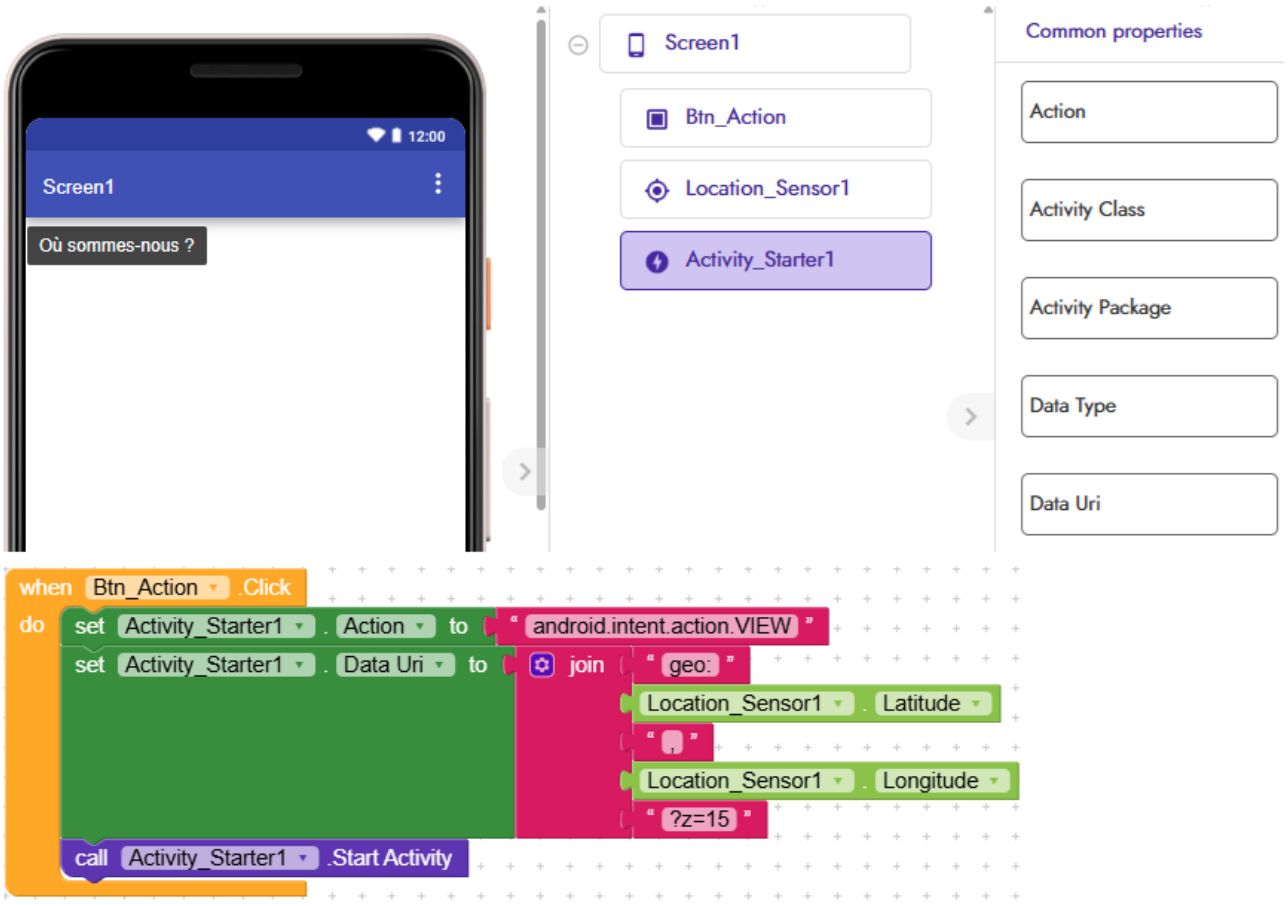
Applications :

- Randonnée ;
- Livraison ;
- Météo locale ;
- Géolocalisation.

Informations disponibles :

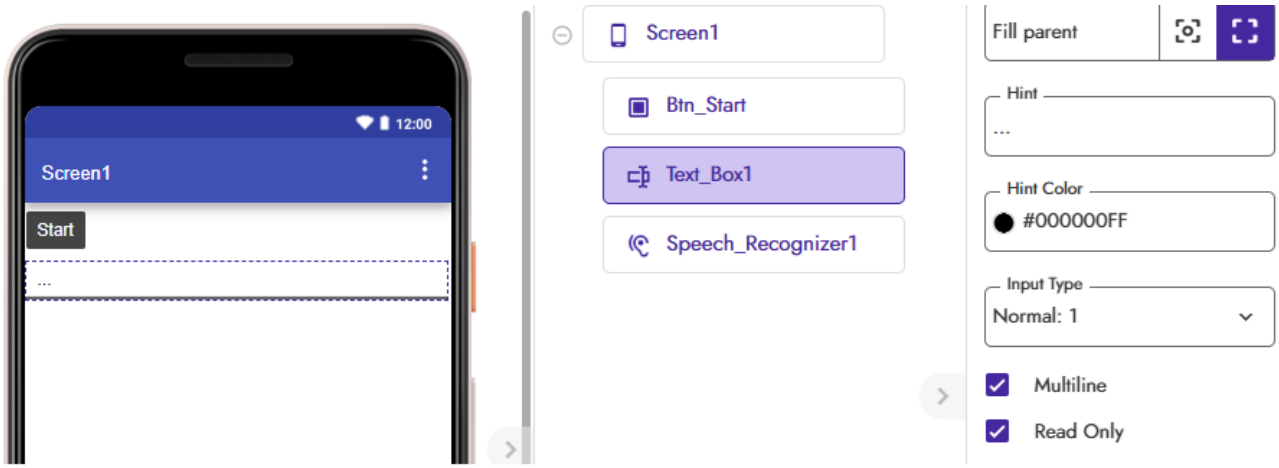
- Latitude ;
- Longitude ;
- Altitude ;
- Vitesse.

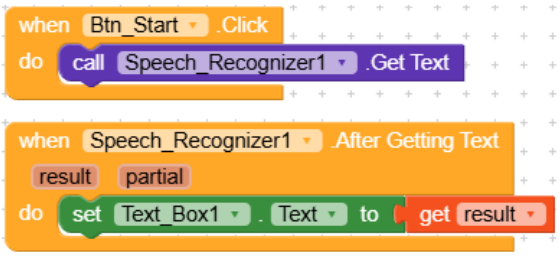
Exemple :
Demander à l'application de nous afficher la carte correspondant à notre position actuelle lorsqu'on clique sur le bouton.



N. LE COMPOSANT SPEECH RECOGNIZER

Le composant **Media** / **Speech Recognizer** permet de transformer un texte dicté par l'utilisateur en un texte écrit.





O. LE COMPOSANT TEXT TO SPEECH

Le composant **Media** / **Text To Speech** est la fonction inverse. Le système dit ce qui est écrit.

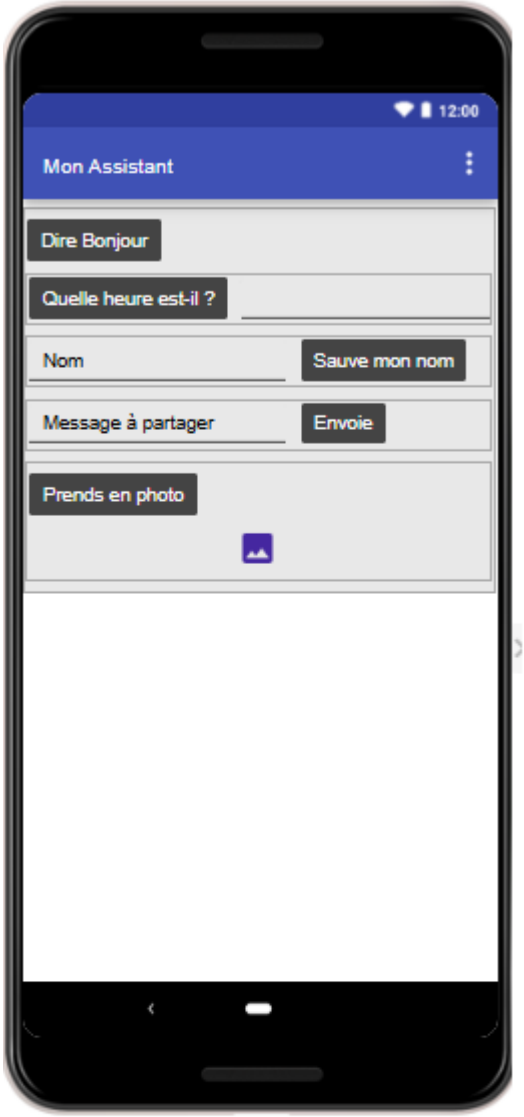


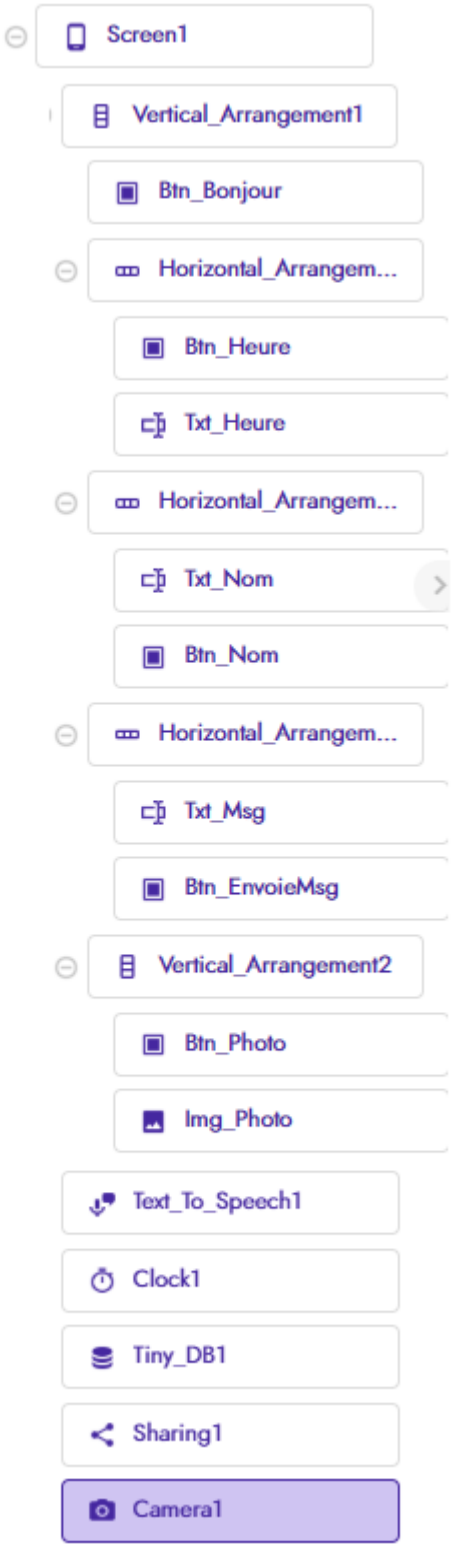
P. EXERCICE : MON ASSISTANT PERSONNEL

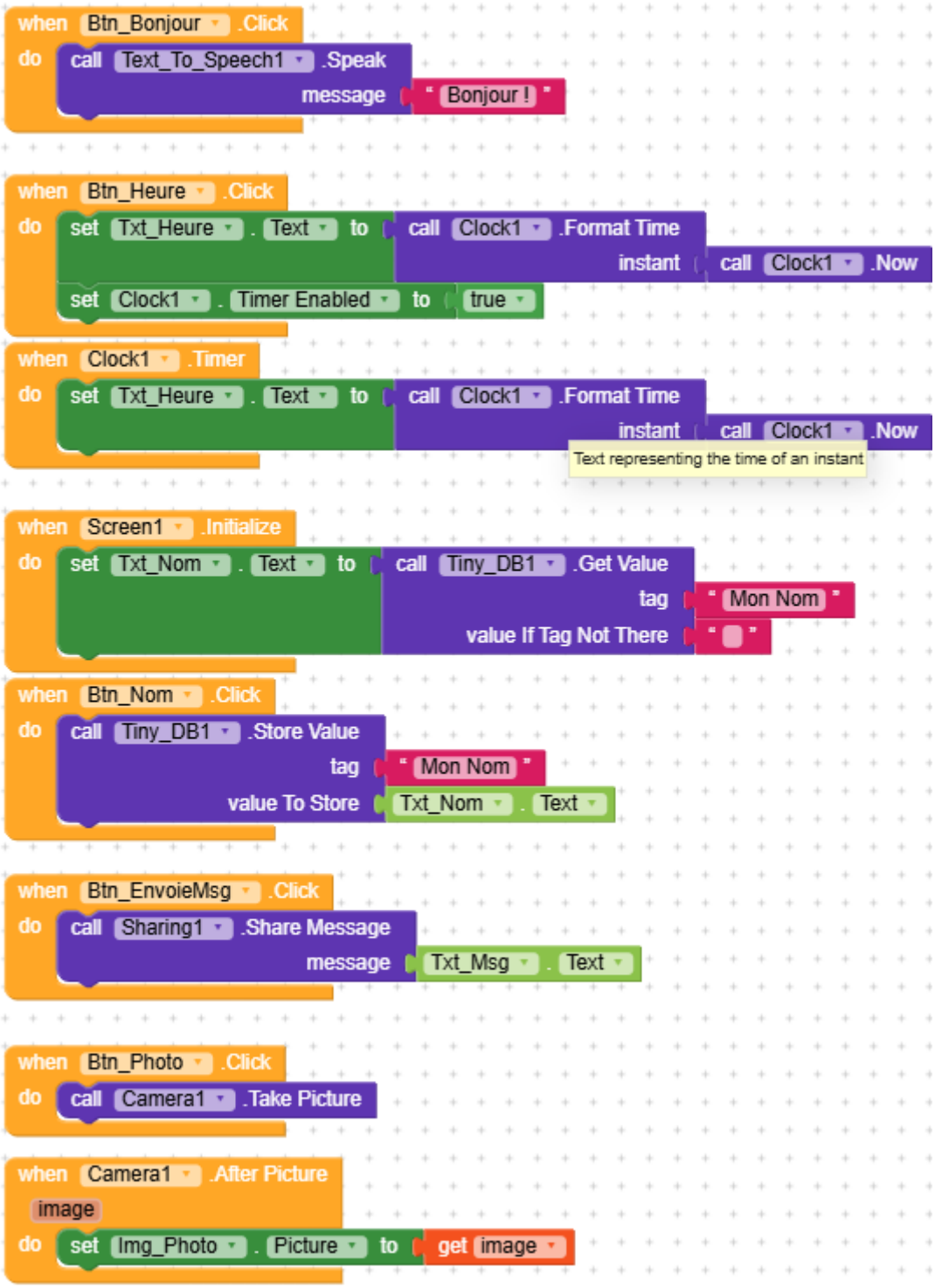
Conçois l'interface d'une application nommée **Mon Assistant** avec les fonctionnalités suivantes :

- Un bouton *Dire bonjour* (**Text To Speech**) ;
- Un bouton *Afficher l'heure* (**Clock**) ;
  Une fois que l'heure est affichée, il faut la réactualiser en permanence ;
- Un bouton *Enregistrer mon nom* (**TinyDB**) ;
- Un bouton *Partager un message* (**Sharing**) ;
- Un bouton *Prendre une photo* (**Camera**).

Réponse :


12:00
Mon Assistant
Dire Bonjour
Quelle heure est-il ?
Nom
Sauve mon nom
Message à partager
Envoie
Prends en photo


Screen1
Vertical_Arrangement1
Btn_Bonjour
Horizontal_Arrangem...
Btn_Heure
Txt_Heure
Horizontal_Arrangem...
Txt_Nom
Btn_Nom
Horizontal_Arrangem...
Txt_Msg
Btn_EnvoieMsg
Vertical_Arrangement2
Btn_Photo
Img_Photo
Text_To_Speech1
Clock1
Tiny_DB1
Sharing1
Camera1



5. PROJET COMPLET, BONNES PRATIQUES

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Concevoir une interface complète et professionnelle ;
- Choisir les composants adaptés à chaque besoin ;
- Organiser correctement les éléments d'un écran ;

- Appliquer les bonnes pratiques de conception utilisées par les développeurs ;
- Préparer ton application avant de commencer sa programmation.

B. INTRODUCTION

Au cours des chapitres précédents, tu as découvert tous les éléments qui permettent de construire l'interface d'une application.

Tu sais maintenant :

- Utiliser le Designer ;
- Créer des mises en page avec les Arrangements ;
- Choisir les composants visibles ;
- Utiliser les composants invisibles.

Dans ce chapitre, nous allons réunir toutes ces connaissances dans un projet complet. L'objectif n'étant toutefois pas encore de réaliser une application complexe. Nous allons surtout apprendre à concevoir une interface propre, logique et agréable à utiliser. C'est cette méthode qui distingue un développeur amateur d'un développeur professionnel.

C. AVANT DE PROGRAMMER

Beaucoup de débutants commettent toujours la même erreur. Ils ouvrent Kodular et commencent immédiatement à programmer.

Résultat :

- Composants mal placés ;
- Noms incompréhensibles ;
- Ecran désorganisé ;
- Programmation difficile.

Les développeurs expérimentés suivent une méthode bien différente. Ils consacrent souvent plusieurs heures à préparer l'interface avant d'écrire le moindre bloc. Pourquoi ? Parce qu'une interface bien pensée facilite énormément la programmation.

Notre projet

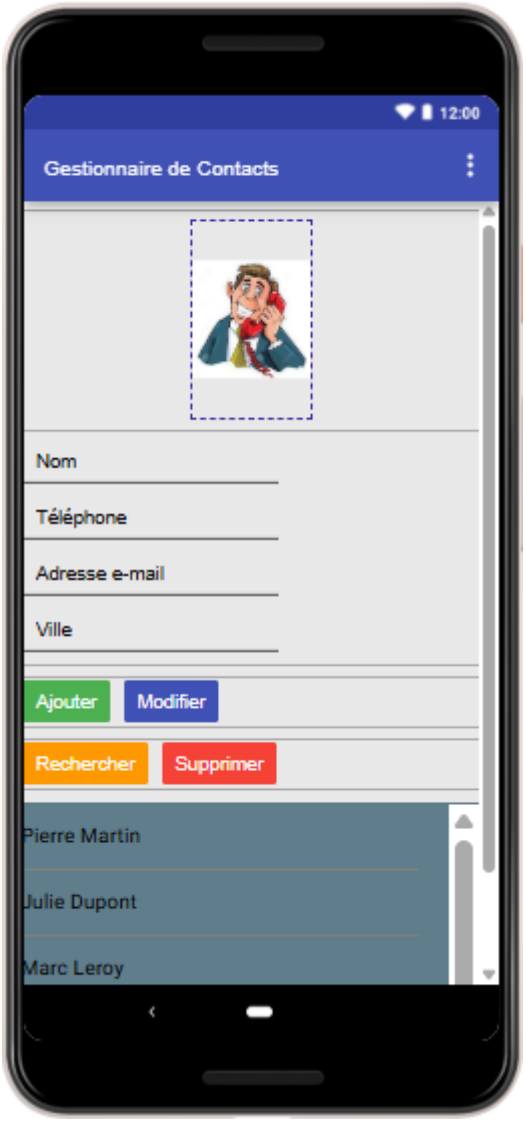
Nous allons créer une application appelée *Mon Gestionnaire de Contacts*

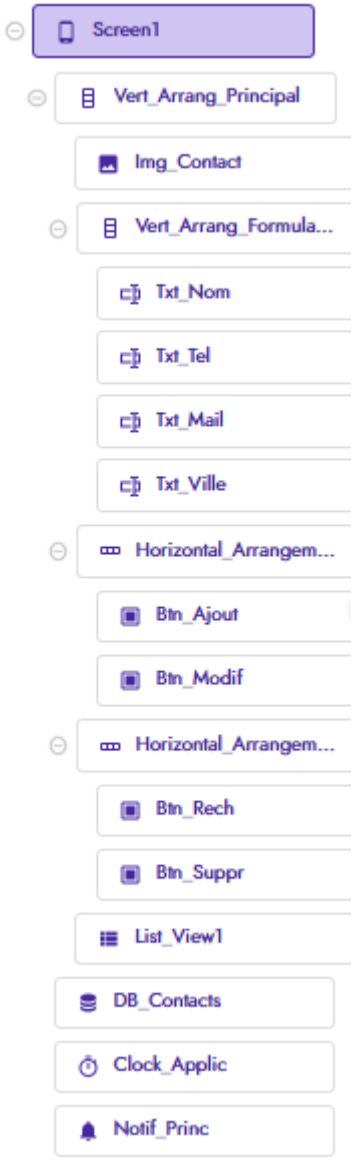
Elle permettra ultérieurement :

- D'ajouter un contact ;
- De rechercher un contact ;
- De modifier un contact ;
- De supprimer un contact.

Dans cette section, nous allons uniquement réaliser son interface.

Aperçu du résultat





D. DEFINITION DE L'INTERFACE

Commence par configurer les propriétés suivantes pour le Screen :

Propriété	**Valeur**
Title	Gestionnaire de Contacts
Align Vertical	Top
Scrollable	True
Background Color	Blanc

Une première règle étant de ne jamais déposer directement les composants sur Screen1, crée d'abord le **Vertical Arrangement** principal avec la largeur **Fill Parent** et la hauteur **Automatic**.

Ajoute ensuite les composants visibles :

- L'image du contact (photo)
  Largeur et **Hauteur** = 25%
- Dans un arrangement vertical, ajoute les zones : *Nom*, *téléphone*, *adresse e-mail*, *ville*
- Dans deux arrangements horizontaux, crée les boutons *Ajouter* (en vert), *Modifier* (en bleu), *Rechercher* (en orange) et *Supprimer* (en rouge). L'utilisateur identifie ainsi immédiatement la fonction de chaque bouton.
- Ajoute une **List View** avec les valeurs suivantes : Pierre Martin, Julie Dupont, Marc Leroy et Nadia Simon.
  Plus tard, ces données proviendront de **TinyDB**.
- Ajoute les composants invisibles dont on aura besoin plus tard : **TinyDB**, **Notifier** et **Clock**.
  Nous les utiliserons dans le chapitre consacré aux blocs.

E. PENSER COMME UN UTILISATEUR

Avant de commencer la programmation, pose-toi toujours quelques questions :

- L'utilisateur comprend-il immédiatement où commencer ?
- Les boutons sont-ils suffisamment grands ?
- Le texte est-il lisible ?
- Les couleurs sont-elles cohérentes ?
- Les éléments importants attirent-ils le regard ?

Une bonne interface répond naturellement à ces questions.

F. VERIFIER L'ERGONOMIE

Voici une petite liste de contrôle utilisée avant la publication d'une application.

Vérification	**Oui**
Les composants sont alignés ?	☐
Les boutons sont lisibles ?	☐
Les couleurs sont cohérentes ?	☐
Les espaces sont suffisants ?	☐
Les composants sont renommés ?	☐
Les images sont optimisées ?	☐
Les textes sont sans faute ?	☐

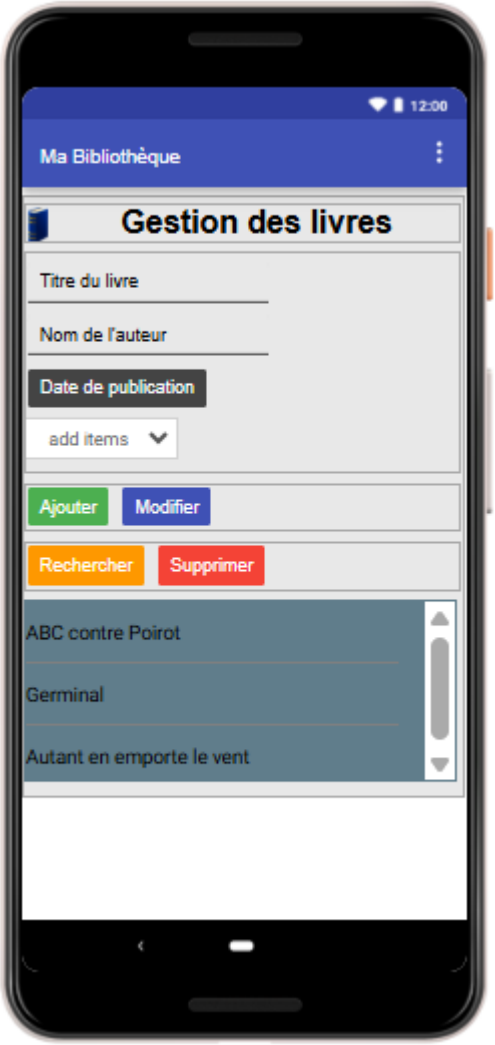
Prends l'habitude de vérifier ces points avant chaque mise en ligne.

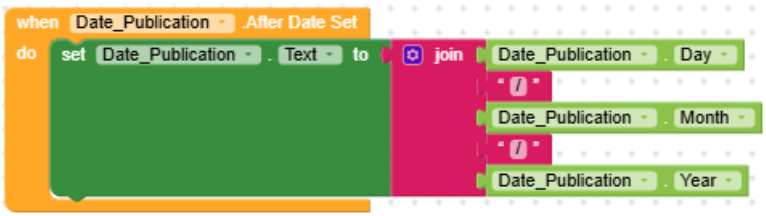
G. EXERCICE : PROJET *MA BIBLIOTHEQUE*

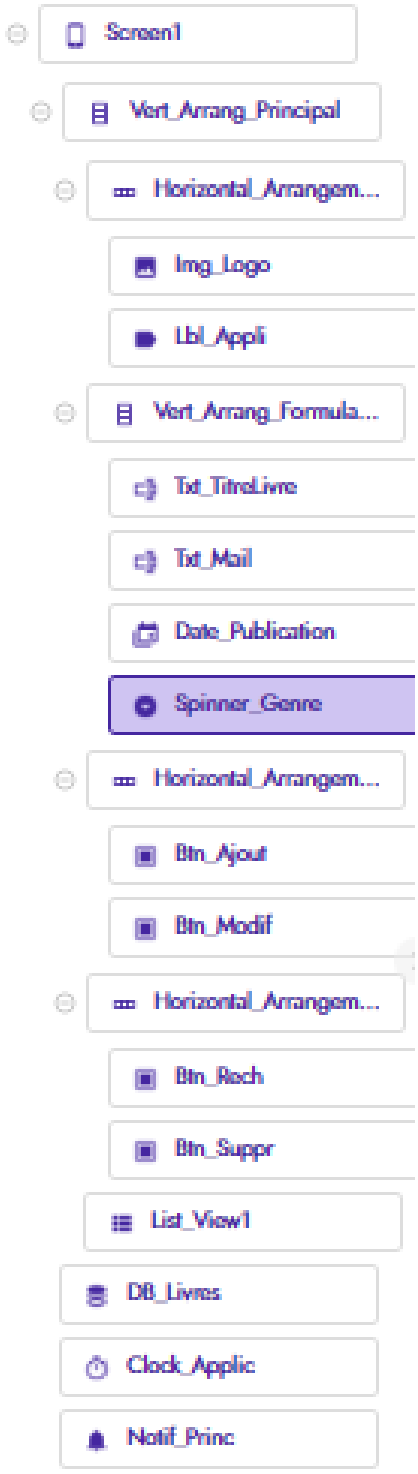
Créer une application intitulée *Ma Bibliothèque* qui servira de base aux exercices des prochains chapitres.

L'interface doit comporter :

- Zone supérieure :
  - Logo
  - Titre du programme
- Zone centrale :
  - Titre du livre
  - Nom de l'auteur
  - Date de publication (avec l'affichage de la date)
  - Genre (**Spinner**)
- Zone boutons
  - Ajouter
  - Modifier
  - Rechercher
  - Supprimer
- Zone inférieure
  - Une **List View** contenant quelques livres.
- Composants invisibles
  - **TinyDB**
  - **Notifier**
  - **Clock**






Screen1
Vert_Arrang_Principal
Horizontal_Arrangem...
Img_Logo
Lbl_Appli
Vert_Arrang_Formula...
Txt_TitreLivre
Txt_Mail
Date_Publication
Spinner_Genre
Horizontal_Arrangem...
Btn_Ajout
Btn_Modif
Horizontal_Arrangem...
Btn_Rech
Btn_Suppr
List_View1
DB_Livres
Clock_Applic
Notif_Princ

Programmer avec les blocs

1. COMPRENDRE LA PROGRAMMATION EVENEMENTIELLE

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre le principe de la programmation événementielle ;
- Distinguer un événement d'une action ;
- Reconnaître les différents types d'événements dans Kodular ;
- Créer des interactions simples entre les composants ;
- Adopter une méthode de programmation claire et efficace.

B. INTRODUCTION

Jusqu'à présent, tu as appris à créer de belles interfaces. Mais une belle interface ne suffit pas. Imagine une voiture flambant neuve. Elle possède un magnifique tableau de bord, des sièges confortables, une belle peinture, de belles jantes. Pourtant… Sans le moteur, impossible de rouler.

Une application fonctionne exactement de la même manière. Le **Designer** construit la carrosserie. Les **Blocks** créent le moteur. C'est grâce aux blocks que ton application devient interactive.

C. QU'EST-CE QUE LA PROGRAMMATION EVENEMENTIELLE ?

Contrairement aux programmes classiques qui exécutent une suite d'instructions du début à la fin, une application Android reste en attente. Elle attend que quelque chose se produise.

Par exemple :

- L'utilisateur appuie sur un bouton ;
- Il saisit un texte ;
- Il secoue le téléphone ;
- Il change l'orientation de l'écran ;
- Il sélectionne un élément dans une liste.

Ces actions sont appelées **événements**. Chaque événement déclenche une ou plusieurs actions. C'est ce que l'on appelle la **programmation événementielle**.

Une application Android ne travaille presque jamais en continu. Elle attend des événements et réagit lorsqu'ils se produisent.

D. LE CYCLE « ÉVENEMENT → TRAITEMENT → RESULTAT »

Toutes les applications suivent le même principe.

Evénement -> Traitement -> Résultat

Prenons un exemple :
L'utilisateur clique sur un bouton -> Kodular le détecte -> Le programme change le texte affiché -> Le nouvel affichage apparaît immédiatement.

Cette logique est présente dans toutes les applications Android.

E. LES PRINCIPAUX EVENEMENTS

Chaque composant possède ses propres événements.

Voici les plus utilisés :

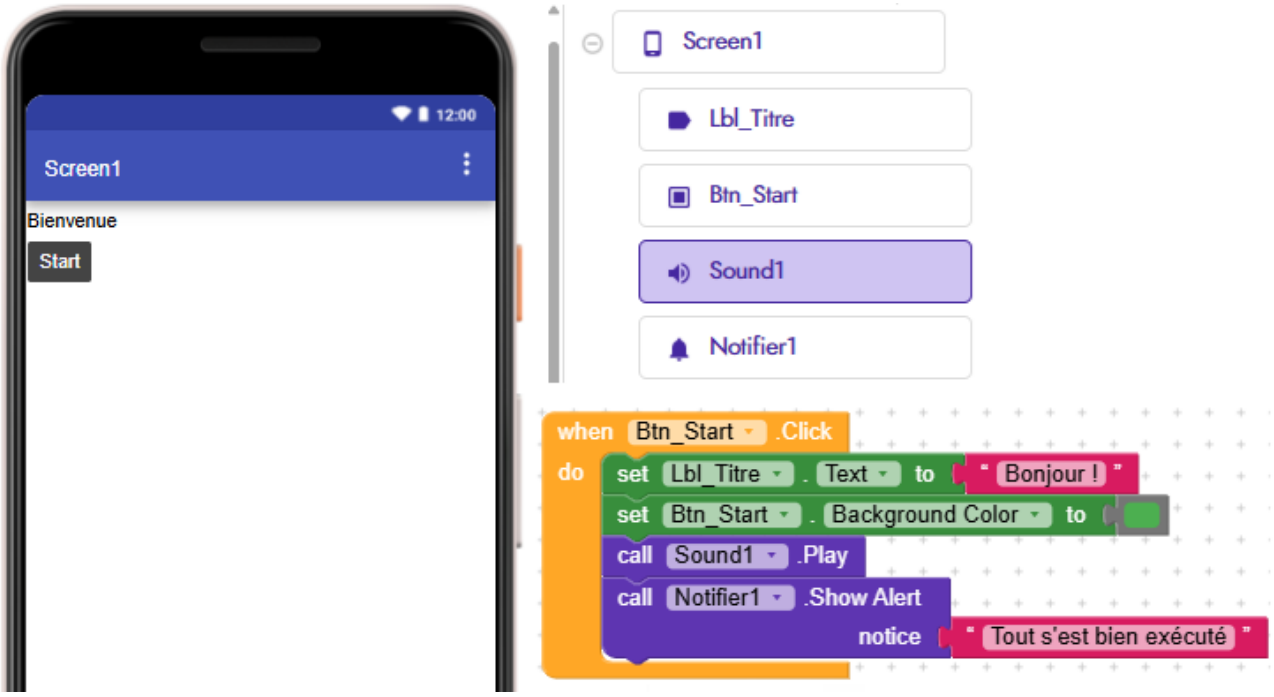
- **Button**
  - **when btnCalcul.Click**
    Déclenché lorsque l'utilisateur appuie sur le bouton.
- **Text Box**
  - **when txtNom.GotFocus**
    Le champ reçoit le curseur.
  - **when txtNom.LostFocus**
    L'utilisateur quitte le champ.
  - **when txtNom.TextChanged**
    Le contenu est modifié. Permet de vérifier une saisie en temps réel.
- **Check Box**
  - **when chkConditions.Changed**
    Déclenché lorsque la case est cochée ou décochée.
- **Switch**
  - **when swModeSombre.Changed**
    Déclenché car la valeur change. Très utilisé dans les écrans de paramètres.
- **Slider**
  - **when sldVolume.PositionChanged**
    Exécuté lorsque l'utilisateur déplace le curseur.
- **List View**
  - **when lstContacts.AfterPicking**
    Déclenché après la sélection d'un élément.
- **Spinner**
  - **when spPays.AfterSelecting**
    L'utilisateur choisit un élément dans la liste.
- **Clock**
  - **when clockApplication.Timer**
    Déclenché automatiquement selon l'intervalle défini.

F. UNE ACTION PEUT DECLENCHER PLUSIEURS ACTIONS

Dans Kodular, un seul événement peut exécuter plusieurs instructions.

Par exemple, lorsque l'utilisateur clique sur un bouton, cela peut déclencher simultanément une sauvegarde des données, un son, l'affichage d'une notification et le changement de couleur d'un bouton. Toutes ces actions sont exécutées dans le même événement.

Prenons l'exemple suivant :



Lorsque l'utilisateur clique sur le bouton :

- Le titre devient *Bonjour !* ;
- Le bouton devient vert ;
- Un son est joué ;
- Un message apparaît.

Un seul clic. Quatre actions différentes.

Kodular exécute les blocs **de haut en bas**. L'ordre des opérations peut être important dans certains cas. Il faut donc y faire attention.

G. LES EVENEMENTS AUTOMATIQUES

Tous les événements ne sont pas déclenchés par l'utilisateur. Certains proviennent directement du téléphone.

Exemples :

- Le GPS change de position ;
- Une connexion Bluetooth est établie ;
- Un chronomètre atteint une seconde ;
- Une réponse Internet est reçue.

Ton application peut donc réagir sans intervention de l'utilisateur.

H. LES ERREURS FREQUENTES

- **Programmer en dehors d'un événement**
  Un bloc isolé ne sera jamais exécuté. Il doit toujours être relié à un événement.

I. EXERCICE

Crée une application de bienvenue contenant un Label, un bouton *Commencer* et un Notifier. Le Label contient le texte *Bienvenue dans Kodular*

Lorsque l'utilisateur clique sur le bouton, le Label devient *Salut !* ; puis le Notifier affiche *Merci d'utiliser mon application*.

J. EXERCICE

Crée deux boutons gris. Le premier affiche le libellé *Rouge*, le second *Bleu*. Lorsque l'utilisateur clique sur un bouton, celui-ci change de couleur (il devient rouge ou bleu en fonction du bouton cliqué).

Tu viens de créer deux événements totalement indépendants.

K. LES EVENEMENTS LES PLUS UTILISES

Composant	**Evénement principal**
Button	**Click**
Text Box	**TextChanged**
Check Box	**Changed**
Switch	**Changed**
Slider	**PositionChanged**
Spinner	**AfterSelecting**
List View	**AfterPicking**
Clock	**Timer**
Camera	**AfterPicture**
Web	**GotText**
Bluetooth	**Connected**

Ce tableau mérite d'être mémorisé. Tu les utiliseras dans presque tous tes projets.

2. LES VARIABLES : MEMORISER ET MANIPULER DES DONNEES

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre ce qu'est une variable ;
- Créer des variables globales et locales dans Kodular ;
- Stocker différents types de données ;
- Modifier le contenu d'une variable ;
- Utiliser les variables dans des applications concrètes.

B. QU'EST-CE QU'UNE VARIABLE ?

Lorsque l'on programme, on doit souvent demander à l'ordinateur de retenir certaines informations pendant le traitement. Ces informations seront stockées par l'ordinateur dans des zones de sa mémoire que l'on peut comparer aux tiroirs d'une armoire. On devra simplement placer une étiquette sur chaque tiroir

utilisé afin de pouvoir retrouver l'information souhaitée. On pourra alors par la suite facilement relire cette information, mais aussi la modifier, voire la supprimer.

En informatique, un tiroir est appelé une variable et l'étiquette, son nom. Sans variables, une application serait incapable de retenir la moindre information.

Ces informations, ou données, sont de différents types : nombres, caractères, dates, etc.

C. LES DIFFERENTS TYPES DE DONNEES

Une variable peut contenir différents types d'informations :

- Le **texte**
  On parle de chaîne de caractères (ou **String**).
- Les **nombres**
  Ils servent aux calculs.
- Les **valeurs logiques**
  Il n'existe que deux possibilités : *Vrai* ou *Faux*.
  Ces valeurs sont appelées **booléens**. Elles sont très utilisées avec les conditions.
- Les **listes**
  Une variable peut également contenir plusieurs éléments.
  Nous étudierons les listes plus loin dans ce chapitre.

D. LES VARIABLES GLOBALES

Une variable globale est accessible depuis toute l'application. Elle peut être utilisée dans plusieurs événements.

Créons une variable globale *score* que l'on initialise à 0 et incrémente de 10 points à chaque clic sur le bouton *Btn_Bonus*.

On aura donc dans les Blocks :



Après 3 clics, la variable vaudra 30 car elle garde sa valeur tant que l'application reste ouverte.

E. LES VARIABLES LOCALES

Les variables locales existent uniquement à l'intérieur d'un bloc précis. Lorsque l'événement est terminé, la variable disparaît. Ces variables sont utilisées pour des calculs temporaires.

F. VARIABLES ET COMPOSANTS

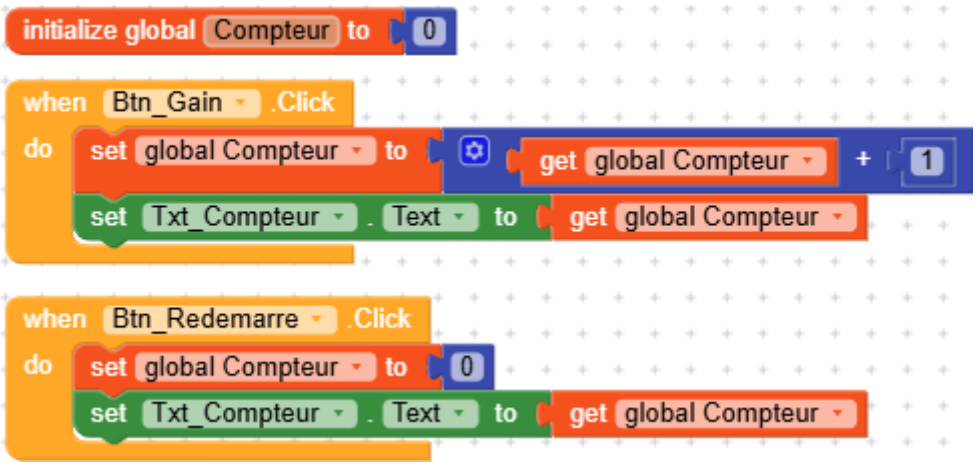
Attention, les variables ne remplacent pas les composants. Elles travaillent avec eux. La variable sert à mémoriser. Le Label ou la Text Box servent à afficher. Chaque élément possède un rôle bien précis.

G. LES ERREURS FREQUENTES

- **Oublier d'initialiser une variable**
  Une variable doit toujours recevoir une valeur initiale.
- **Utiliser un nom peu explicite**
- **Mélanger plusieurs informations au sein d'un même variable**
  Exemple : Une adresse est composée idéalement de plusieurs variables

H. EXERCICE : LE COMPTEUR

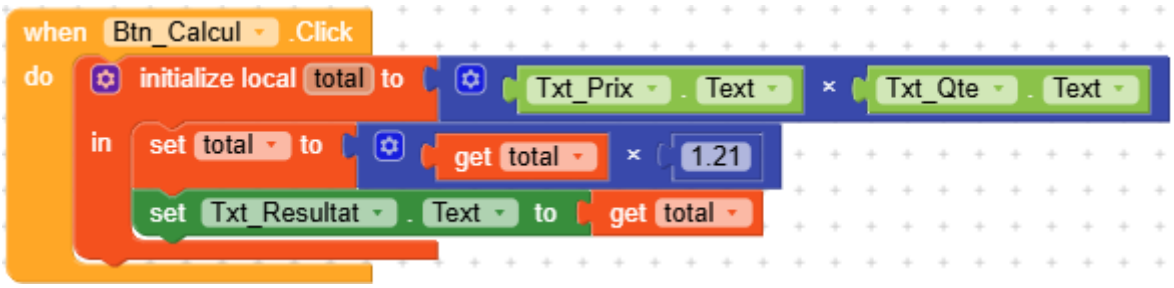
Crée une interface affichant un compteur initialisé à 0, incrémenté de 1 à chaque clic sur un bouton *Gain* et réinitialisé à 0 lors d'un clic sur le bouton *Redémarre*.



I. EXERCICE : CALCUL DU TOTAL

Crée 3 Text Boxes *Prix HTVA*, *Quantite* et *Resultat* ; ainsi qu'un bouton *Calculer*.

Lorsque l'utilisateur cliquera sur le bouton *Calculer*, l'application devra calculer le résultat (= Prix x Quantite et ajouter 21% de TVA) et l'afficher. Pour ce faire, il utilisera une variable locale.



3. LES CONDITIONS : APPRENDRE A PRENDRE DES DECISIONS

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre le principe des conditions ;
- Utiliser les blocs if, else if et else de Kodular ;
- Comparer des nombres, des textes et des valeurs logiques ;
- Créer des applications qui réagissent différemment selon les situations ;
- Eviter les erreurs les plus courantes lors de l'utilisation des conditions.

B. INTRODUCTION

Ce chapitre est dédié à l'apprentissage de tout ce qui concerne les outils de test. Ceux-ci sont fort utiles en programmation. Ils permettent de vérifier la validité des valeurs encodées par un utilisateur, de sélectionner le traitement à effectuer en fonction de certaines valeurs, etc.

En effet, avant d'enregistrer un formulaire, il est indispensable de vérifier les informations. Cette vérification améliore énormément la qualité d'une application.

Jusqu'à présent, nos applications exécutaient toujours les mêmes actions :

- Cliquer sur un bouton affichait toujours le même message
- Un compteur augmentait systématiquement de 1
- Une variable recevait toujours une nouvelle valeur.

Mais une véritable application doit savoir prendre des décisions.

Imagine un distributeur de billets. Si tu saisis un code PIN correct, il doit afficher le menu. Par contre, si le code est incorrect, il doit afficher un message d'erreur. Le distributeur ne réagit donc pas toujours de la même manière. Il analyse une situation puis choisit une action. C'est exactement le rôle des conditions.

C. QU'EST-CE QU'UNE CONDITION ?

Une condition est une question dont la réponse ne peut être que :

- Vrai
- Faux

Exemples :

- L'utilisateur est-il connecté ?
- Le score est-il supérieur à 100 ?
- La zone Nom est-elle vide ?
- Le mot de passe est-il correct ?

Toutes ces questions peuvent être résolues par une condition.

D. LE BLOC IF

Le bloc le plus simple est :

SI condition
ALORS faire quelque chose

En anglais :

IF condition
THEN action

Exemple :

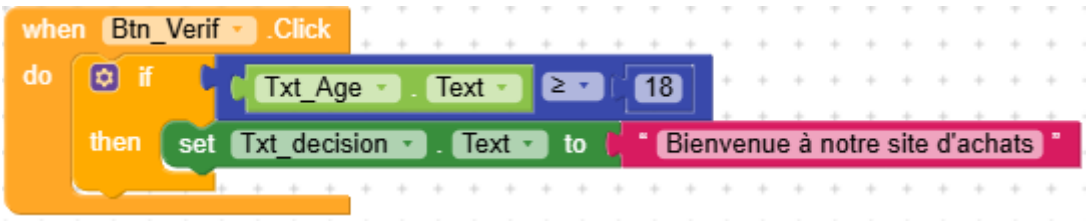
SI la zone nom est significative
ALORS activer le bouton *Envoyer*

Dans Kodular, cela signifie :

Si la condition est vraie, les blocs placés à l'intérieur seront exécutés.
Sinon, rien ne se passe.

E. EXEMPLE CONCRET

Créons l'application de vente en ligne demandant et vérifiant l'âge de l'utilisateur. S'il est majeur, l'accès sera autorisé.



Le programme vérifie et prend donc une décision.

F. LES OPERATEURS DE COMPARAISON

Les conditions utilisent des comparaisons.

Voici les plus importantes.

Opérateur	**Signification**
=	égal à
≠	différent de
>	supérieur à
<	inférieur à
≥	supérieur ou égal à
≤	inférieur ou égal à

G. LE BLOC IF...ELSE

Très souvent, on souhaite exécuter une action différente lorsque la condition est fausse.

Le bloc devient alors :

SI condition
ALORS faire action 1
SINON faire action 2

En anglais :

IF condition
THEN action 1
ELSE action 2

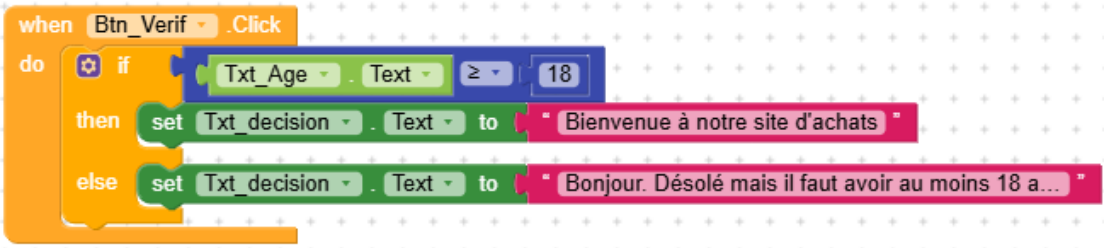
Exemple :

SI la zone nom est significative
ALORS activer le bouton *Envoyer*
SINON désactiver le bouton *Envoyer*

Dans Kodular, cela signifie :

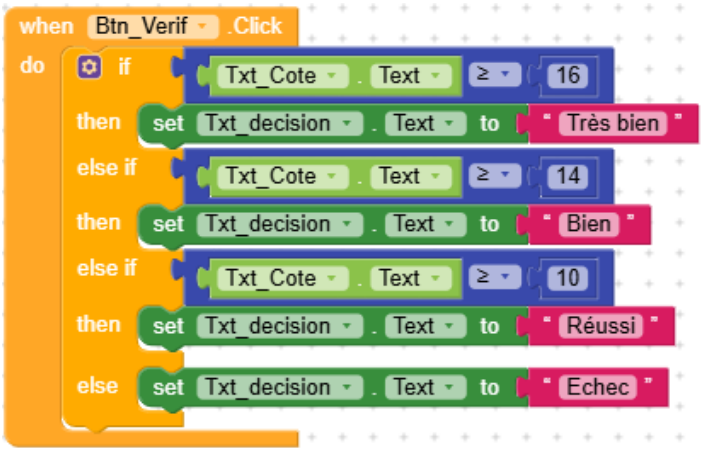
Si la condition est vraie, les blocs placés à l'intérieur de la section **THEN** sont exécutés ; sinon, ce sont ceux de la section **ELSE**.

Exemple :



Certaines situations nécessitent néanmoins plusieurs choix. La section **ELSE** se transforme dans ce cas en **ELSE IF** grâce à la roue dentée bleue du bloc **IF**.

Prenons l'exemple d'une application scolaire qui doit afficher une mention différente en fonction du résultat de l'élève. Si la note ≥ 16, il écrira *Très bien* ; sinon si la note ≥ 14, *Bien* ; sinon si note ≥ 10, *Réussi* ; sinon *Echec*.



Chaque condition est testée dans l'ordre. Dès qu'une condition est vraie, les suivantes ne sont plus vérifiées.

Remarques :
Les conditions ne servent pas uniquement à comparer des nombres.

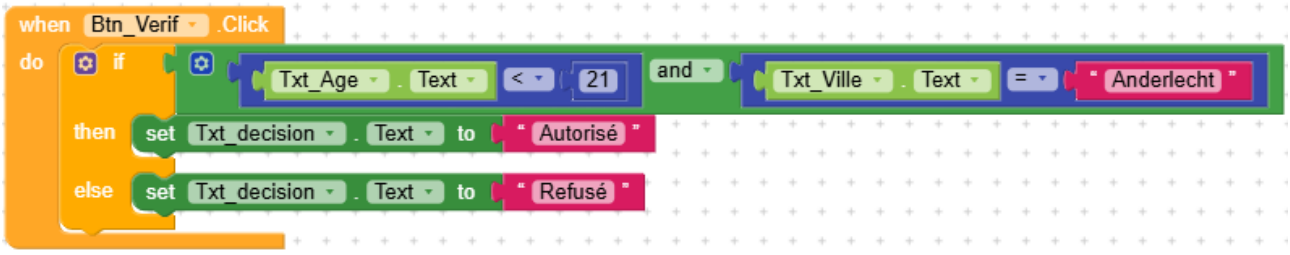
- On peut également comparer des textes.
  Attention, les espaces comptent ! Pierre est différent de Pierre (avec un espace à la fin).

- Les variables booléennes sont idéales pour les conditions.
  Exemple : *Si Connecté = Vrai Alors …* peut même s'écrire *Si Connecté Alors …* .
  En effet, dans un test, il n'est jamais obligatoire d'écrire = *Vrai*.

H. LES CONDITIONS MULTIPLES

Parfois, plusieurs critères doivent être vérifiés.

Par exemple, pour pouvoir participer à certains concours, il faut avoir moins de 21 ans et habiter Anderlecht. Les deux conditions doivent être remplies. On utilise dans ce cas l'opérateur **ET** (**AND**).



Dans le cas où une seule condition suffirait, on utilisera l'opérateur **OU** (**OR**).

L'opérateur **NON** (**NOT**), quant à lui, inverse une condition. Au lieu d'écrire, *Si connecté = Faux Alors …* ; on pourrait écrire *Si NON connecté Alors …*
C'est souvent plus simple à lire.

Attention, lorsque l'on écrit une condition de ce type, il ne faut jamais oublier de se mettre à la place de l'ordinateur et pas à celle de l'humain. En effet, nous ne parlons pas toujours de la même manière. Supposons par exemple que l'on souhaite accepter tous les habitants de Jette et de Laeken. Comme vous pouvez le constater, un être humain dit généralement qu'il veut les habitants de Jette ET de Laeken. Alors que l'ordinateur devra faire le test si ville = "Jette" OU ville = "Laeken" ! En effet, si l'algorithme contenait un ET, l'ordinateur n'aurait affiché aucun nom car le ET signifie que la ville doit être à la fois égale à Jette et à Laeken... Ce qui est impossible !

Pour écrire correctement un test, il suffit de se mettre à la place d'un ordinateur et se rendre compte qu'il n'a pas la même vue qu'un être humain. Les hommes ont une vue d'ensemble : "On veut tous les habitants de ..." ; les ordinateurs, quant à eux, traitent une donnée (ici, un habitant) à la fois : "Cet habitant habite-t-il à Jette OU à Laeken ?". Si on adopte bien cette vision des choses, nos tests seront bien meilleurs !

I. EXERCICE : CALCUL D'UNE REMISE

Créer :

- Une **Text Box** contenant le montant d'un achat ;
- Un **Button** ;
- Un **Label** qui affiche le montant final à payer.

Règle :

- Si le montant est supérieur ou égal à 100 €, appliquer une remise de 10 %
- Sinon, aucune remise.



J. EXERCICE : CONTROLE D'ACCES

Crée une application nommée *Contrôle d'accès*.

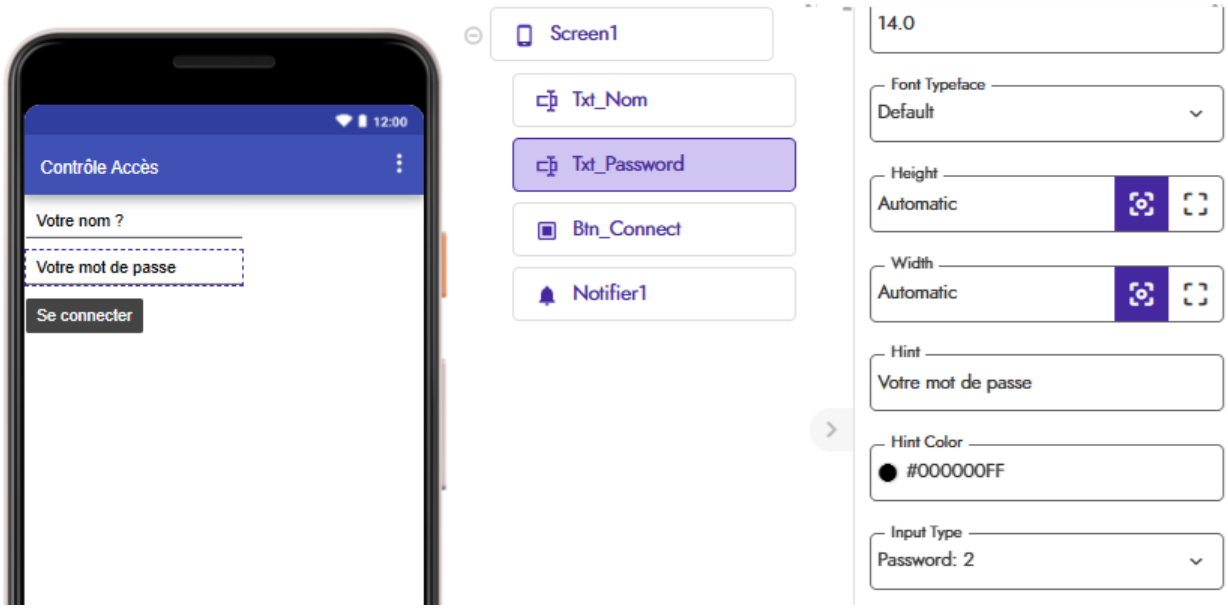
L'interface comprend :

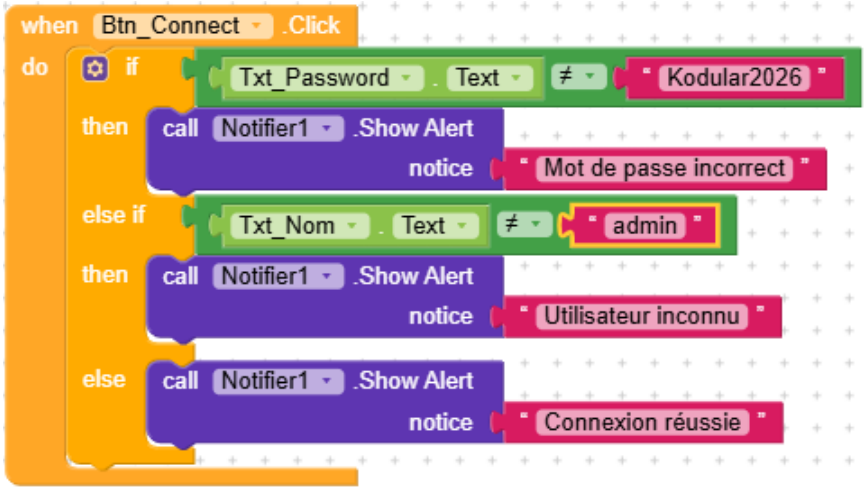
- Un champ *Nom d'utilisateur* ;
- Un champ *Mot de passe* ;
- Un **bouton** *Se connecter* ;
- Un composant Notifier.

Le fonctionnement est le suivant :

- Si le nom d'utilisateur est *admin* ET que le mot de passe est *Kodular2026*, afficher *Connexion réussie* ;
- Si seul le nom est correct mais que le mot de passe est erroné, afficher *Mot de passe incorrect* ;
- Si le nom d'utilisateur est inconnu, afficher *Utilisateur inconnu*.

Cet exercice combine plusieurs conditions et t'entraîne à construire une logique de décision complète.





4. LES BOUCLES : REPETER DES ACTIONS INTELLIGEMMENT

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre le principe des boucles ;
- Savoir quand utiliser une boucle plutôt que de répéter des blocs ;
- Utiliser les principales boucles de Kodular ;
- Parcourir une liste d'éléments ;
- Réaliser des traitements répétitifs de manière efficace.

B. INTRODUCTION

Les boucles (ou répétitive, ou itératives) sont des instructions spéciales qui permettent de dire à l'ordinateur qu'il doit exécuter plusieurs fois un certain nombre d'opérations, avec éventuellement quelques variantes (exemples : afficher les nombres de 1 à 100, afficher une table de multiplication, etc.).

Les types de boucles sont très variables d'un langage à l'autre. Ces types se différencient surtout par la façon dont on spécifie le nombre d'itérations. Une itération correspond à UNE exécution des instructions de la boucle.

Une boucle permet :

- De gagner du temps ;
- De réduire le nombre de blocs ;
- De faciliter la maintenance ;
- D'éviter les erreurs.

Kodular propose plusieurs types de boucles. Les plus utilisées sont :

- **repeat**
- **for each number**
- **for each item in list**
- **while** (selon les composants et extensions utilisées)

Nous allons les découvrir progressivement.

C. LA BOUCLE FOR EACH NUMBER FROM … TO …

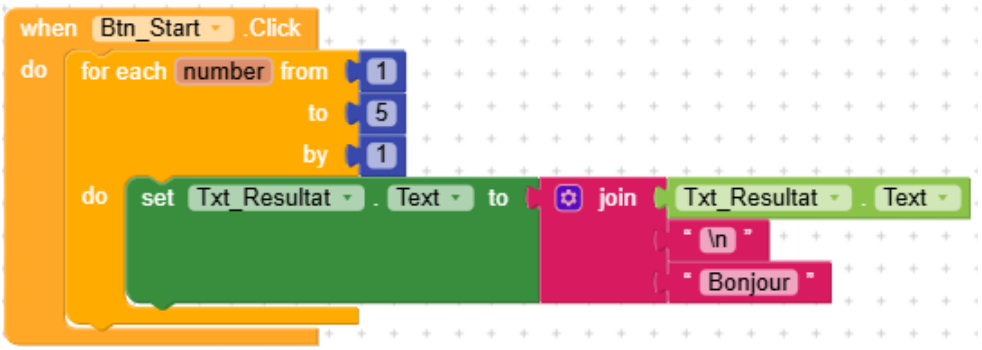
C'est la plus simple. Elle exécute une série d'actions un nombre précis de fois.

Les instructions se trouvant dans la boucle sont exécutées pour toutes les valeurs d'un compteur.

Avant la première itération de la boucle, ce compteur est initialisé à la valeur initiale FROM.

Ensuite, avant chaque itération, la commande vérifie si la condition est respectée (la valeur du compteur ne peut dépasser la valeur finale). Si c'est le cas, les instructions de la boucle sont exécutées et, ensuite, le compteur est incrémenté tel que c'est demandé (on ajoute la valeur incrément au compteur). La boucle s'arrête dès que la condition n'est plus respectée.

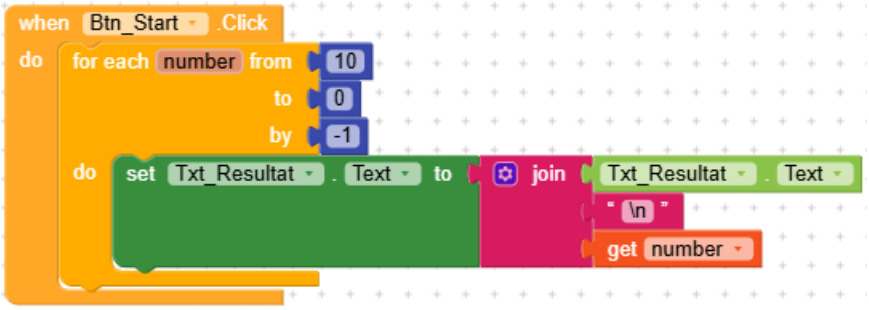
Notons qu'il est possible de mettre une valeur d'incrément négative. Dans ce cas, la valeur initiale devra être plus grande ou égale à la valeur finale.



Dans l'exemple ci-avant, la boucle écrit 5 fois *Bonjour* dans la zone *Txt_Resultat* en changeant de ligne entre chaque mot (grâce au **\n**).

Cette boucle est idéale lorsque tu connais à l'avance le nombre de répétitions. Par exemple, afficher les tables de multiplication.

L'exemple ci-après montre l'affichage d'un décompte de 10 à 0 :



D. LA BOUCLE FOR EACH ITEM IN LIST

Cette boucle parcourt automatiquement tous les éléments d'une liste.



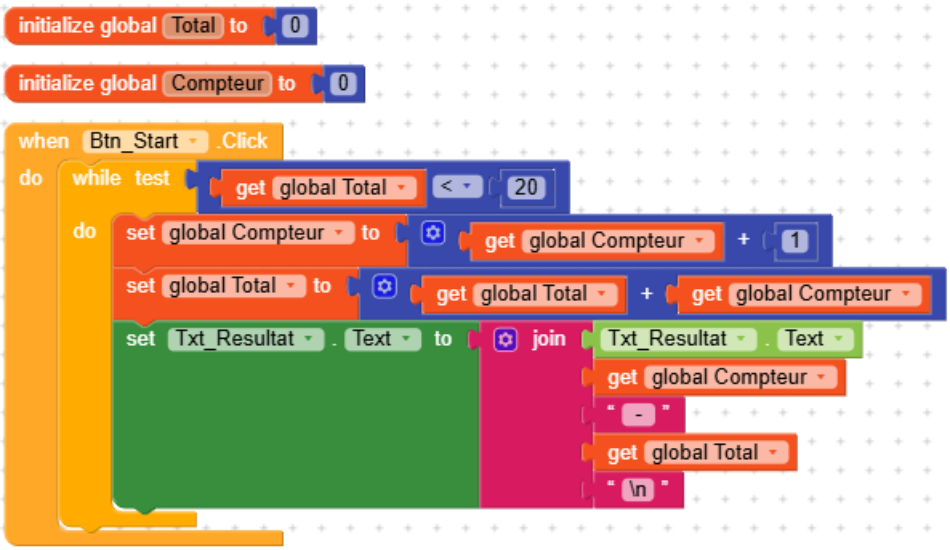
Cette boucle affichera le nom de chaque membre de la famille.

À chaque passage, elle exécute les mêmes instructions.

E. LA BOUCLE WHILE … DO

La boucle **while** signifie **Répète … Tant que...** Elle répète les instructions tant qu'une condition est vraie.

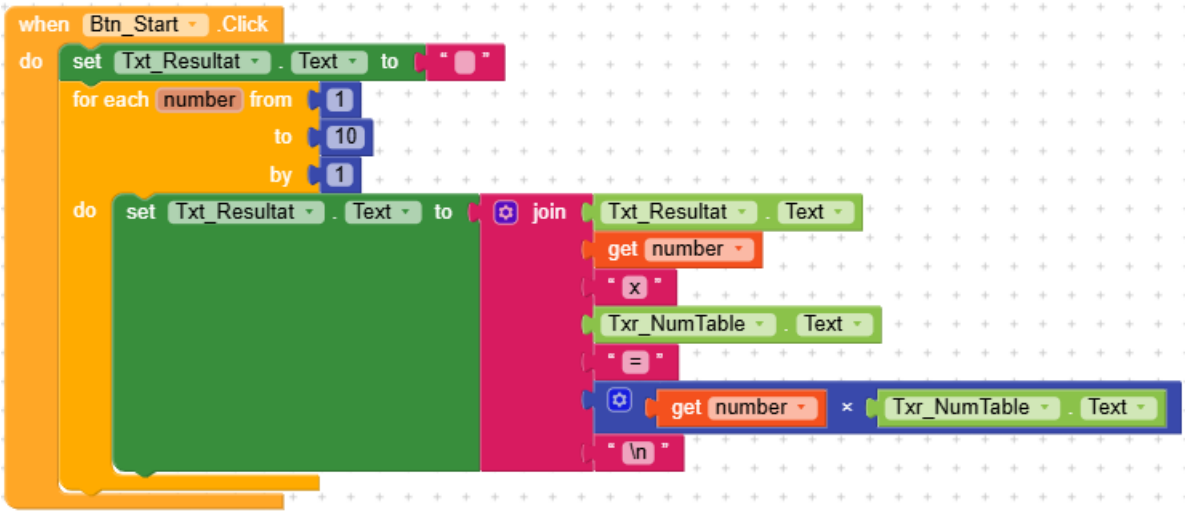
Dans l'exemple ci-après, on veut savoir combien de nombres on doit additionner à partir de 1 pour atteindre au moins 20.



Attention, si la condition reste toujours vraie, la boucle ne s'arrête jamais. On parle alors de **boucle infinie**.

F. EXEMPLE CONCRET : TABLE DE MULTIPLICATION

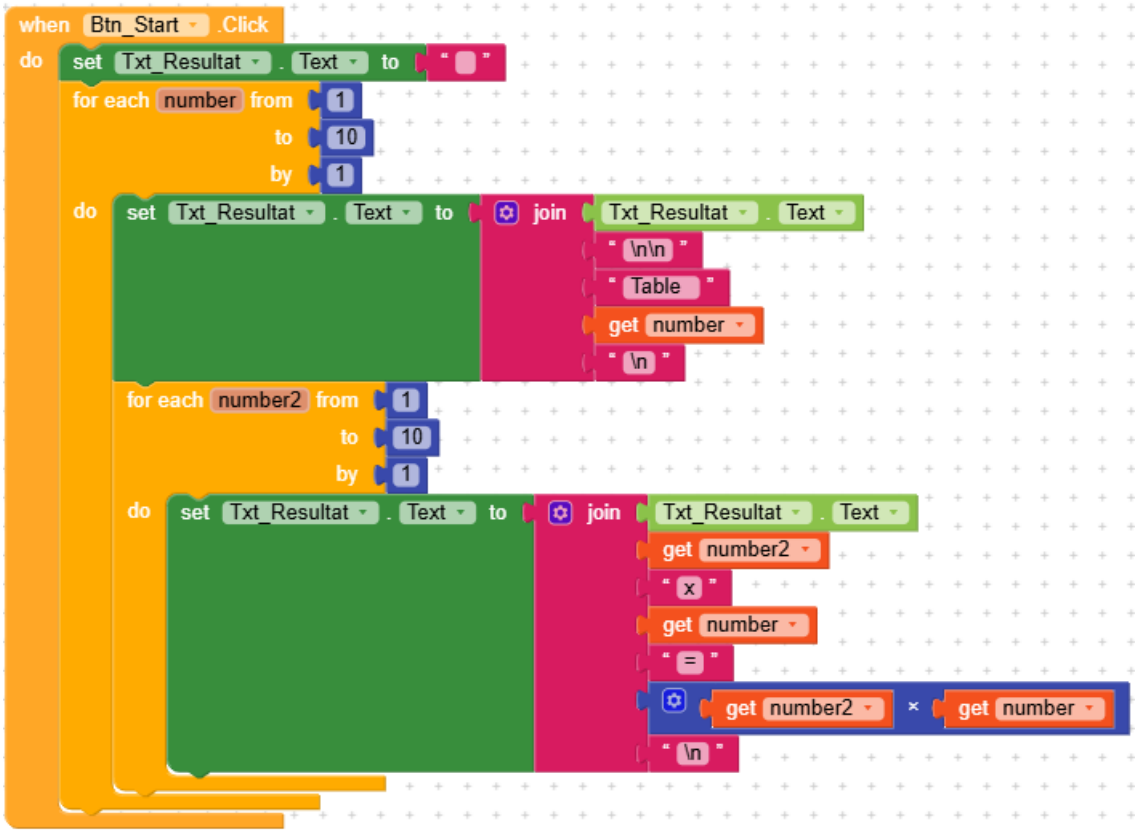
Crée une application dans laquelle l'utilisateur saisit le numéro correspondant à une table de multiplication et puis clique sur le bouton pour l'afficher. Attention, l'utilisateur doit pouvoir faire plusieurs essais. Il faut donc effacer le résultat précédent avant d'afficher une nouvelle table.



G. LES BOUCLES IMBRIQUEES

Une boucle peut contenir une autre boucle.

Les commandes suivantes permettent d'afficher les 10 tables de multiplication.



H. EXERCICE : ANALYSE DES NOTES D'UNE CLASSE

Conçois une application **Résultats de classe**.

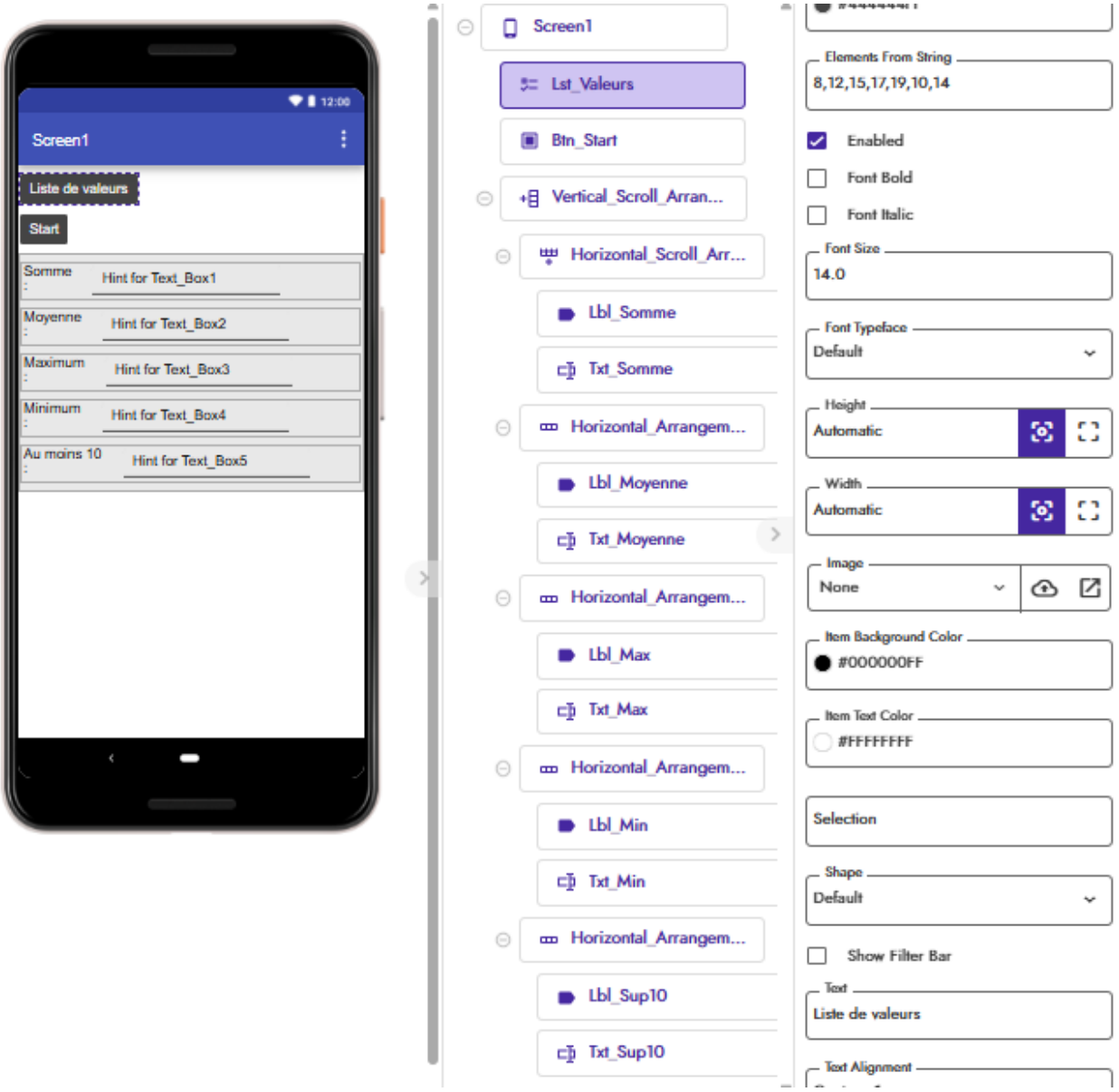
L'application contient une liste de notes (exemple : 8, 12, 15, 17, 19, 10, 14).

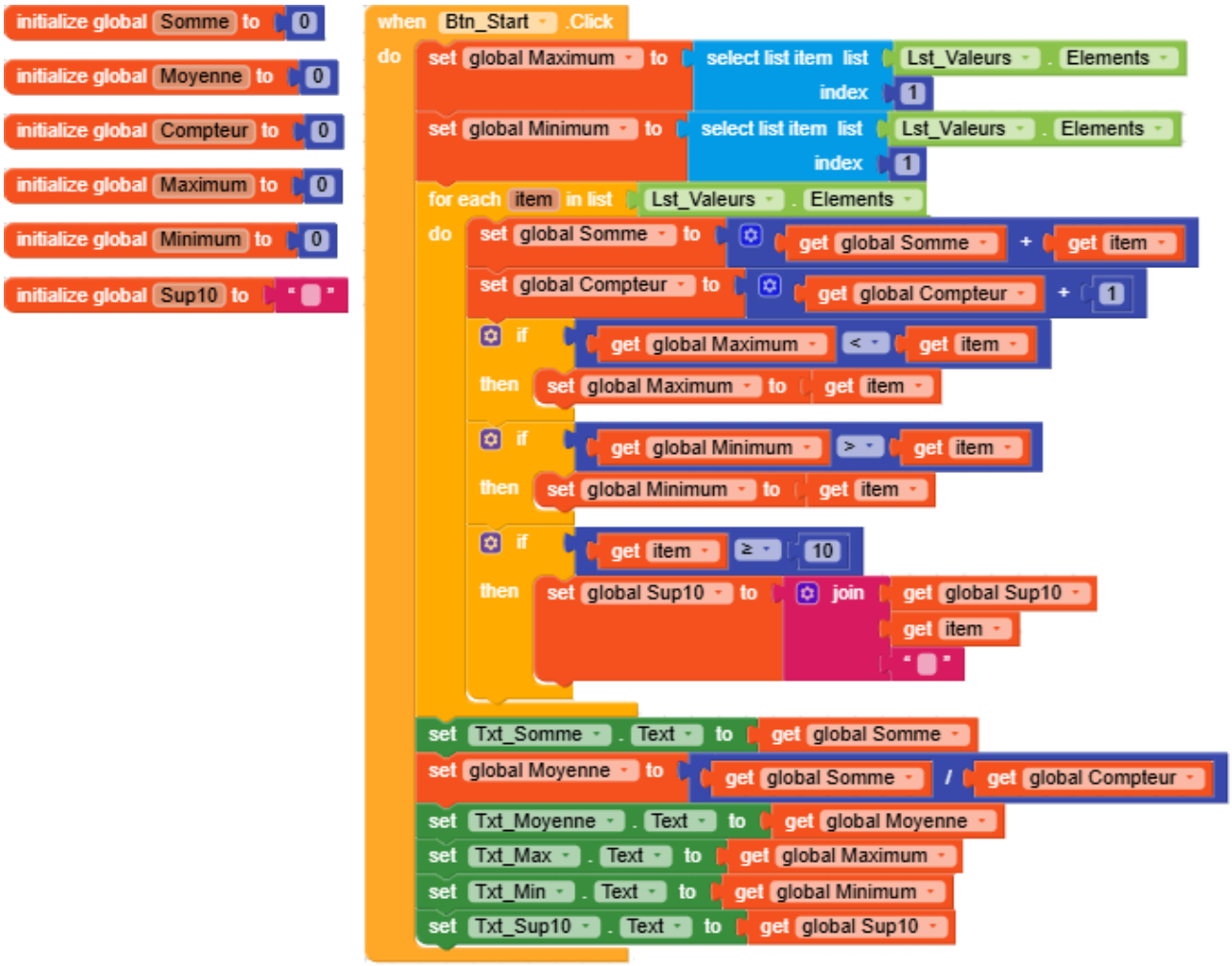
À l'aide d'une boucle, le programme doit :

- Calculer la somme des notes ;
- Calculer la moyenne ;
- Rechercher la note la plus élevée ;
- Rechercher la note la plus faible ;
- Afficher les notes supérieures ou égales à 10.

Affiche ensuite tous les résultats dans des Labels distincts.

Cet exercice mobilise les **boucles**, les **variables**, les **conditions** et les **listes**. Il constitue une excellente synthèse des notions étudiées depuis le début du chapitre.





5. LES PROCEDURES : ORGANISER SON CODE COMME UN DEVELOPPEUR PROFESSIONNEL

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre l'intérêt des procédures ;
- Créer des procédures avec et sans paramètres ;
- Réutiliser du code dans plusieurs événements ;
- Rendre tes programmes plus lisibles et plus faciles à maintenir ;
- Réaliser un projet complet utilisant les notions étudiées dans tout le chapitre.

B. INTRODUCTION

Imagine que tu prépares souvent la même recette.

À chaque fois, tu dois expliquer :

- Sortir les ingrédients ;
- Préchauffer le four ;
- Préparer la pâte ;
- Cuire pendant 30 minutes.

Au lieu de tout réécrire, tu pourrais simplement écrire *Préparer le gâteau*. Cette simple phrase représente toute une série d'actions.

En programmation, une **procédure** fonctionne exactement de cette manière. Elle regroupe plusieurs instructions sous un seul nom. Ensuite, il suffit d'appeler cette procédure chaque fois que l'on en a besoin.

C. POURQUOI CREER UNE PROCEDURE ?

Lorsque l'on écrit une application relativement importante, on se rend rapidement compte que l'on effectue à divers endroits les mêmes traitements (ou traitements très similaires). Cela peut aller de la simple vérification d'une valeur encodée à un code nettement plus élaboré.

Un programmeur débutant se contentera de répéter plusieurs fois les mêmes commandes aux différents endroits. Ce n'est bien évidemment pas une bonne manière de procéder. Ce code sera inutilement allongé, sa lisibilité diminuée et le risque d'erreur augmenté. De plus, cela entraînera des problèmes considérables de maintenance car en cas de modification du code, il va falloir traquer toutes les apparitions plus ou moins identiques de ce code pour faire convenablement la modification ! Avec le risque d'en oublier certaines...

C'est afin de limiter tous ces problèmes que les procédures ont été créées. Une procédure est un groupe d'instructions qui ne sera exécuté que lorsque le code principal en a besoin. Celui-ci pourra l'exécuter autant de fois que nécessaire.

Les avantages des procédures sont :

- Elles permettent de n'écrire qu'une seule fois des parties de codes nécessaires à plusieurs reprises.
- Elles permettent de découper un problème important en plusieurs sous-problèmes plus simples et donc simplifient la conception de l'application.
- Elles augmentent la lisibilité de l'application.

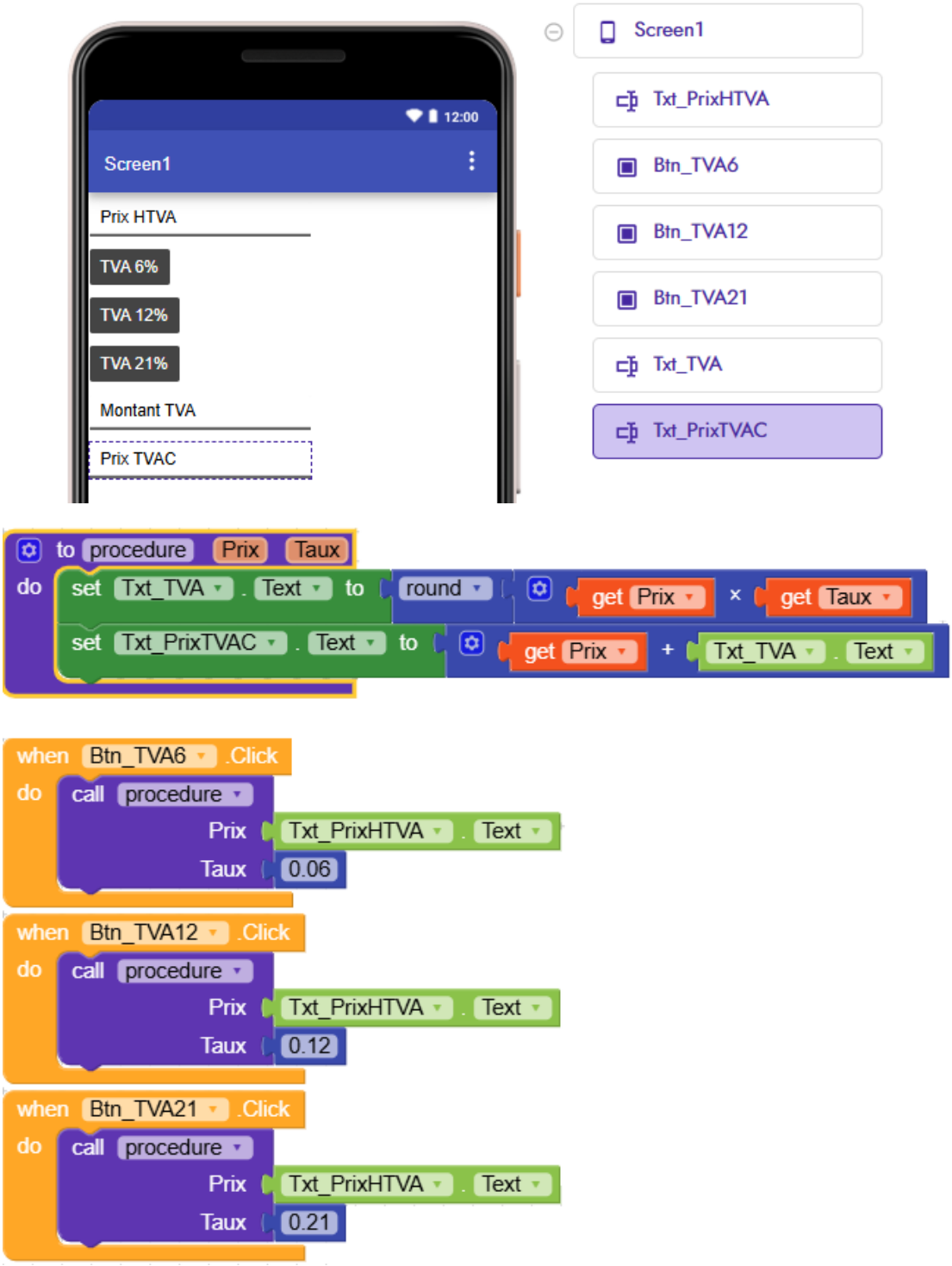
D. CREER ET APPELER UNE PROCEDURE

Kodular permet de créer une procédure dans la catégorie **Procedures** du panneau **Blocks**. Cette procédure peut contenir autant de blocs que nécessaire.

De plus, une procédure peut recevoir des informations. Celles-ci sont appelées **paramètres**.

Pour exécuter une procédure, il suffit de l'appeler en lui donnant les paramètres requis.

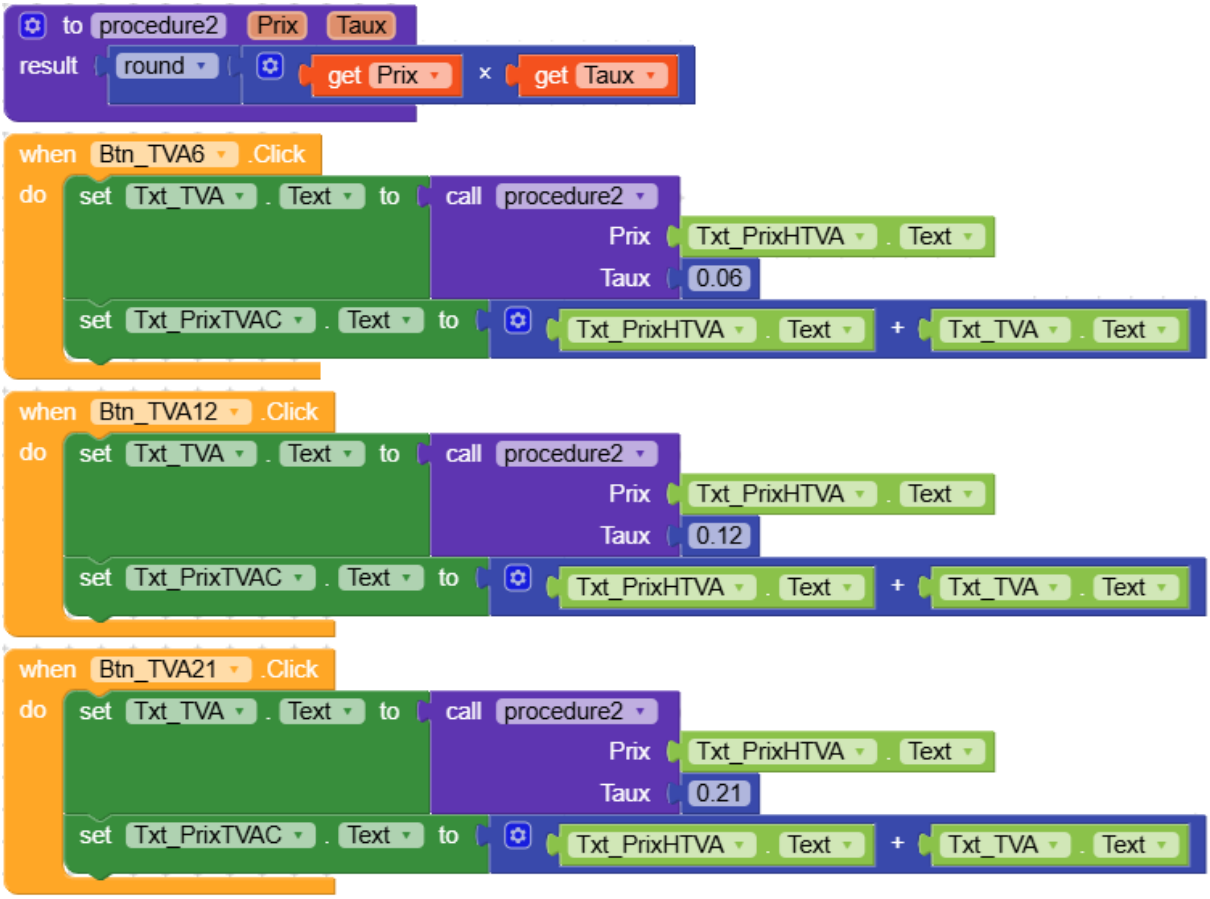
Dans l'exemple ci-après, l'utilisateur peut demander à l'application de calculer la TVA et le prix TVAC d'un montant initial encodé à l'écran. Le taux de TVA appliqué sera celui correspondant au bouton cliqué (6, 12 ou 21%).



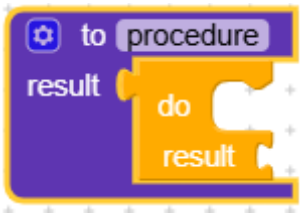
E. LES PROCEDURES QUI RENVOIENT UN RESULTAT

Certaines procédures ne se contentent pas d'exécuter des actions. Elles retournent une valeur.

Si, dans l'exemple précédent, on avait voulu créer une procédure plus généraliste afin de pouvoir l'utiliser dans plus de situations, on aurait dû créer une procédure qui se contentait de calculer le montant de la TVA et renvoyer cette valeur à la commande l'ayant appelée.



Remarque, qu'ici, la procédure n'a qu'une seule instruction qui correspond au calcul et au renvoi de la valeur attendue. Mais il peut arriver qu'il faille exécuter plusieurs instructions avant de pouvoir renvoyer une valeur. Dans ce cas, on utilisera la combinaison de blocs suivante (Blocs de **Procedures** et de **Control**) :



F. EXERCICE : VIDER UN FORMULAIRE

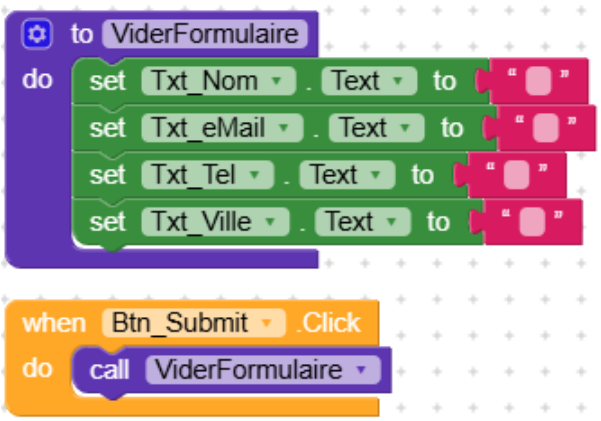
Crée une interface contenant :

- Nom ;
- Téléphone ;
- Adresse e-mail ;
- Ville.

Crée une procédure *ViderFormulaire* qui :

- Effacer tous les champs ;
- Replacer le curseur dans le premier champ.

Après chaque soumission, appelle simplement cette procédure.


to ViderFormulaire
do
set Txt_Nom . Text to
set Txt_eMail . Text to
set Txt_Tel . Text to
set Txt_Ville . Text to
when Btn_Submit .Click
do
call ViderFormulaire

Les listes et les dictionnaires

1. COMPRENDRE LES LISTES : STOCKER PLUSIEURS INFORMATIONS DANS UNE VARIABLE

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre ce qu'est une liste ;
- Créer une liste dans Kodular ;
- Accéder aux éléments d'une liste ;
- Connaître les principales utilisations des listes ;
- Comprendre pourquoi elles sont indispensables dans les applications modernes.

B. INTRODUCTION

Jusqu'à présent, nous avons utilisé des variables. Une variable permet de stocker une seule information. Par exemple, un nom. Mais imaginons maintenant que tu souhaites mémoriser les prénoms de toute une classe. Tu pourrais créer une variable pour chaque élève. Cette méthode fonctionne… Mais elle devient rapidement ingérable. Imagine une école de 600 élèves. Faudrait-il créer 600 variables ? Bien sûr que non. C'est précisément pour résoudre ce problème que les listes existent.

C. QU'EST-CE QU'UNE LISTE ?

Une liste est une variable capable de contenir plusieurs informations. Au lieu d'une seule valeur, elle peut en stocker des dizaines, des centaines, voire des milliers.

Une liste est donc un conteneur de plusieurs valeurs. Elles sont présentes dans presque toutes les applications.

D. LES ELEMENTS D'UNE LISTE

Chaque information contenue dans une liste est appelée un **élément**. Pae exemple, si on crée une liste de fruits, chaque fruit est un élément de la liste. Chaque élément est indépendant des autres.

Contrairement à certaines bases de données, une liste conserve toujours l'ordre dans lequel les éléments ont été ajoutés. Chaque nouvel élément est ajouté par défaut à la fin de la liste.

Chaque élément possède une position. Cette position est appelée **index**. Dans Kodular, les listes commencent à **1**. C'est une différence importante avec certains langages de programmation qui commencent à **0**.

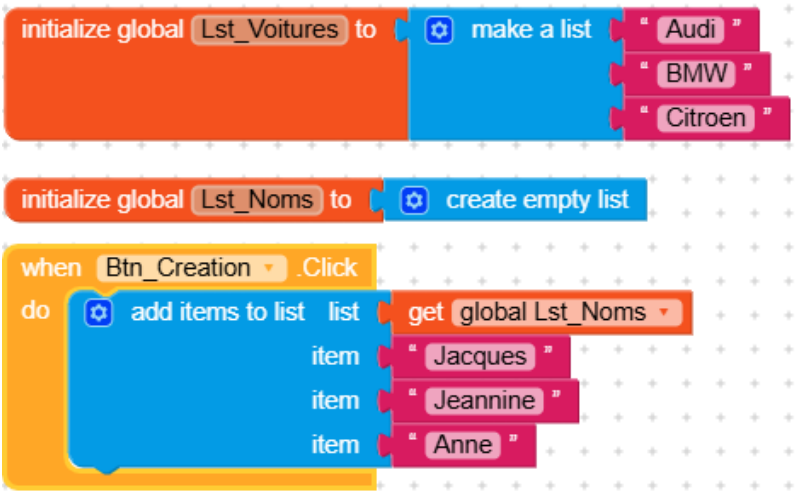
E. LES TYPES DE DONNEES DANS UNE LISTE

Une liste peut contenir différents types de données : texte, nombre, valeur logique, liste (On parle alors de liste imbriquée).

F. CREER UNE LISTE DANS KODULAR

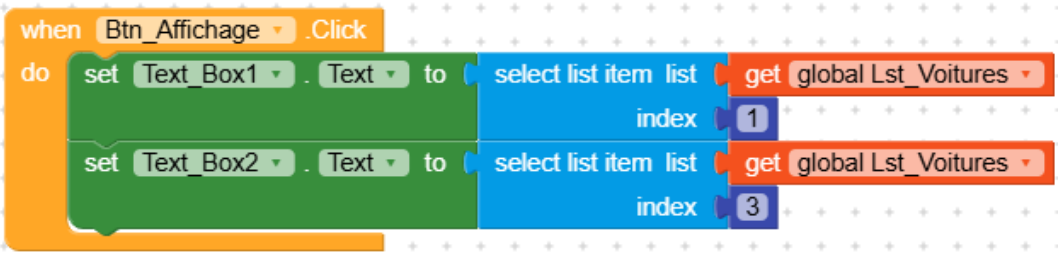
Les blocs permettant de créer une liste se trouvent dans la catégorie **Lists** :

- **Create empty list**
  Il crée une liste vide.
- **Make a list**
  Crée une liste contenant directement plusieurs éléments.



Ces listes sont immédiatement prêtes à être utilisées.

Pour vérifier si la création fonctionne correctement, on peut ajouter des blocs semblables à :



G. COMBIEN D'ELEMENTS CONTIENT UNE LISTE ?

Il est souvent nécessaire de connaître la taille d'une liste.



Cette information est utile pour :

- Afficher le nombre de contacts ;
- Compter les produits d'un panier ;
- Vérifier si une liste est vide.

H. LES LISTES DANS LA VIE QUOTIDIENNE

Tu utilises déjà des listes tous les jours.

Exemples :

- Les contacts de ton téléphone ;
- Les chansons de ta playlist ;
- Les favoris de ton navigateur ;
- Les applications installées ;
- Les photos de ta galerie ;
- Les messages reçus.

Toutes ces informations sont stockées sous forme de listes.

I. EXERCICE : MES FILMS PREFERES

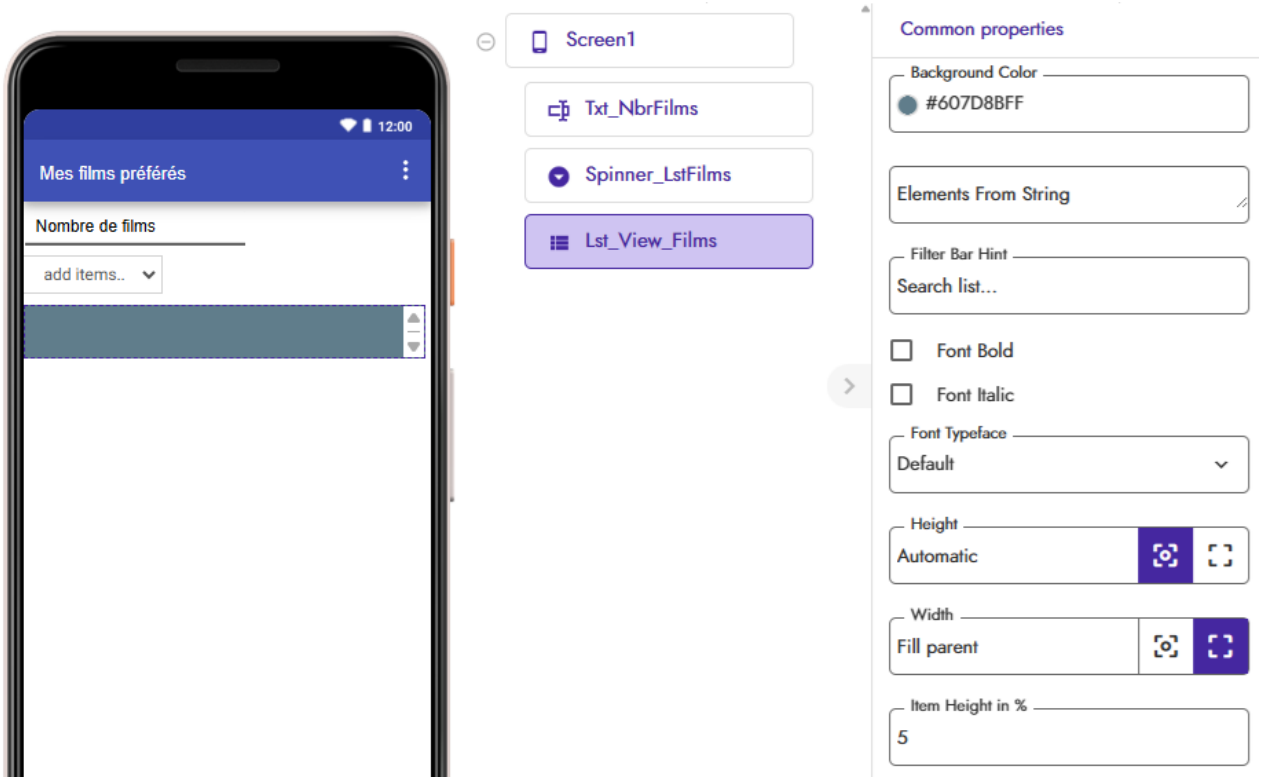
Crée une application intitulée *Mes films préférés* qui contient :

- Un Label indiquant le nombre de films ;
- Une liste contenant au moins 10 films de ton choix.
- Un Spinner ou une List View affichant les titres ;

Au démarrage de l'application :

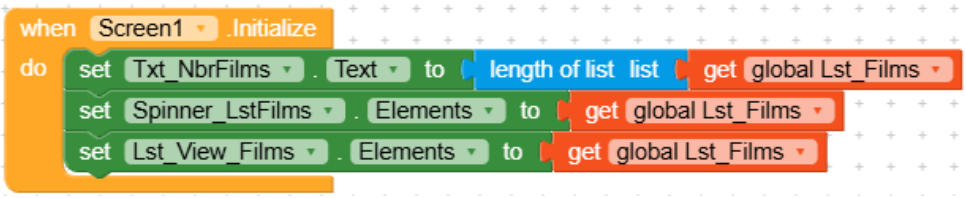
- Le Spinner (ou la List View) est chargé à partir de la liste et affiche tous les films ;
- Le Label indique le nombre total de films enregistrés.

Tu utiliseras cette application dans les prochaines parties pour apprendre à ajouter, supprimer, modifier et rechercher des éléments dans une liste.





ou plus simplement



2. MANIPULER LES LISTES : AJOUTER, MODIFIER ET SUPPRIMER DES ELEMENTS

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Ajouter un élément dans une liste ;
- Insérer un élément à une position précise ;
- Modifier une valeur existante ;
- Supprimer un élément ;
- Vider complètement une liste ;
- Utiliser ces opérations dans des applications concrètes.

B. INTRODUCTION

Dans le premier chapitre, tu as découvert ce qu'est une liste. Mais une liste n'est pas figée. Au contraire ! Elle évolue en permanence.

Prenons l'exemple de la liste de contacts de ton téléphone. Chaque jour, tu peux :

- Ajouter un nouveau contact ;
- Modifier un numéro de téléphone ;
- Supprimer un ancien contact.

Les listes de Kodular fonctionnent exactement de la même manière. Dans cette partie, nous allons apprendre à les manipuler.

C. AJOUTER UN ELEMENT A UNE LISTE

L'opération la plus fréquente consiste à ajouter une nouvelle information.

Exemple :
On crée au départ une liste de fruits (Banane, Fraise, Pomme). Ensuite, via une Text Box et un Bouton, on ajoute un nouveau fruit (Cerise) à la liste. Si souhaité, on ajoutera plusieurs fruits en répétant l'opération d'ajout. Les nouveaux éléments seront, par défaut, ajoutés à la fin de la liste.

La liste étant affichée à l'écran via une List View à l'initialisation de l'écran, il faut penser à rafraîchir la List View à chaque modification de la liste de fruits (via 

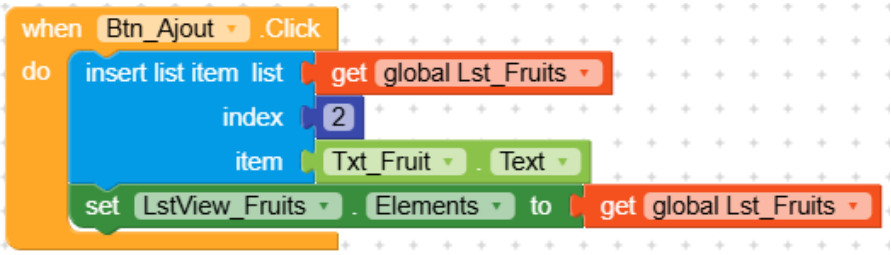
).



Ce que l'on a fait ici pour une liste de fruits peut évidemment se faire pour une liste de contacts.

D. INSERER UN ELEMENT A UNE POSITION PRECISE

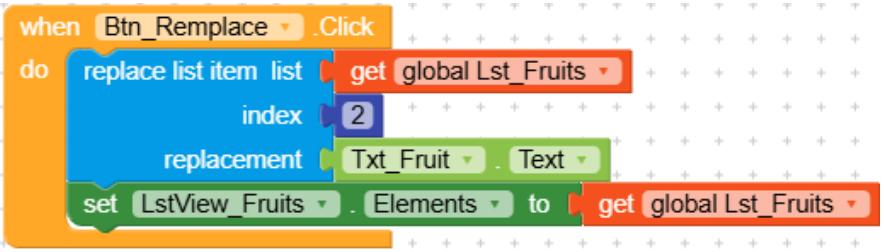
Il est parfois nécessaire d'insérer un élément au milieu de la liste. Dans ce cas, on utilisera la commande **Insert List Item** en indiquant la position dans la liste souhaitée au moyen du paramètre **Index**.



Tous les éléments suivants sont automatiquement décalés.

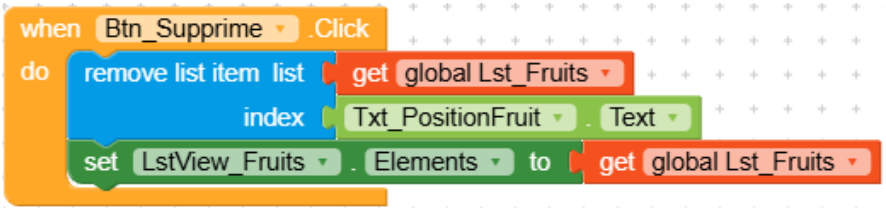
E. MODIFIER UN ELEMENT

Une liste permet également de remplacer une valeur. Le nombre d'éléments ne change pas. Seule la valeur est remplacée.



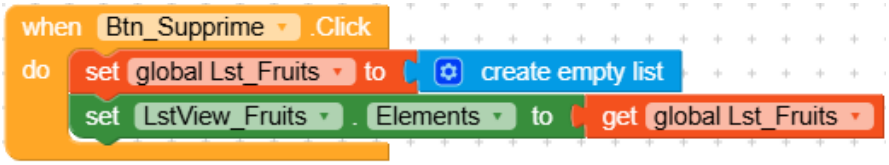
F. SUPPRIMER UN ELEMENT

Tu peux supprimer un élément qui n'est plus utile. Tous les éléments suivants remontent automatiquement et les index sont recalculés.



G. VIDER COMPLETEMENT UNE LISTE

Pour vider entièrement une liste en une opération, il faut procéder comme suit :



H. VERIFIER SI UNE LISTE EST VIDE

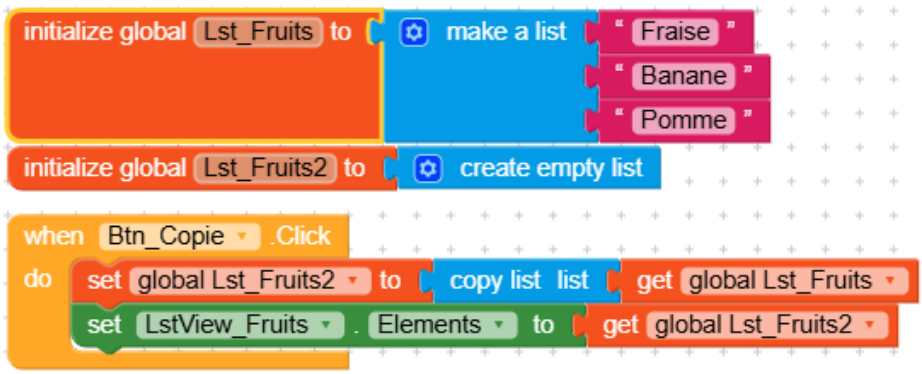
Avant de parcourir une liste, il est conseillé de vérifier si elle contient des éléments. Pour ce faire, soit on utilise le test ad hoc, soit on teste si sa longueur est égale à 0 :



Cette vérification évite de nombreuses erreurs.

I. COPIER UNE LISTE

Kodular nous permet de demander de faire une copie indépendante d'une liste. Indépendante signifie qu'après la copie, toute modification d'une liste n'entraînera pas la modification de l'autre liste.



J. EXERCICE : GESTIONNAIRE DE TACHES

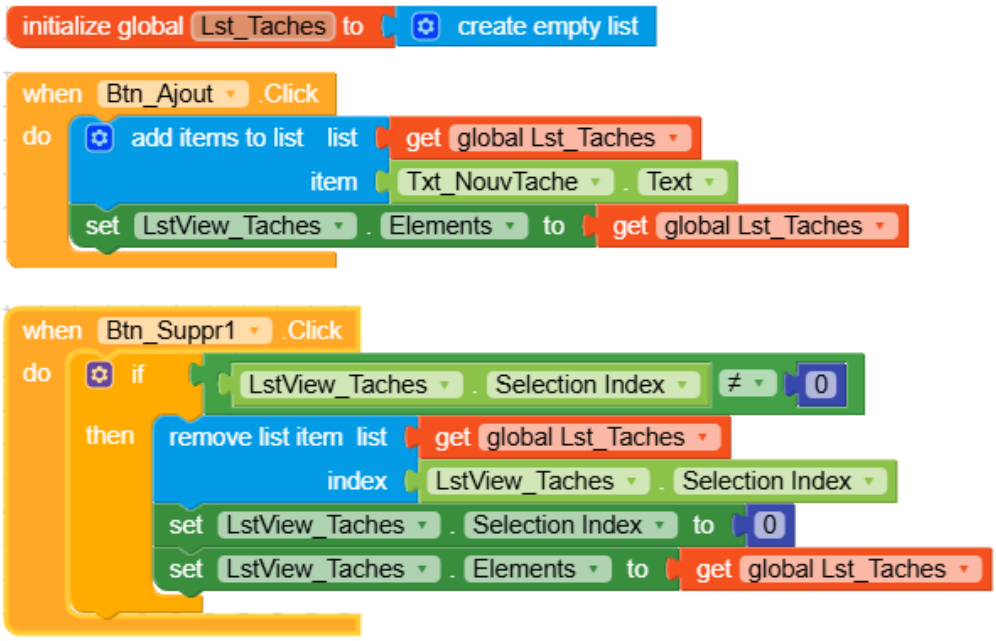
Crée une application intitulée *Mes tâches*.

L'application contient :

- Un champ de saisie ;
- Un bouton *Ajouter* ;
- Un bouton *Supprimer la tâche sélectionnée* ;
- Un bouton *Vider la liste* ;
- Une **List View**.

Le fonctionnement est le suivant :

- L'utilisateur saisit une nouvelle tâche et clique sur *Ajouter* ;
- La tâche est ajoutée à la liste et la **List View** est immédiatement mise à jour ;
- Si une tâche est sélectionnée, le bouton *Supprimer* la retire de la liste ;
- Le bouton *Vider la liste* efface toutes les tâches après une demande de confirmation via **Notifier**.





3. RECHERCHER, TRIER ET PARCOURIR LES LISTES

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Rechercher un élément dans une liste ;
- Vérifier si une valeur est présente ;
- Retrouver la position d'un élément ;
- Parcourir une liste à l'aide d'une boucle ;
- Trier une liste ;
- Créer des recherches simples dans tes applications.

B. INTRODUCTION

Les listes sont très utiles pour stocker des informations. Mais leur véritable avantage apparaît lorsque l'on doit retrouver rapidement une donnée.

Imagine ton téléphone.

Il contient peut-être :

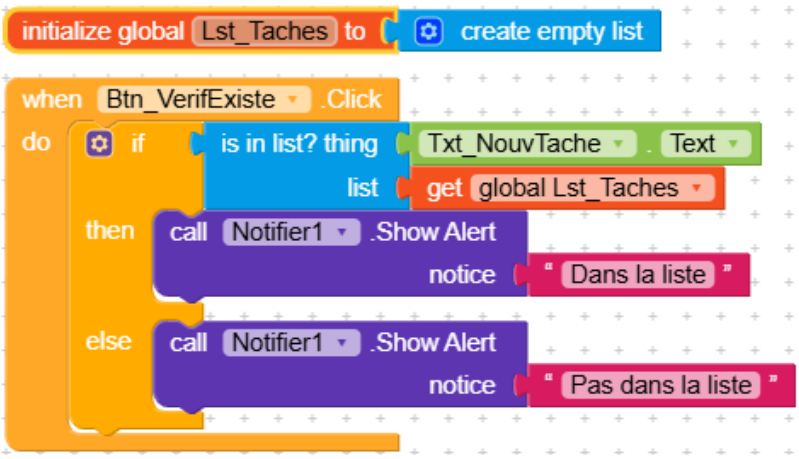
- 500 contacts ;
- 3 000 photos ;
- 2 000 chansons.

Si tu devais faire défiler toute la liste à chaque recherche, cela deviendrait vite impossible. Les applications modernes utilisent donc des méthodes permettant de rechercher, parcourir et organiser les données efficacement. C'est ce que nous allons apprendre dans ce chapitre.

C. RECHERCHER UN ELEMENT

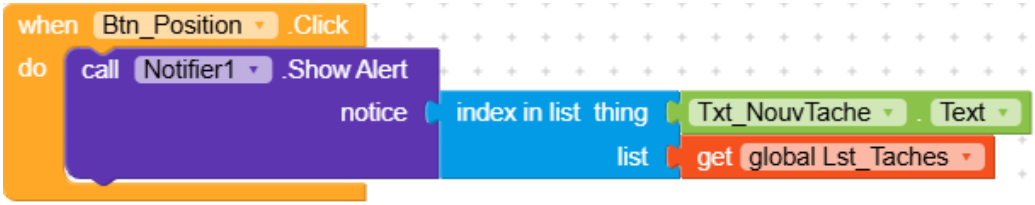
La première opération consiste à savoir si une information existe dans une liste. En effet avant d'ajouter un élément, il est souvent préférable de vérifier qu'il n'existe pas déjà.

Prenons une liste de tâches. Tu souhaites savoir si "Faire la vaisselle" est présente. La recherche renvoie True ou False selon le cas.



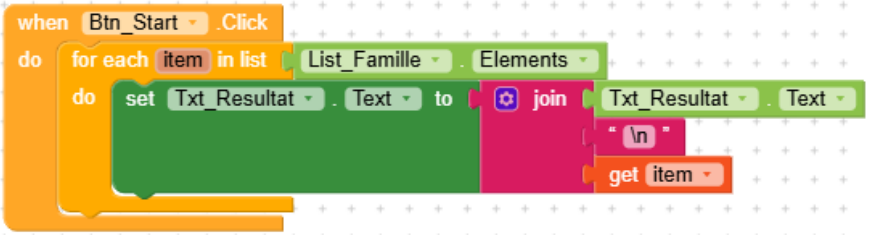
D. TROUVER LA POSITION D'UN ELEMENT

Il ne suffit pas toujours de savoir si un élément existe. On souhaite parfois connaître sa position. La commande **Index in list** renvoie sa position dans la liste ou 0 si elle n'existe pas.



E. PARCOURIR UNE LISTE

Très souvent, on souhaite traiter tous les éléments d'une liste. Pour cela, on utilise une boucle comme expliqué dans la section Partie 4 / Chapitre 4 / Section D : *la boucle FOR EACH ITEM IN LIST* :



La boucle lit et traite successivement chaque élément de la liste.

L'usage d'autres boucles est bien évidemment possible, si l'on utilise un compteur et l'index de la liste par exemple.

Les boucles sont fort utilisées pour traiter les listes car cela permet de traiter facilement tous les éléments et/ou d'en sortir un résultat final.

Exemples :

Cette technique est utilisée dans :

- Les statistiques ;
- Les tableaux de bord ;
- Les applications scolaires (ex.: nombre de réussites, d'échecs, …).

Comment faire ?

- **Si l'on veut rechercher la plus grande valeur d'une liste.**
  Au départ, on prend le premier élément de la liste pour valeur maximum. Puis chaque nouvel élément est comparé au maximum temporaire et s'il est plus grand, il devient le nouveau maximum temporaire.
  A la fin de la liste, le maximum temporaire est le maximum de la liste.
- **Si l'on veut rechercher la plus petite valeur d'une liste.**
  On procède en suivant un procédé semblable.
- **Si l'on veut calculer une moyenne de tous les nombres d'une liste.**
  On additionne tous les nombres en parcourant la liste et à la fin on divise le total obtenu par le nombre de valeurs additionnées.

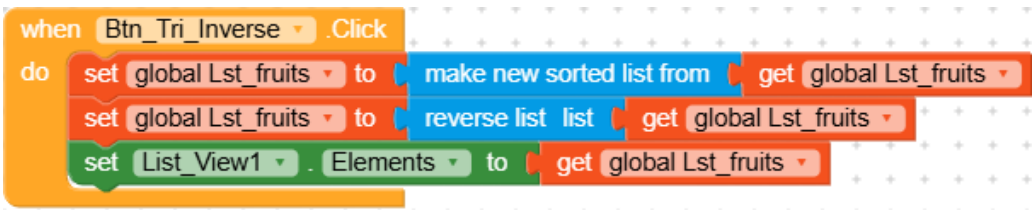
F. TRIER UNE LISTE

Le tri consiste à réorganiser les éléments.

Kodular nous permet de trier une liste dans l'ordre croissant via la commande **Make new sorted list from**. Cette commande crée en fait une nouvelle liste triée que l'on peut assigner à la liste originale ou une autre.



Si l'on veut trier dans l'ordre décroissant, on doit d'abord trier, puis lui dire d'inverser l'ordre des éléments via la commande **Reverse list** :

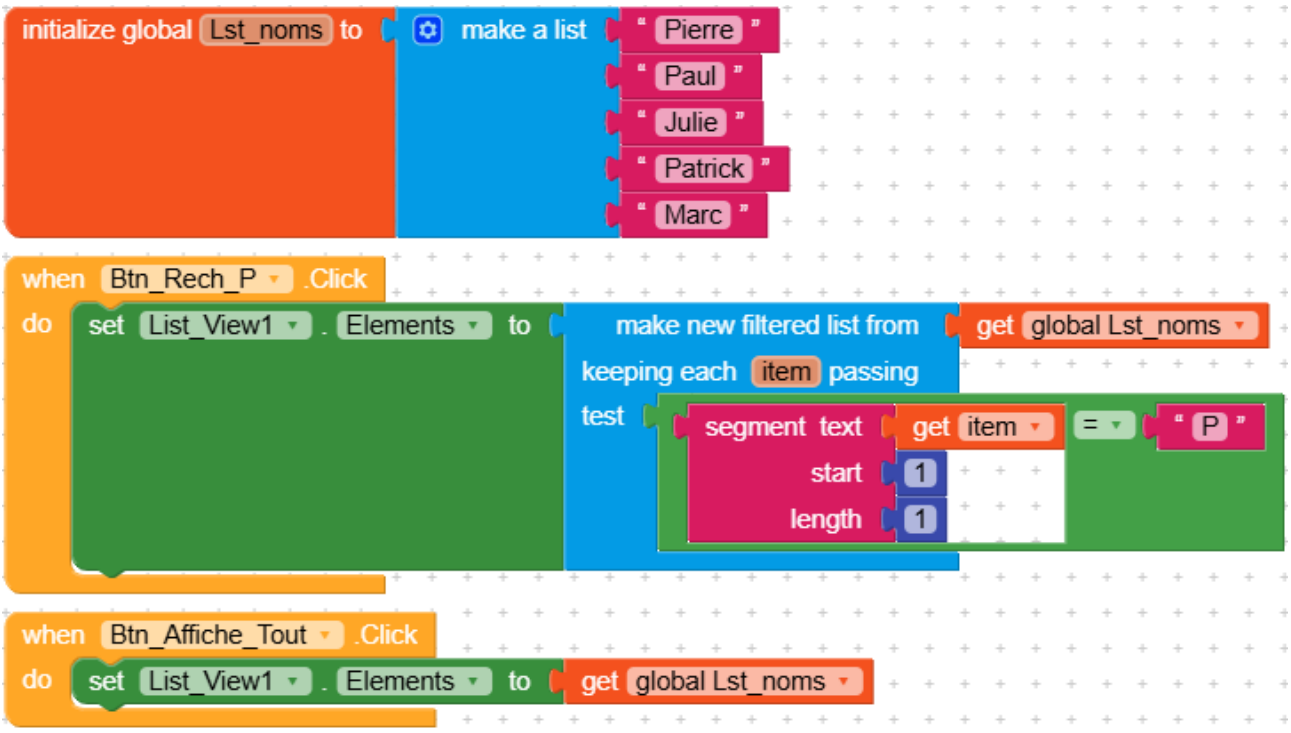


G. FILTRER UNE LISTE

Il s'avère parfois intéressant de filtrer l'affichage en fonction d'un critère.

Supposons que j'aie une liste de prénoms et que je ne souhaite visualiser que ceux dont la première lettre est P.

Voici comment faire :



Si l'on regarde attentivement, on verra que la commande de filtrage ne s'applique pas sur la liste originale mais sur la liste affichée. C'est très important de procéder ainsi car si on appliquait le filtre sur la liste de départ, certains éléments seraient effacés définitivement. Ici, on peut voir que si l'on clique sur le bouton pour tout réafficher, tous les éléments reviennent.

H. LES PERFORMANCES

Une liste de cinq éléments est parcourue très rapidement.

Mais certaines applications manipulent :

- 5 000 contacts ;
- 20 000 produits ;
- 100 000 lignes.

Dans ce cas, il faut limiter les recherches inutiles. C'est pourquoi les développeurs privilégient souvent :

- Les dictionnaires ;
- Les bases de données ;
- Les index.

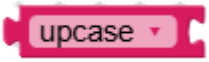
Nous découvrirons ces techniques dans les prochains chapitres.

I. ASTUCE DE DEVELOPPEUR

Lorsque plusieurs traitements doivent être effectués sur une même liste (par exemple calculer la somme, rechercher le maximum et compter les éléments), essaie de les réaliser dans une seule boucle. Au lieu de parcourir la liste trois fois, tu ne la parcours qu'une seule fois. Le programme est plus rapide et consomme moins de ressources.

J. LES ERREURS FREQUENTES

- Rechercher sans tenir compte des majuscules
  Les mots Pierre et pierre sont différents. Si nécessaire, convertis les textes dans le même format avant de comparer via la commande **Block** / **Text**



- Oublier de vérifier si la liste est vide
  Une recherche dans une liste vide ne renvoie aucun résultat. Teste toujours la taille de la liste avant de la parcourir.

- Parcourir plusieurs fois inutilement
  Chaque parcours prend du temps. Regroupe les traitements lorsque c'est possible.

- Modifier une liste pendant son parcours
  Supprimer ou ajouter des éléments pendant qu'une boucle la parcourt peut produire des résultats inattendus. Il est souvent préférable de terminer le parcours avant de modifier la liste.

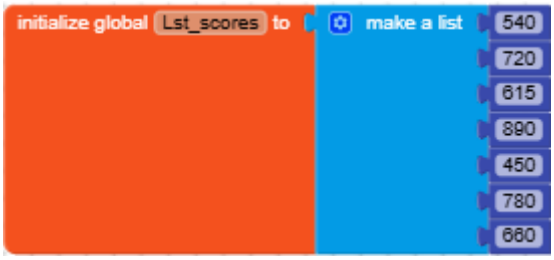
K. EXERCICE : CLASSEMENT DES JOUEURS

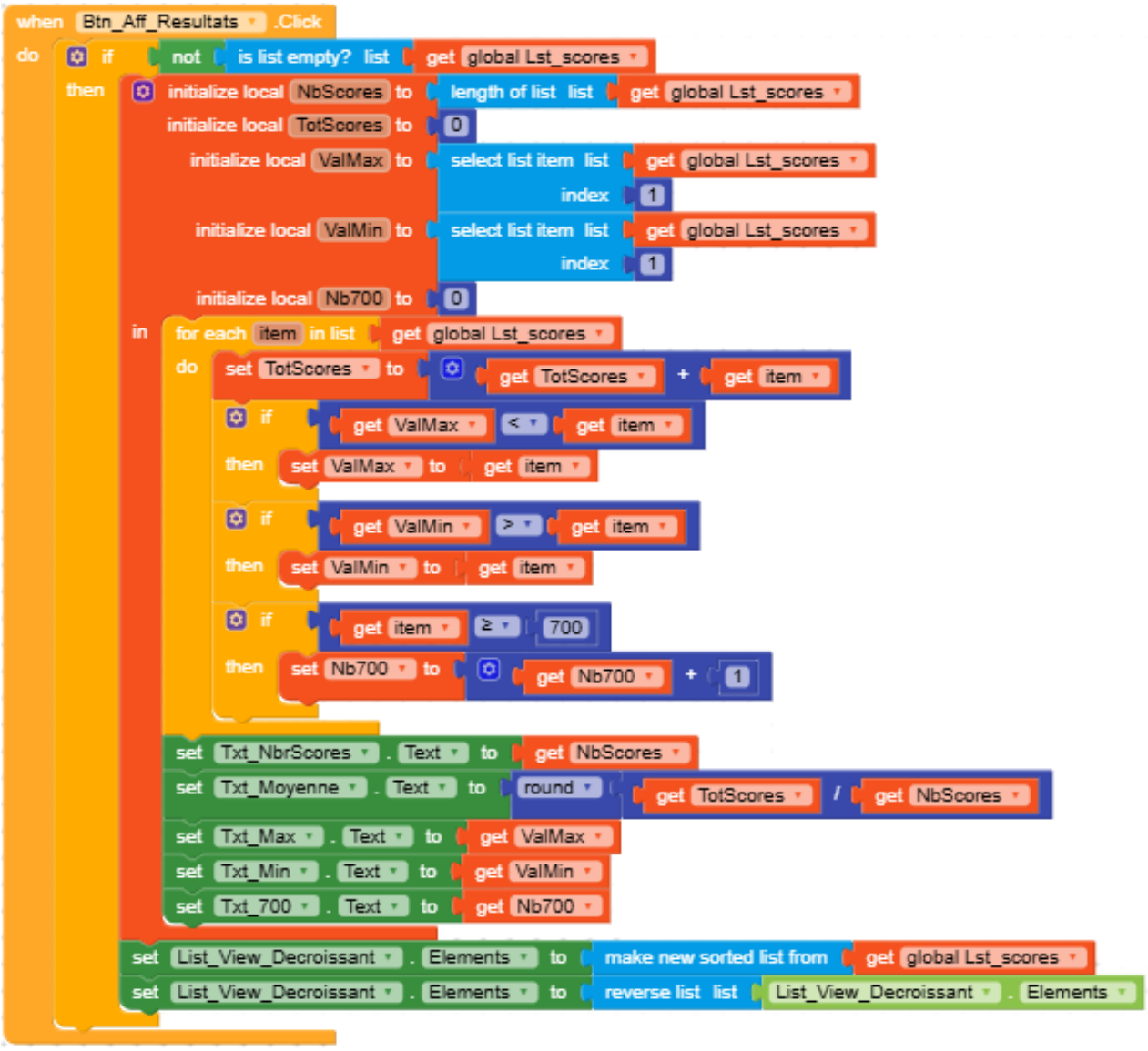
Crée une application *Classement des joueurs*.

L'application contient une liste de scores : 540, 720, 615, 890, 450, 780, 660.

Le programme doit permettre de :

- Afficher le nombre total de scores ;
- Calculer la moyenne ;
- Afficher le meilleur score ;
- Afficher le plus faible score ;
- Compter le nombre de scores supérieurs ou égaux à **700** ;
- Trier les scores par ordre décroissant ;
- Afficher les résultats dans des Labels ou une **List View**.





Cet exercice constitue une excellente synthèse de la manipulation des listes et prépare l'étude des **dictionnaires**, qui permettront d'associer une clé (par exemple un nom de joueur) à une valeur (son score).

4. LES DICTIONNAIRES : ASSOCIER DES INFORMATIONS INTELLIGEMMENT

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre ce qu'est un dictionnaire ;
- Faire la différence entre une liste et un dictionnaire ;
- Créer un dictionnaire dans Kodular ;
- Ajouter, modifier et supprimer des informations ;
- Rechercher rapidement une valeur à partir d'une clé ;
- Utiliser les dictionnaires dans des applications concrètes.

B. INTRODUCTION

Jusqu'à présent, nous avons utilisé des **listes**. Mais imaginons maintenant que tu souhaites mémoriser un carnet d'adresses. Tu ne veux pas uniquement conserver les noms. Tu souhaites également enregistrer le numéro de téléphone, l'adresse e-mail, la ville, … Une simple liste ne suffit plus. Il faut une structure capable d'associer plusieurs informations entre elles. C'est exactement le rôle d'un **dictionnaire**.

C. QU'EST-CE QU'UN DICTIONNAIRE ?

Un dictionnaire est une collection composée de **clés** et de **valeurs**. Chaque clé étant associée à une seule valeur. Par exemple, une clé correspondant au nom d'une personne et une valeur à son numéro de téléphone. Ou alors, un mot pour clé et sa définition pour valeur comme dans un dictionnaire français. Kodular fonctionne exactement de la même manière.

Une clé permet d'identifier une information, elle doit donc être unique.

Les valeurs peuvent être de différents types : du texte, un nombre, une valeur logique, une liste. Un dictionnaire peut donc contenir des données très riches.

D. POURQUOI UTILISER UN DICTIONNAIRE ?

Les dictionnaires sont extrêmement rapides et présents dans de nombreuses applications.

Exemples :

- Un traducteur :

Clé	Valeur
Bonjour	Hello
Merci	Thank you
Oui	Yes

- Une boutique :

Clé	Valeur
Clavier	39 €
Ecran	229 €
Souris	12 €

- Une météo :

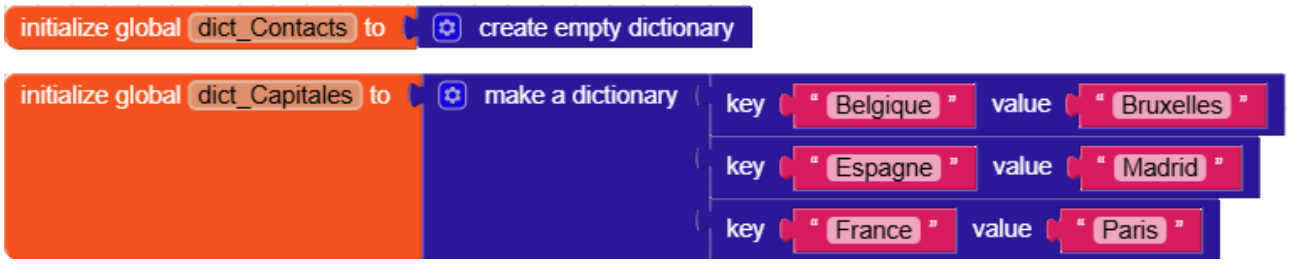
Clé	Valeur
Spa	22°C
Paris	25°C
Madrid	34°C

E. CREER UN DICTIONNAIRE DANS KODULAR

Kodular propose une catégorie dédiée aux dictionnaires.

Tu peux créer :

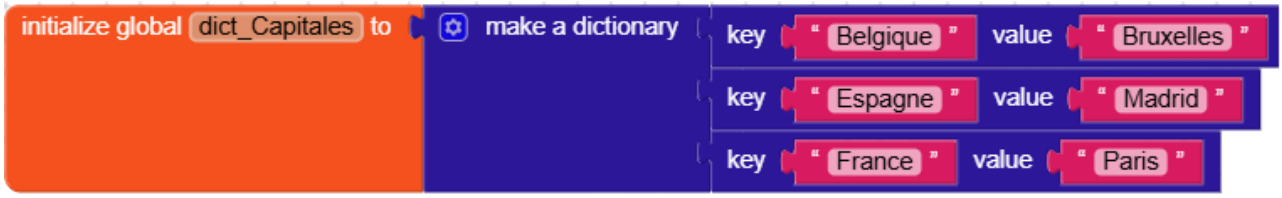
- Un dictionnaire vide ;
- Un dictionnaire contenant déjà plusieurs clés et valeurs.

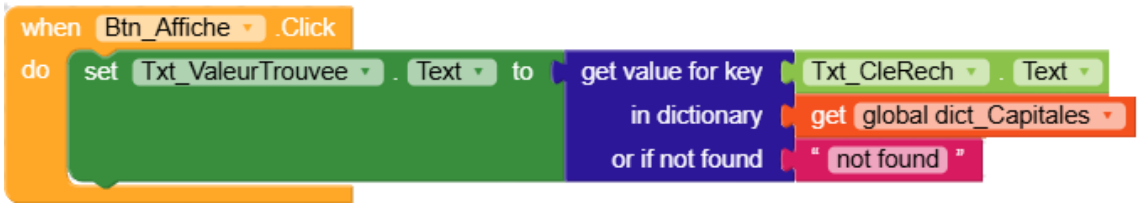


Le dictionnaire est immédiatement prêt à être utilisé.

F. RECHERCHER UNE VALEUR

Le principal avantage d'un dictionnaire est la rapidité des recherches.





Remarque que la commande de recherche **Get Value** permet d'indiquer ce que l'application doit faire si la clé recherchée n'existe pas dans le dictionnaire. Le programme doit-il renvoyer une valeur vide, une valeur par défaut ou afficher un message d'erreur ?

G. AJOUTER UNE NOUVELLE ENTREE

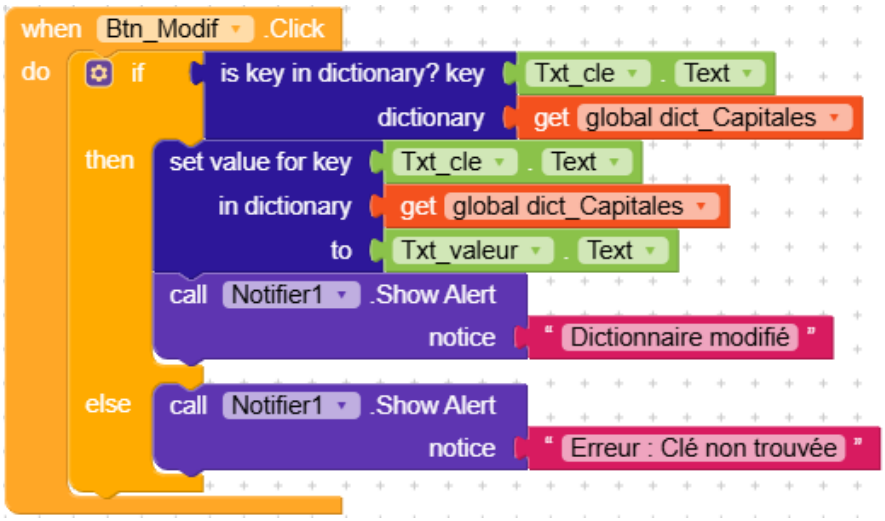
On peut facilement ajouter une nouvelle entrée dans un dictionnaire.

Si nous prenons l'exemple du dictionnaire précédent et souhaitons lui ajouter l'entrée Japon / Tokyo, nous procéderions de la manière suivante :



H. MODIFIER UNE VALEUR

Si l'on souhaite modifier la valeur associée à une clé, on utilisera les commandes suivantes en prenant soin de bien vérifier que la clé est présente dans le dictionnaire avant de procéder au changement demandé.



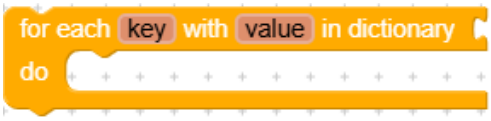
I. SUPPRIMER UNE ENTREE

Pour supprimer une entrée (et sa valeur), on procédera comme suit :



J. PARCOURIR UN DICTIONNAIRE

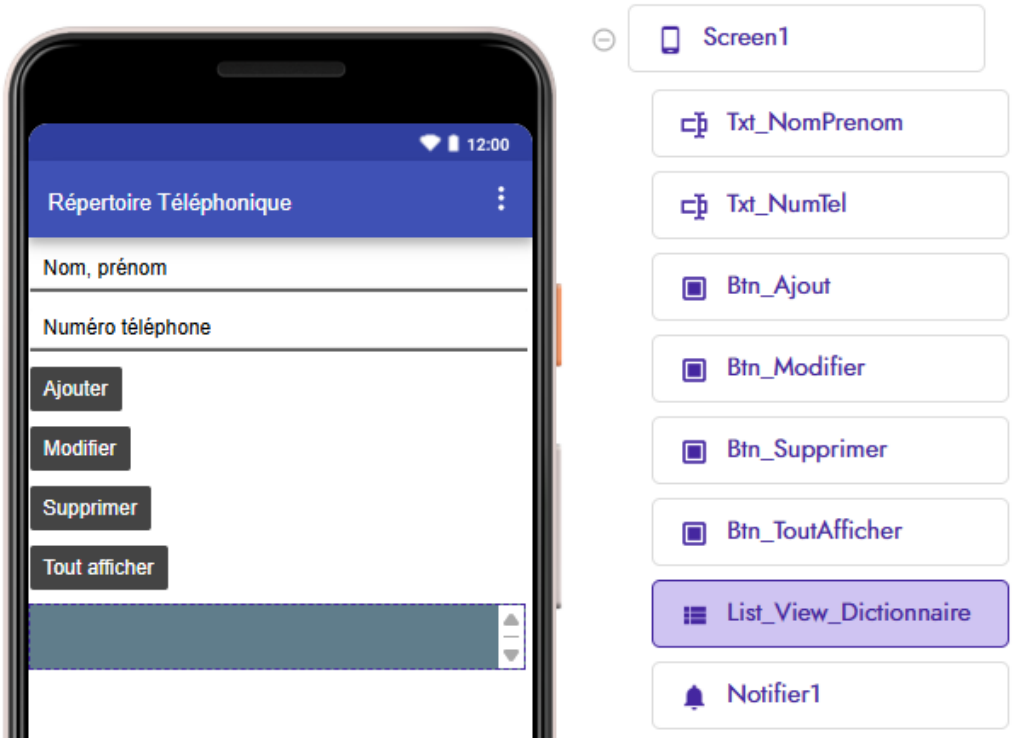
Comme les listes, les dictionnaires peuvent être parcourus. Pour ce faire, il faut utiliser la commande :



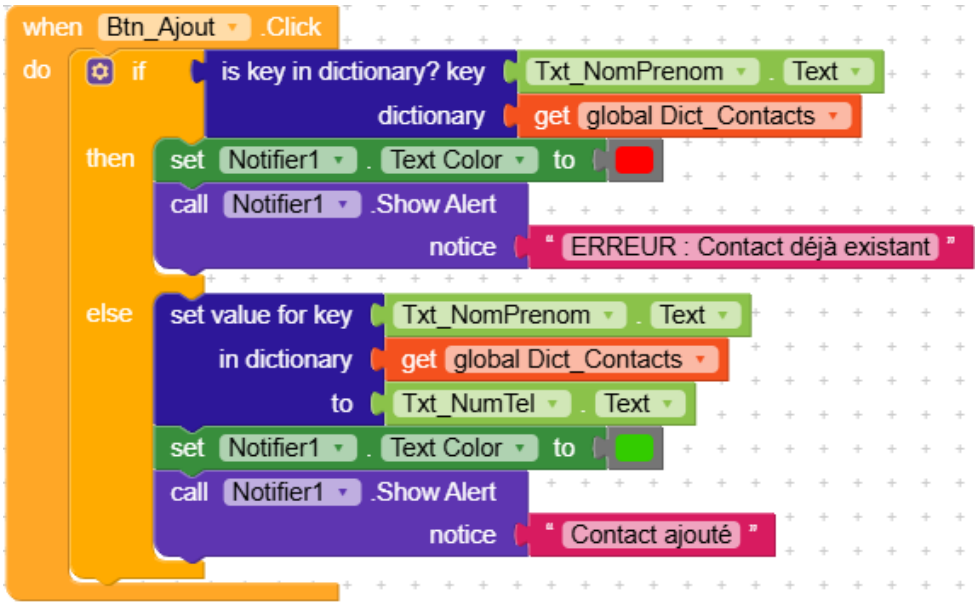
K. EXERCICE : REPERTOIRE TELEPHONIQUE

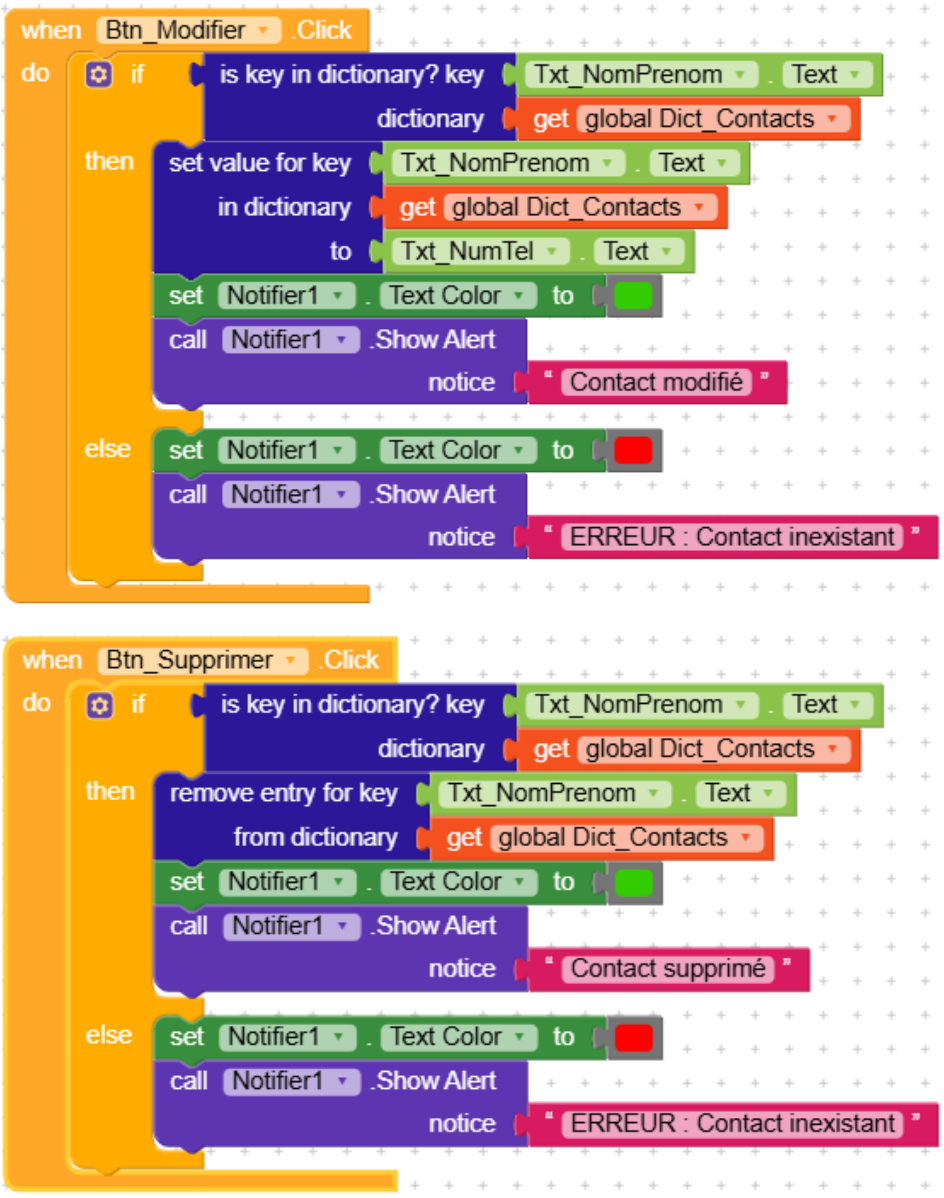
Crée un dictionnaire contenant tes contacts téléphoniques. La clé est le nom de ton contact et la valeur son numéro de téléphone. Ajoute ensuite les boutons (et les traitements associés) : *Ajouter*, *Modifier*, *Supprimer*, *Tout afficher*.

N'oublie pas de faire en sorte que l'écran puisse défiler (scrollable).


12:00
Répertoire Téléphonique
Nom, prénom
Numéro téléphone
Ajouter
Modifier
Supprimer
Tout afficher
Screen1
Txt_NomPrenom
Txt_NumTel
Btn_Ajout
Btn_Modifier
Btn_Supprimer
Btn_ToutAfficher
List_View_Dictionnaire
Notifier1


initialize global Dict_Contacts to
make a dictionary
key " Hofmans, Pierre " value " +32 2 123 45 78 "
key " Nowé, Catherine " value " +32 486 12 34 56 "
when Btn_ToutAfficher .Click
do set List_View_Dictionnaire . Elements to get global Dict_Contacts


when Btn_Ajout .Click
do if is key in dictionary? key Txt_NomPrenom . Text
dictionary get global Dict_Contacts
then set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice " ERREUR : Contact déjà existant "
else set value for key Txt_NomPrenom . Text
in dictionary get global Dict_Contacts
to Txt_NumTel . Text
set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice " Contact ajouté "



L. ASTUCES DE DEVELOPPEUR

- Un dictionnaire peut contenir une **liste** comme valeur. Ce qui est d'ailleurs également le cas pour une liste.
  Ainsi, on pourrait faire en sorte d'avoir en plus du numéro de téléphone, l'adresse et l'adresse mail de chaque contact. Cela permet de créer une véritable fiche de contact. Cette technique est très utilisée dans les applications professionnelles.

- Utiliser un **dictionnaire** alors qu'une **liste** suffit. Si les données sont simplement ordonnées, une **liste** reste souvent plus adaptée.

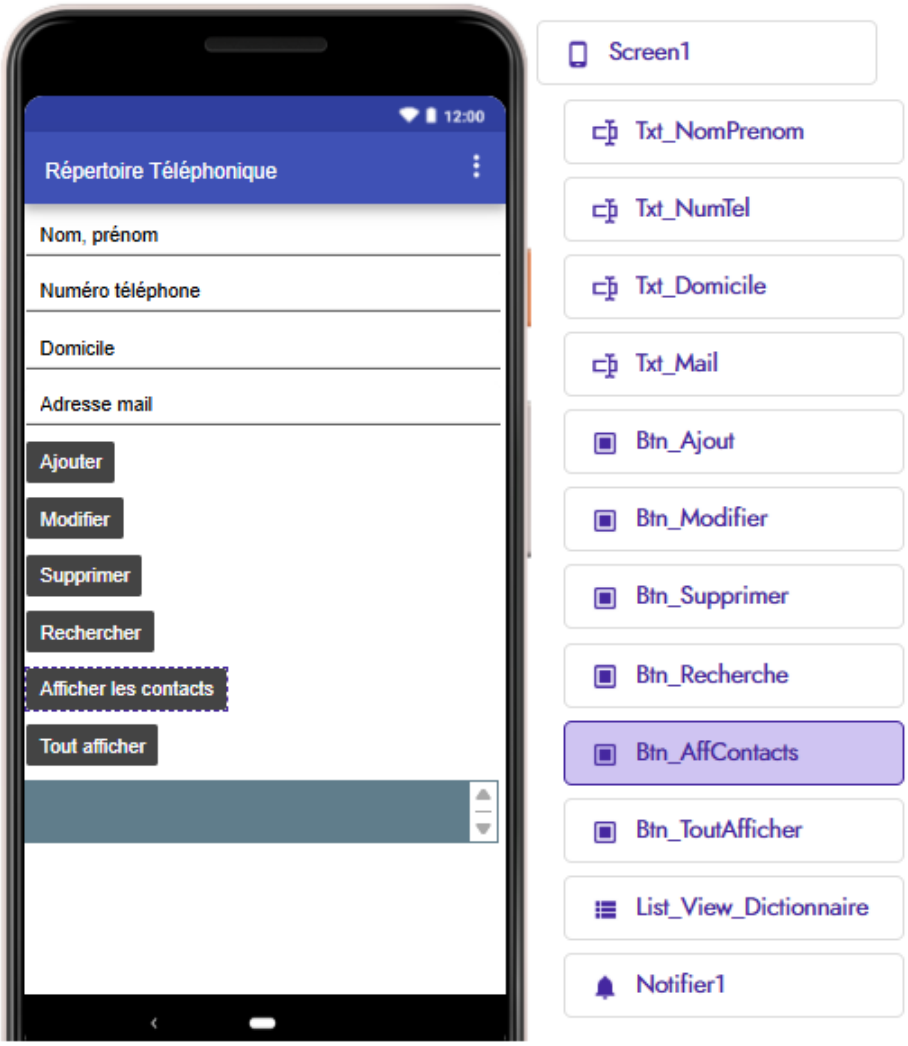
M. EXERCICE : ANNUAIRE INTELLIGENT

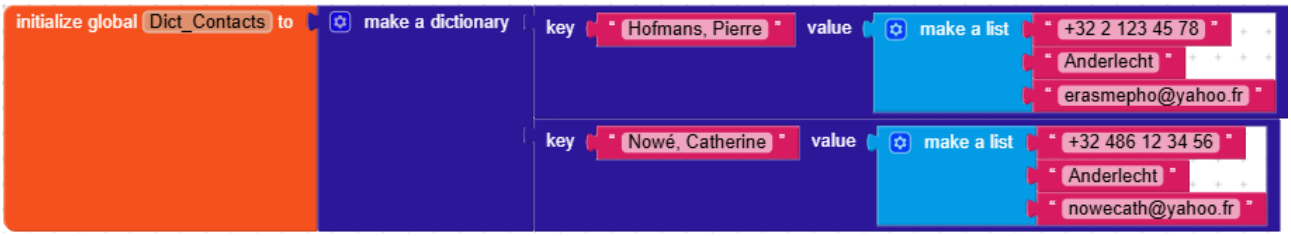
Modifie l'application précédente, à savoir, le *Répertoire téléphonique* afin d'ajouter les valeurs *Domicile* et *Adresse e-mail* pour chaque contact.

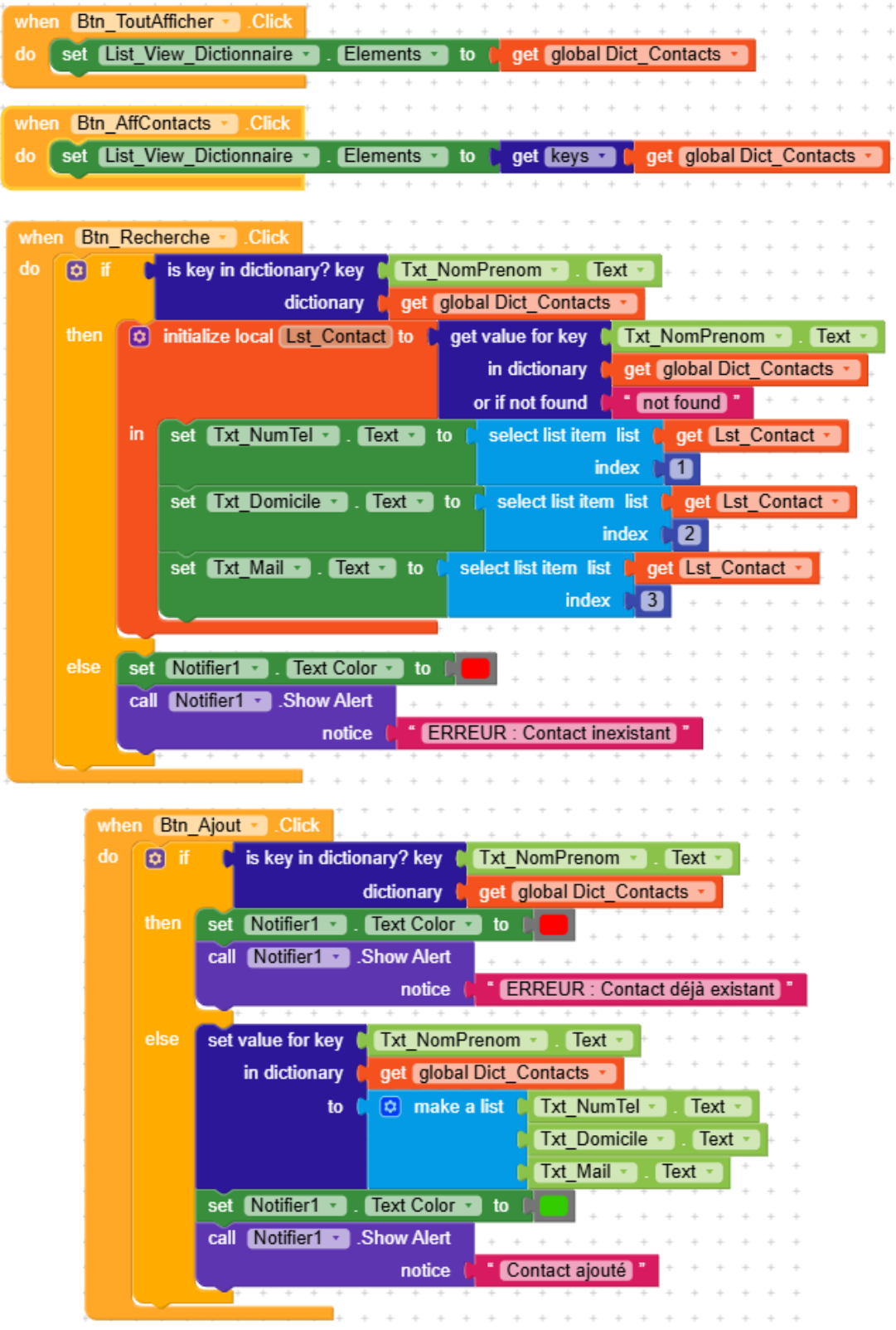
Ajoute également le bouton *Rechercher* qui affiche toutes les informations disponibles du contact dont le nom a été encodé.

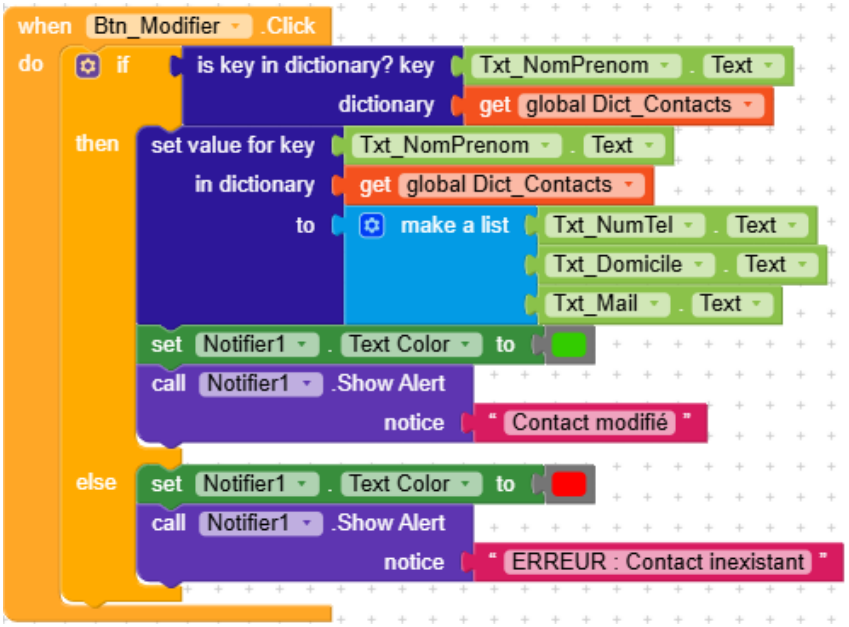
Ce projet met en pratique les principales opérations sur les dictionnaires et montre leur intérêt pour organiser efficacement des données liées entre elles.

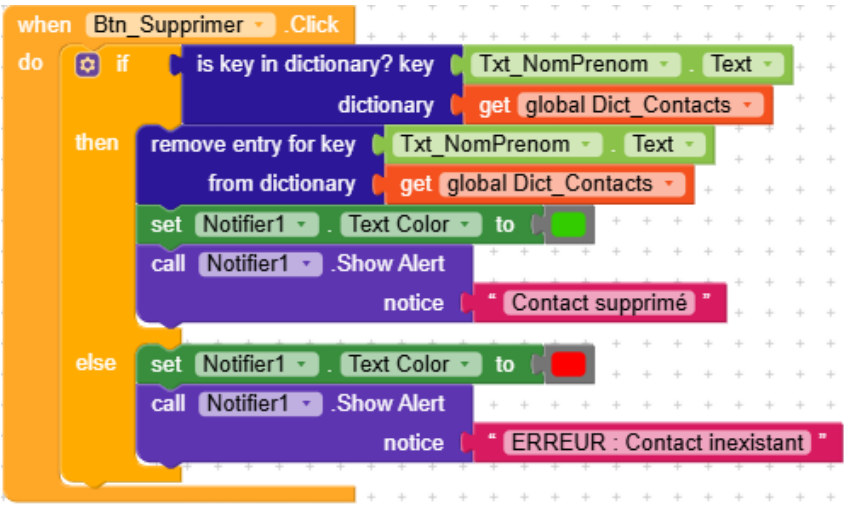
Exercice de dépassement :
Ajoute un bouton *Afficher les contacts* qui affiche dans la **List View** uniquement les noms et prénoms des contacts ; contrairement au bouton *Tout afficher* qui affiche toutes les infos disponibles.






when Btn_ToutAfficher .Click
do set List_View_Dictionnaire . Elements to get global Dict_Contacts
when Btn_AffContacts .Click
do set List_View_Dictionnaire . Elements to get keys get global Dict_Contacts
when Btn_Recherche .Click
do if is key in dictionary? key Txt_NomPrenom . Text
dictionary get global Dict_Contacts
then initialize local Lst_Contact to get value for key Txt_NomPrenom . Text
in dictionary get global Dict_Contacts
or if not found " not found "
in set Txt_NumTel . Text to select list item list get Lst_Contact
index 1
set Txt_Domicile . Text to select list item list get Lst_Contact
index 2
set Txt_Mail . Text to select list item list get Lst_Contact
index 3
else set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice " ERREUR : Contact inexistant "
when Btn_Ajout .Click
do if is key in dictionary? key Txt_NomPrenom . Text
dictionary get global Dict_Contacts
then set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice " ERREUR : Contact déjà existant "
else set value for key Txt_NomPrenom . Text
in dictionary get global Dict_Contacts
to make a list Txt_NumTel . Text
Txt_Domicile . Text
Txt_Mail . Text
set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice " Contact ajouté "



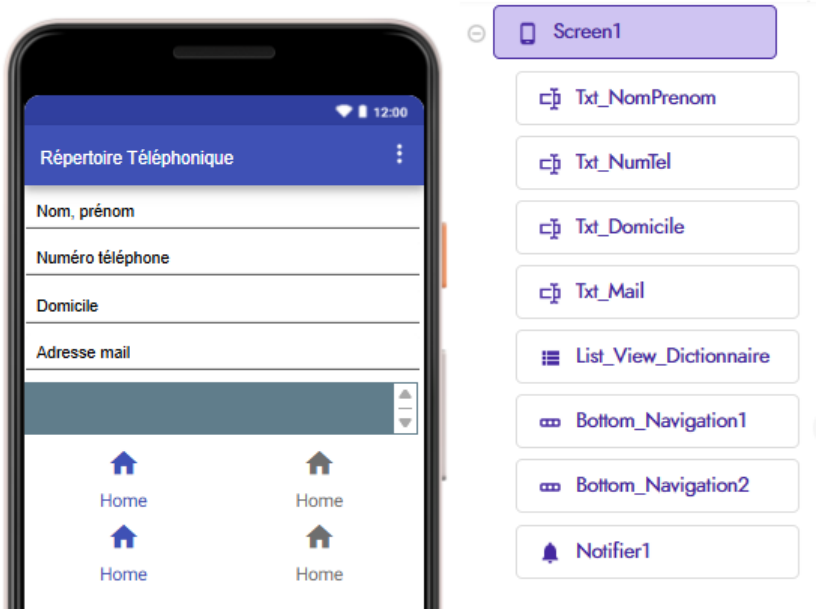


N. REMPLACER TOUS LES BOUTONS PAR UN MENU DANS LE BAS DE L'ECRAN

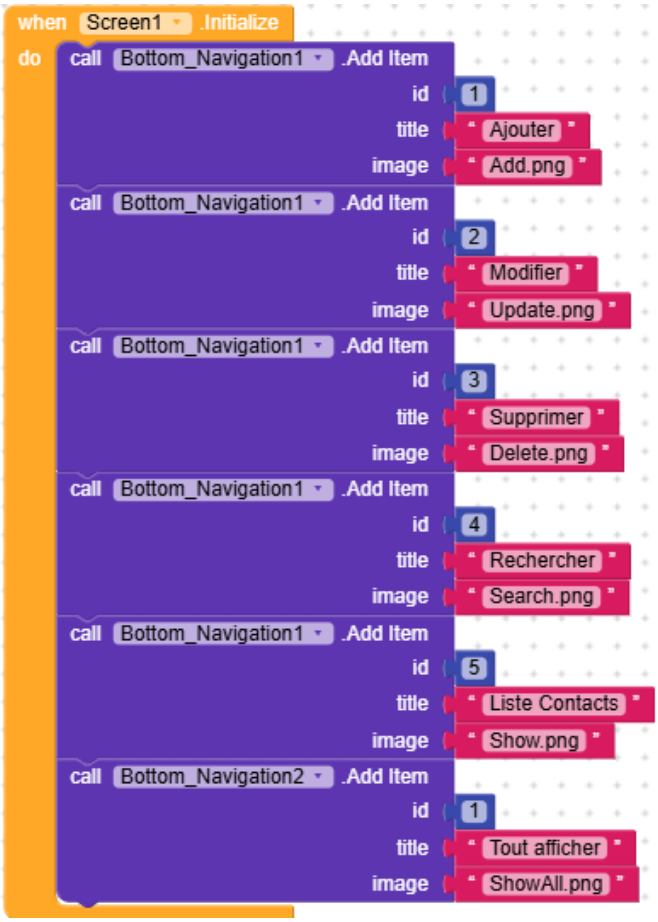
Lorsqu'on a beaucoup de boutons liés aux commandes disponibles, il est parfois préférable, pour une question esthétique, de les regrouper dans une barre de menu. Dans le cas présent, je propose d'utiliser le **Layout** / **Navigation** / **Bottom Navigation** qui permet de regrouper jusqu'à 5 boutons sur une ligne.

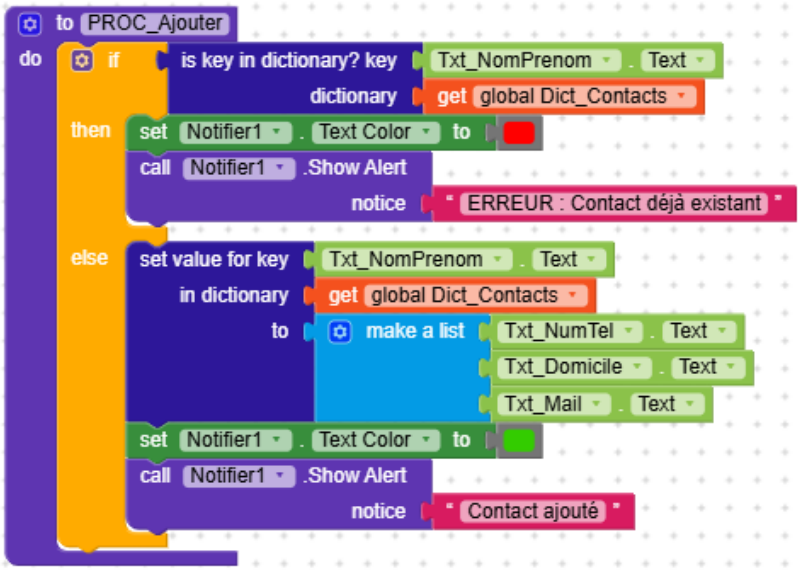
Puisqu'on a 6 boutons, remplaçons-les par 2 **Bottom Navigation**. Et améliorons le tout en utilisant des procédures pour chaque opération "complexe".

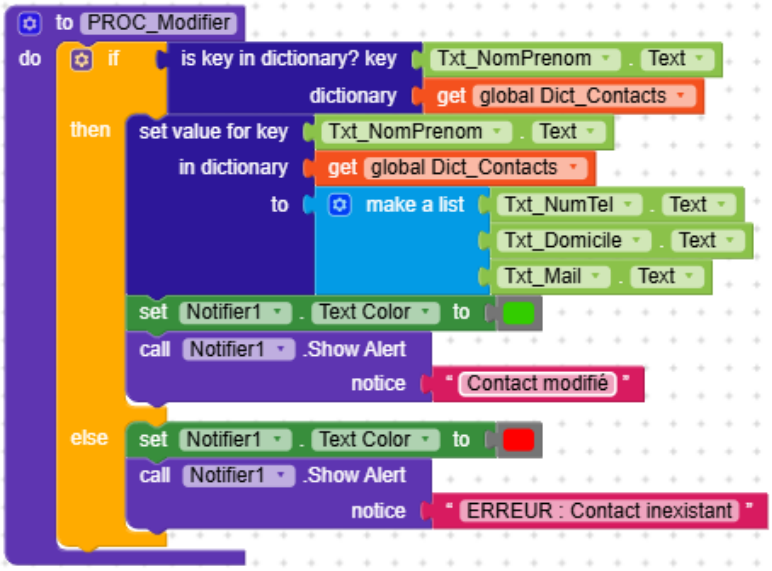
Attention, le système demande d'ajouter des images pour chaque bouton. Mais on ne peut pas ajouter n'importe quelle image car le système (Kodular ?) ne les traite pas correctement. Il faut des images sous le format .PNG transparent, de maximum 96x96 pixels. De plus, pour pouvoir les utiliser, il faut d'abord les ajouter dans **Designer** / **Assets**. On peut trouver des images sur le site https://fonts.google.com/icons?icon.size=40&icon.color=%231f1f1f&icon.platform=web

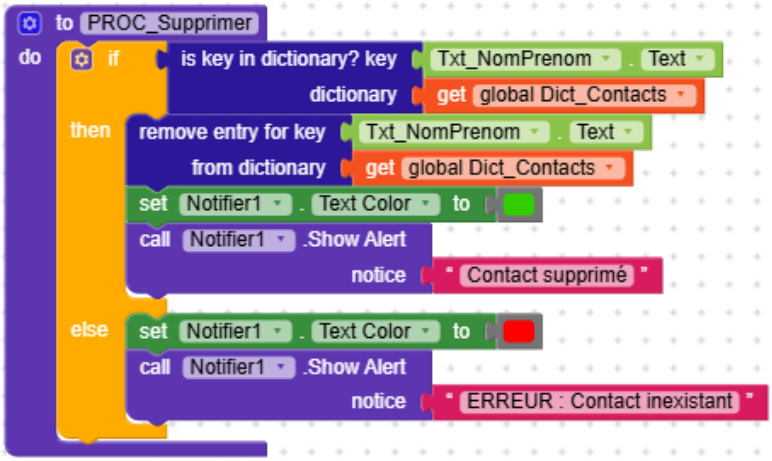

12:00
Répertoire Téléphonique
Nom, prénom
Numéro téléphone
Domicile
Adresse mail
Home
Home
Home
Home
Screen1
Txt_NomPrenom
Txt_NumTel
Txt_Domicile
Txt_Mail
List_View_Dictionnaire
Bottom_Navigation1
Bottom_Navigation2
Notifier1

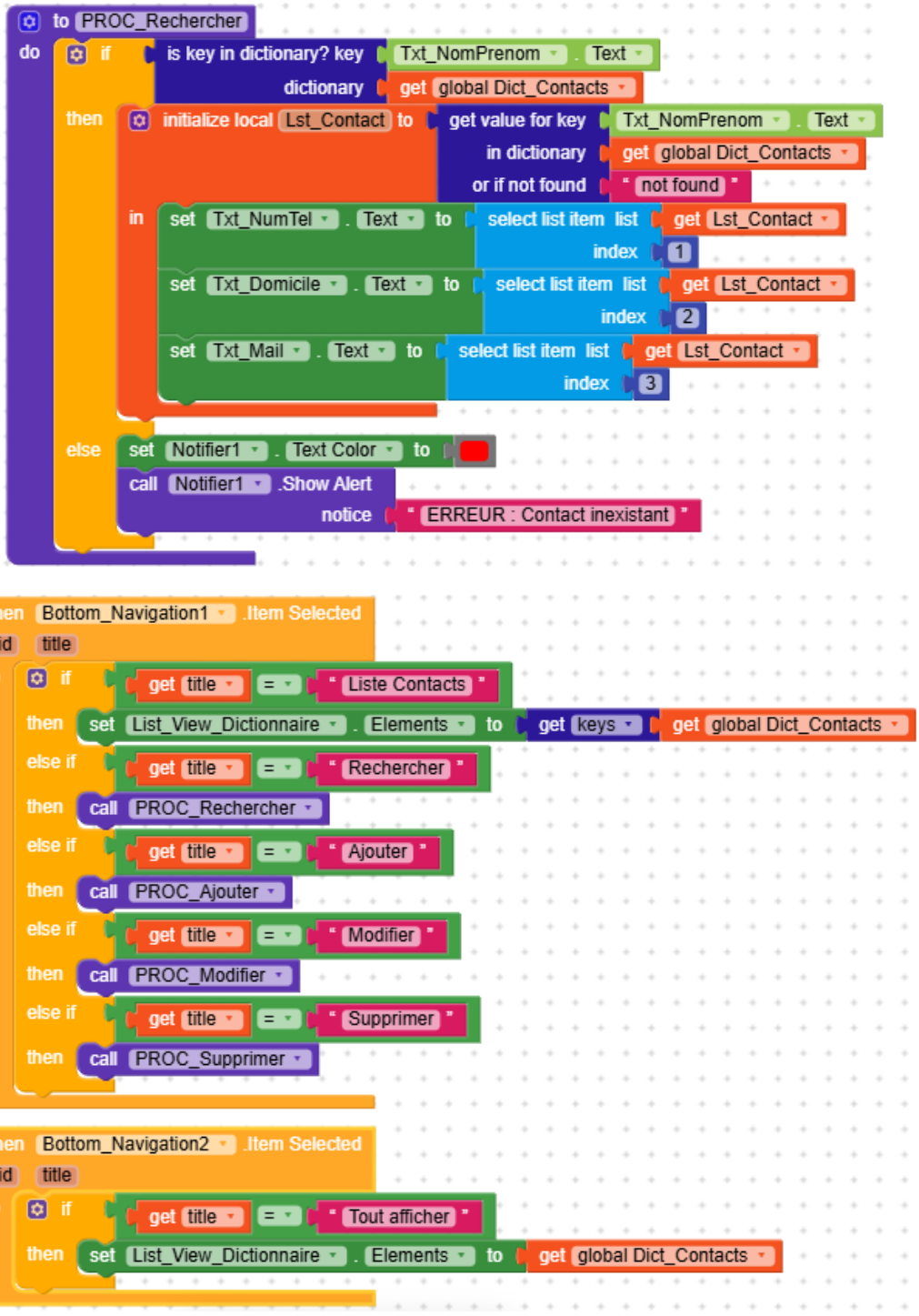

initialize global Dict_Contacts to
make a dictionary
key Hofmans, Pierre value make a list +32 2 123 45 78 Anderlecht erasmepho@yahoo.fr
key Nowé, Catherine value make a list +32 486 12 34 56 Anderlecht nowecath@yahoo.fr


when Screen1 .Initialize
do
call Bottom_Navigation1 .Add Item id 1 title Ajouter image Add.png
call Bottom_Navigation1 .Add Item id 2 title Modifier image Update.png
call Bottom_Navigation1 .Add Item id 3 title Supprimer image Delete.png
call Bottom_Navigation1 .Add Item id 4 title Rechercher image Search.png
call Bottom_Navigation1 .Add Item id 5 title Liste Contacts image Show.png
call Bottom_Navigation2 .Add Item id 1 title Tout afficher image ShowAll.png


to PROC_Ajouter
do
if
is key in dictionary? key
Txt_NomPrenom . Text
dictionary
get global Dict_Contacts
then
set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice
" ERREUR : Contact déjà existant "
else
set value for key
Txt_NomPrenom . Text
in dictionary
get global Dict_Contacts
to
make a list
Txt_NumTel . Text
Txt_Domicile . Text
Txt_Mail . Text
set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice
" Contact ajouté "

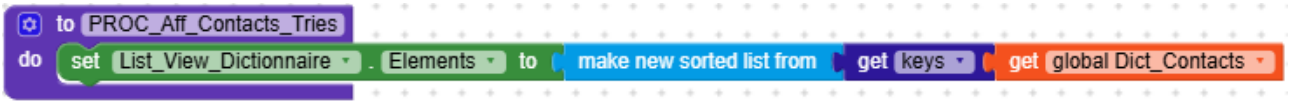

to PROC_Modifier
do
if
is key in dictionary? key
Txt_NomPrenom . Text
dictionary
get global Dict_Contacts
then
set value for key
Txt_NomPrenom . Text
in dictionary
get global Dict_Contacts
to
make a list
Txt_NumTel . Text
Txt_Domicile . Text
Txt_Mail . Text
set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice
" Contact modifié "
else
set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice
" ERREUR : Contact inexistant "

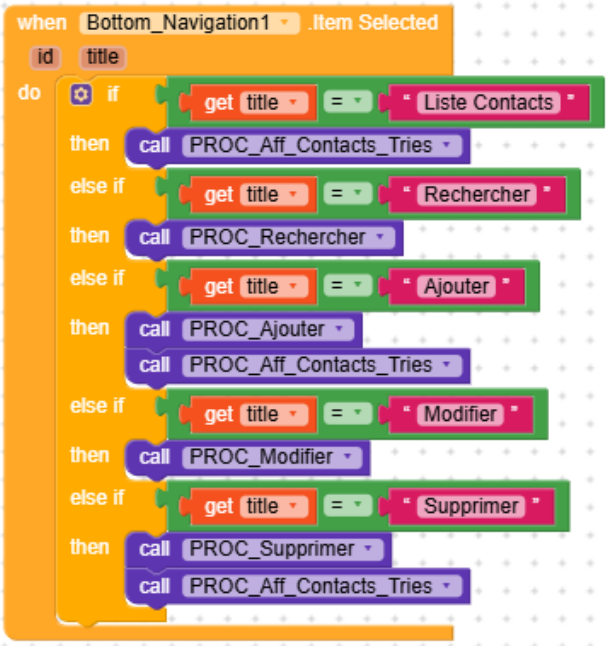

to PROC_Supprimer
do
if
is key in dictionary? key
Txt_NomPrenom . Text
dictionary
get global Dict_Contacts
then
remove entry for key
Txt_NomPrenom . Text
from dictionary
get global Dict_Contacts
set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice
" Contact supprimé "
else
set Notifier1 . Text Color to
call Notifier1 .Show Alert
notice
" ERREUR : Contact inexistant "



O. AJOUTER UN TRI ALPHABETIQUE

Créer une procédure pour l'affichage de la liste actualisée et triée à chaque bouton cliqué modifiant celle-ci → appel de la procédure plutôt que tout réécrire à chaque fois

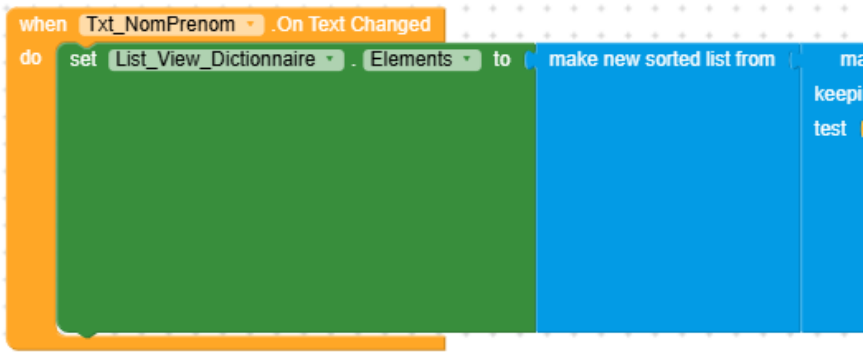


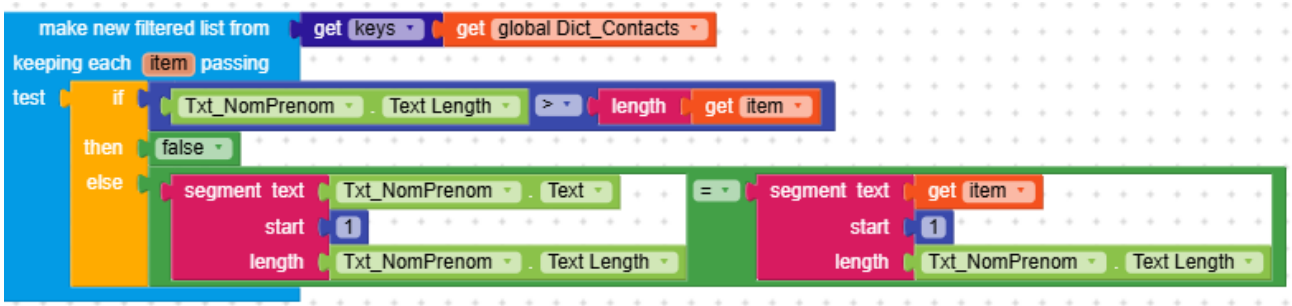


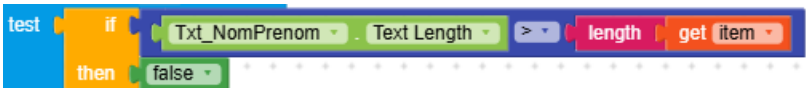
P. EXERCICE DE DEPASSEMENT : AJOUTER UNE RECHERCHE INSTANTANEE

Il faudrait modifier l'application pour qu'à chaque lettre saisie dans la zone Nom, la List View affiche uniquement les noms qui commencent par les caractères déjà introduits dans la zone.

Cette fonctionnalité est présente dans la plupart des applications professionnelles.





Le test 

a dû être ajouté car, sans lui, la commande Segment text de l'item provoquait une erreur dès que

la longueur de la clé (= le nom dans le dictionnaire = l'item) était inférieure à la longueur du nom introduit sur le smartphone.

Q. SAUVEGARDER LES CONTACTS

Pour le moment, les contacts disparaissent lorsque l'application est fermée.

Dans les prochains chapitres, tu découvriras le stockage des données. Tu apprendras à utiliser **TinyDB**, les fichiers et **SQLite** pour conserver les informations même après la fermeture de l'application.

Le gestionnaire de contacts deviendra alors permanent.

Le stockage des données

1. Conserver les donnees avec TinyDB

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre pourquoi il est nécessaire de sauvegarder les données ;
- Découvrir le composant TinyDB de Kodular ;
- Enregistrer des informations sur le téléphone ;
- Récupérer des données au démarrage de l'application ;
- Supprimer des données enregistrées ;
- Connaître les bonnes pratiques pour utiliser TinyDB.

B. INTRODUCTION

Imagine que tu développes une application de liste de courses. Tu ajoutes une liste de produits et tout fonctionne parfaitement. Mais lorsque tu fermes l'application puis que tu la rouvres… La liste est vide. Toutes les informations ont disparu. C'est normal! Jusqu'à présent, nous avons utilisé des variables. Or, une variable ne conserve son contenu que tant que l'application est ouverte. Dès que l'utilisateur quitte l'application, toutes les données sont perdues. Pour créer une véritable application, il faut être capable de mémoriser les informations. C'est précisément le rôle de **TinyDB**.

C. QU'EST-CE QUE TINYDB ?

TinyDB est un composant de stockage local intégré à Kodular. Il permet d'enregistrer des données directement dans la mémoire du téléphone.

Ces données restent disponibles même si :

- L'application est fermée ;
- Le téléphone est redémarré ;
- Plusieurs jours se sont écoulés.

Autrement dit, **TinyDB** agit comme une petite base de données personnelle propre à ton application.

D. OU SE TROUVE TINYDB ?

Dans Kodular, **TinyDB** est un composant invisible. Il se trouve dans la catégorie **Storage**.

Pour l'utiliser :

- Ouvre le **Designer**.
- Dans la palette, choisis **Storage**.
- Fais glisser **TinyDB** dans ton projet.

Comme il s'agit d'un composant invisible, il apparaît en bas de l'écran, dans la zone réservée aux composants non visuels.

E. COMMENT FONCTIONNE TINYDB ?

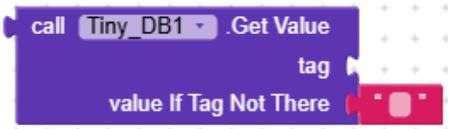
TinyDB enregistre les informations en associant une valeur à une clé. La clé sert à identifier la donnée. La valeur correspond à l'information enregistrée.

Prenons un exemple très simple. L'utilisateur saisit son prénom dans une **Text Box**, puis il appuie sur le bouton *Enregistrer*. Le programme demande à **TinyDB** de mémoriser cette valeur. A partir de ce moment, le prénom est stocké dans le téléphone. Même après la fermeture de l'application, il restera disponible.

La commande à utiliser pour sauver une donnée est :



La commande pour lire une donnée est :



Lors du démarrage de l'application, tu peux récupérer les informations enregistrées. Par exemple, le nom de l'utilisateur ; ce qui te permettra d'afficher directement le message *Bonjour, Pierre*. L'utilisateur retrouve immédiatement ses informations.

F. QUE SE PASSE-T-IL SI LA CLE N'EXISTE PAS ?

Imaginons que tu recherches *CouleurPréférée*, mais qu'aucune donnée n'a encore été enregistrée. **TinyDB** renvoie alors une *valeur par défaut*. Par exemple *Inconnue* ou *Aucune donnée*. Cette valeur est choisie par le développeur.

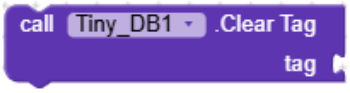
Pourquoi utiliser une valeur par défaut ?
Elle évite que l'application affiche des informations vides ou provoque une erreur. Par exemple, *Bienvenue, invité !* au lieu d'un texte incomplet.

G. MODIFIER UNE DONNEE

TinyDB ne possède pas de commande spéciale pour modifier une valeur, il suffit simplement d'enregistrer une nouvelle valeur avec la même clé. L'ancienne valeur est automatiquement remplacée.

H. SUPPRIMER UNE DONNEE

Si une information n'est plus utile, elle peut être supprimée. Après suppression, la clé n'existe plus. Si le programme tente ensuite de la lire, **TinyDB** renverra la valeur par défaut.



I. QUELS TYPES DE DONNEES PEUT-ON ENREGISTRER ?

TinyDB est très souple.

Il peut stocker :

- Du texte ;
- Des nombres ;
- Des valeurs logiques ;
- Des listes ;
- Des dictionnaires.

J. QUE PEUT-ON SAUVEGARDER ?

Voici quelques exemples d'applications.

- **Une liste de tâches**
  Les tâches à réaliser ;
  Leur état (terminée ou non).

- **Un jeu**
  Le meilleur score ;
  Le niveau atteint ;
  Le nombre de pièces.

- **Une application météo**
  La dernière ville consultée.

- **Une application de recettes**
  Les recettes favorites.

K. EXERCICE : MEMORISER UN PRENOM ET UN SCORE

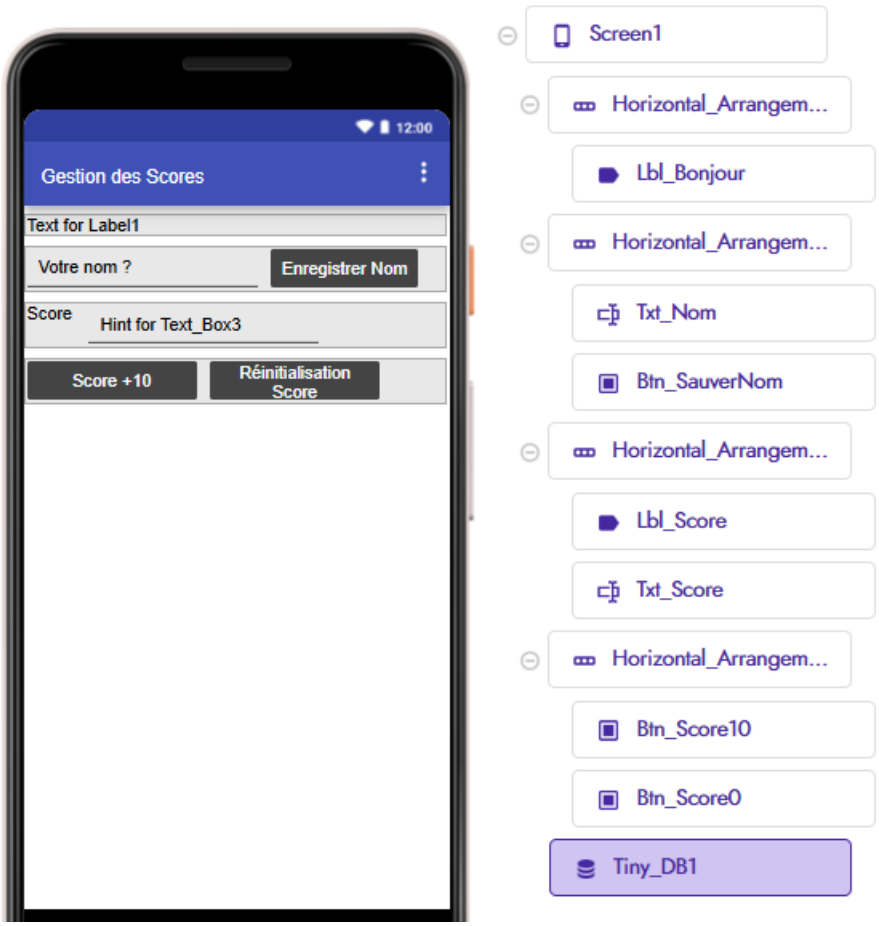
Créer une application qui retient le *Nom* et le *Score* de l'utilisateur.

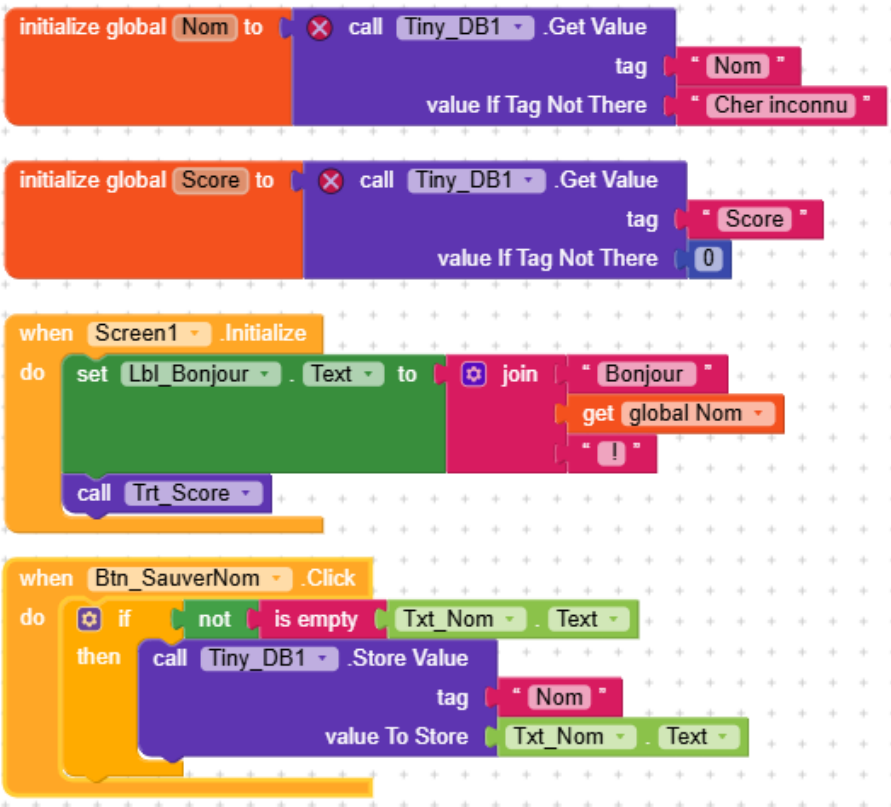
L'écran affichera :

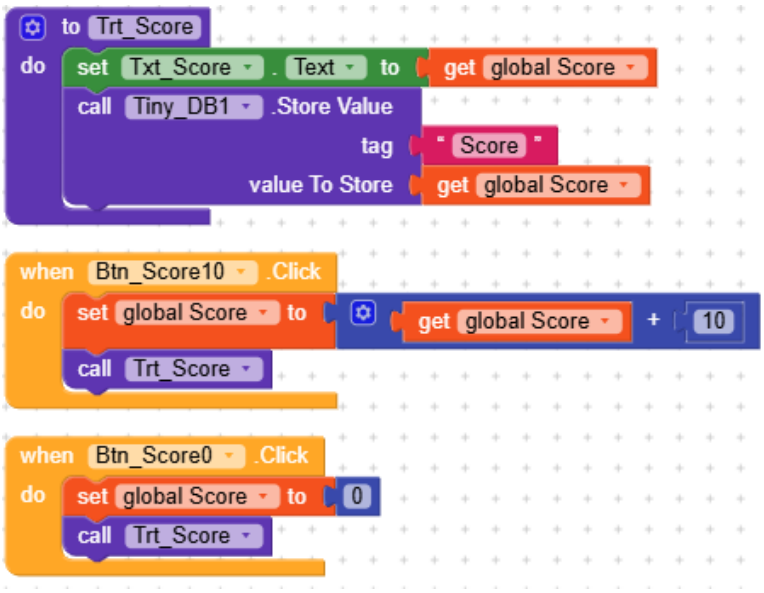
- Un **Label** "Bonjour nom"
- Une **Text Box** *Nom*
- Un **Bouton** *Enregistrer nom*
- Un **Text Box** *Score*
- Un **Bouton** *Score +10*
- Un **Bouton** *Réinitialisation Score*

Lorsque l'utilisateur saisit son prénom et clique sur le bouton de sauvegarde, celui-ci est enregistré dans TinyDB. Au démarrage suivant, le Label affiche automatiquement Bonjour nom !

Le score, lui, est enregistré après chaque modification et est affiché dès le démarrage de l'application avec la valeur précédente. Le bouton de réinitialisation le remet à 0.


12:00
Gestion des Scores
Text for Label1
Votre nom ?
Enregistrer Nom
Score
Hint for Text_Box3
Score +10
Réinitialisation Score
Screen1
Horizontal_Arrangem...
Lbl_Bonjour
Horizontal_Arrangem...
Txt_Nom
Btn_SauverNom
Horizontal_Arrangem...
Lbl_Score
Txt_Score
Horizontal_Arrangem...
Btn_Score10
Btn_Score0
Tiny_DB1


initialize global Nom to call Tiny_DB1 .Get Value
tag "Nom"
value If Tag Not There "Cher inconnu"
initialize global Score to call Tiny_DB1 .Get Value
tag "Score"
value If Tag Not There 0
when Screen1 .Initialize
do set Lbl_Bonjour . Text to join "Bonjour"
get global Nom
"!"
call Trt_Score
when Btn_SauverNom .Click
do if not is empty Txt_Nom . Text
then call Tiny_DB1 .Store Value
tag "Nom"
value To Store Txt_Nom . Text



2. SAUVEGARDER DES LISTES ET DES DICTIONNAIRES AVEC TINYDB

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Enregistrer une liste complète dans **TinyDB** ;
- Sauvegarder un dictionnaire ;
- Restaurer automatiquement des données complexes ;
- Mettre à jour des listes sauvegardées ;
- Utiliser **TinyDB** comme une petite base de données locale.

B. INTRODUCTION

Dans le chapitre précédent, nous avons appris à sauvegarder une seule information. Par exemple, un prénom, une ville ou un score. Mais les applications modernes manipulent généralement beaucoup plus de données.

Prenons quelques exemples :

- Une liste de courses contenant 50 articles ;
- Une liste de contacts ;
- Un agenda ;
- Une collection de recettes ;
- Une bibliothèque de livres.

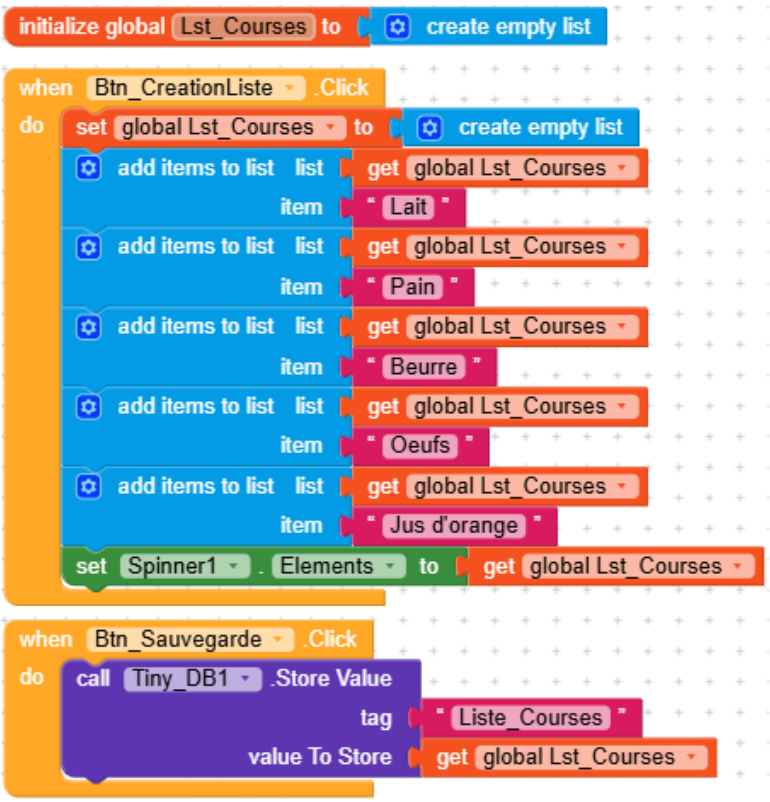
Créer une clé **TinyDB** pour chaque information serait vite compliqué. Heureusement, **TinyDB** peut enregistrer directement des **listes** et des **dictionnaires**. Cette possibilité simplifie énormément le développement.

C. SAUVEGARDER UNE LISTE

Imagine une application de courses.

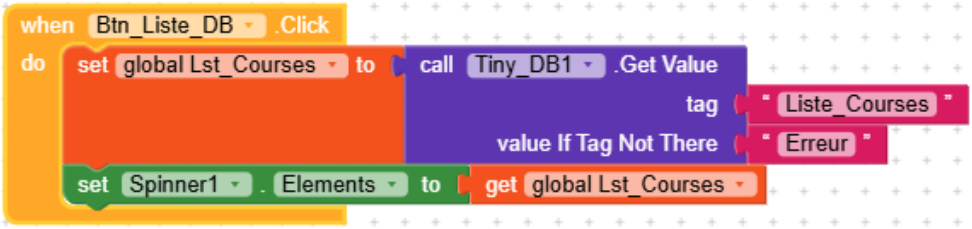
La liste contient : lait, pain, beurre, œufs et jus d'orange. Au lieu d'enregistrer chaque élément séparément, tu peux sauvegarder la liste entière.

La clé serait *Liste_Courses* et la valeur la liste complète. Dans ce cas, une seule opération suffit.



D. RESTAURER UNE LISTE

Lorsque l'application démarre, **TinyDB** restitue immédiatement toute la liste.



E. MODIFIER UNE LISTE SAUVEGARDEE

Une liste enregistrée dans **TinyDB** ne se met pas à jour automatiquement.

Le déroulement est toujours le même :

1. Lire la liste.
2. La modifier (-> Modifier ou supprimer un ou plusieurs éléments).
3. La sauvegarder à nouveau.

Exemple :

1. Liste actuelle : Pain, Beurre
2. Ajout : Lait
3. Nouvelle liste : Pain, Beurre, Lait
4. Il faut ensuite enregistrer cette nouvelle version dans **TinyDB**.

F. SAUVEGARDER UN DICTIONNAIRE

TinyDB peut également enregistrer un dictionnaire entier.

Une seule clé **TinyDB** suffit. Par exemple, la clé serait *NotesClasse* et la valeur le dictionnaire complet. Toutes les données sont conservées ensemble.

G. CHARGER LES DONNEES AU DEMARRAGE

Au lancement de l'application, une bonne pratique consiste à :

- Lire **TinyDB** ;
- Charger les listes ;
- Charger les dictionnaires ;
- Mettre à jour l'affichage.

L'utilisateur retrouve immédiatement ses données.

H. VERIFIER SI UNE SAUVEGARDE EXISTE

Lors du premier lancement de l'application, **TinyDB** est vide. Il faut donc prévoir ce cas.

Par exemple, si la clé *Contacts* n'existe pas, il faut créer automatiquement une liste ou un dictionnaire vide. L'application est ainsi prête à fonctionner.

I. EXERCICE : LISTE DE COURSES PERMANENTE

Créer une application contenant :

- Une **Text Box** ;
- Un bouton *Ajouter* ;
- Une **List View**.

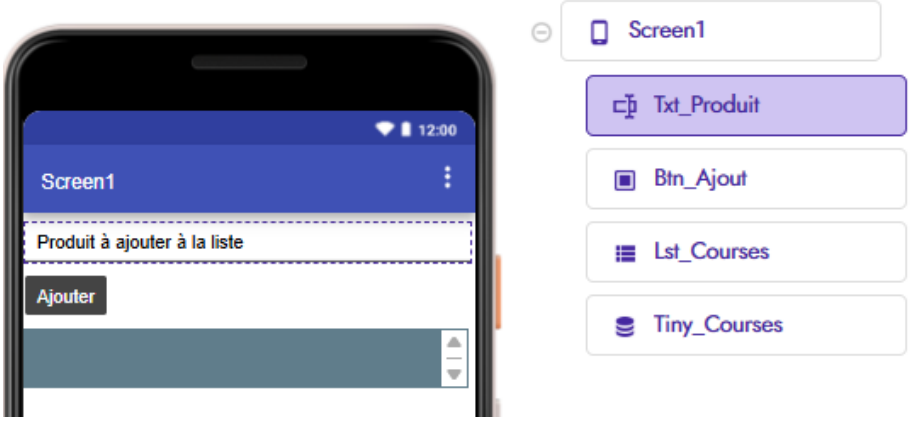
Chaque nouvel article est :

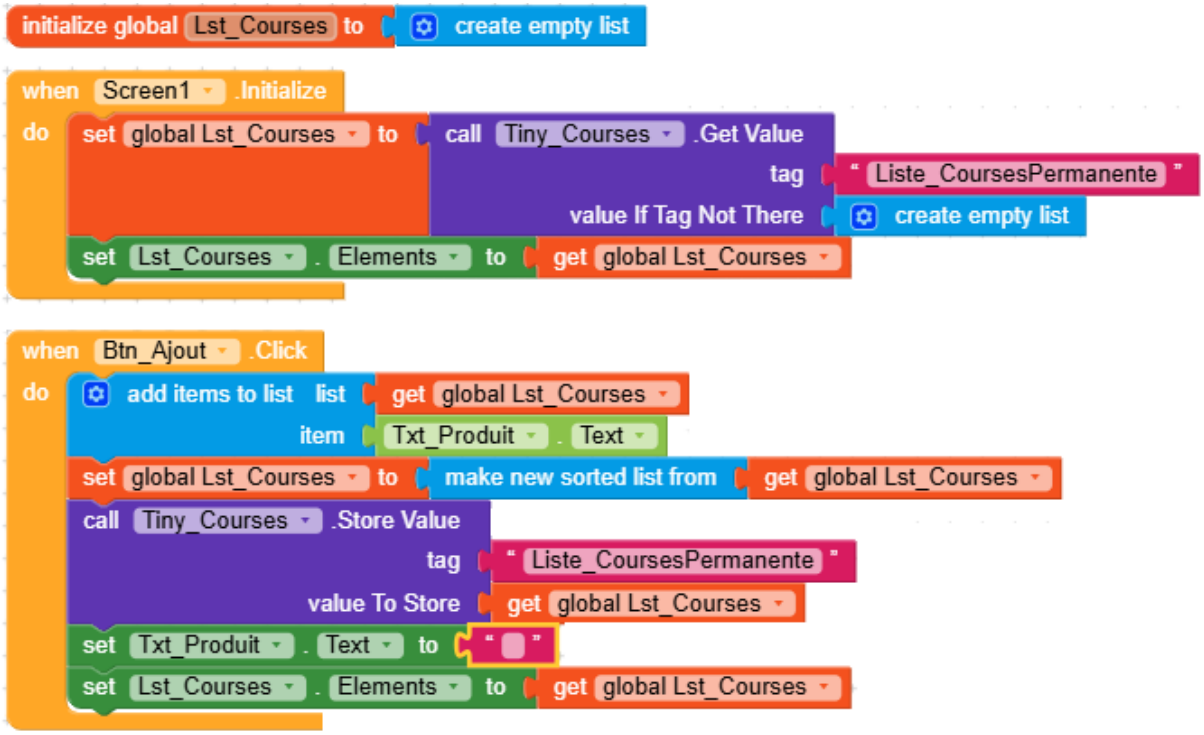
- Ajouté à la liste ;
- Sauvegardé dans **TinyDB**.

Après avoir fermé puis rouvert l'application, la liste est toujours présente.

Il faut penser à :

- Trier la liste à chaque ajout ;
- Vider la **Text Box** après chaque ajout.

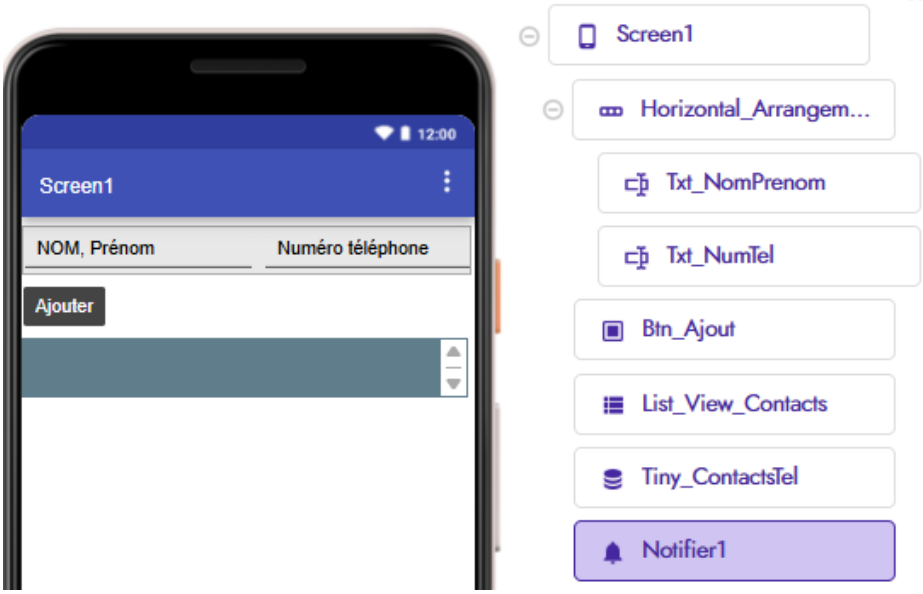


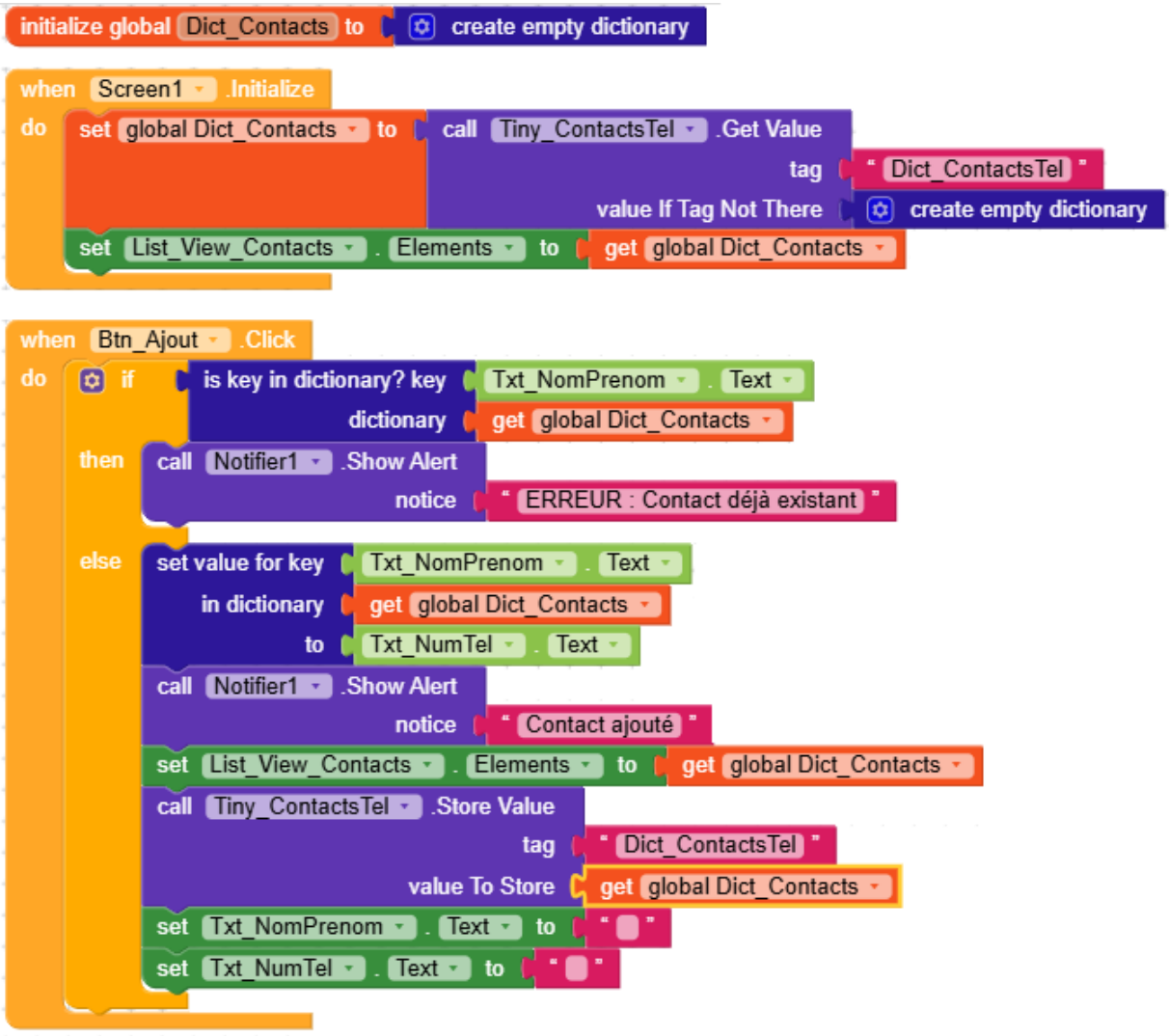


J. EXERCICE : CARNET DE CONTACTS

Faire la même application que ci-précédemment mais cette fois-ci avec un carnet de contacts sauvé dans un dictionnaire (NOM, Prénom & N°tél).

Vérifier que les données restent présentes après la fermeture de l'application.





K. LES ERREURS FREQUENTES

- **Modifier une liste sans la sauvegarder**
  Les modifications restent uniquement en mémoire. Après fermeture de l'application, elles disparaissent.

- **Oublier de charger TinyDB au démarrage**
  Les données existent toujours. Mais elles ne sont pas copiées dans les variables. La **List View** apparaît vide.

- **Sauvegarder plusieurs fois la même information**
  Évite les doublons. Utilise une seule clé par ensemble logique.

L. BONNES PRATIQUES

- Sauvegarder immédiatement après une modification.
- Recharger les données au démarrage.
- Utiliser des clés explicites.
- Regrouper les informations dans des listes ou des dictionnaires.
- Tester toujours le redémarrage de l'application.

3. LES FICHIERS : ENREGISTRER ET LIRE DES DONNEES

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre la différence entre **TinyDB** et les fichiers ;
- Créer un fichier texte avec Kodular ;
- Enregistrer des informations dans un fichier ;
- Lire le contenu d'un fichier ;
- Modifier un fichier existant ;
- Connaître les bonnes pratiques pour gérer les fichiers dans une application Android.

B. INTRODUCTION

Précédemment, nous avons utilisé **TinyDB** pour sauvegarder des informations. **TinyDB** est parfait pour conserver les données internes d'une application.

Mais imaginons maintenant que tu souhaites :

- Exporter une liste de contacts ;
- Sauvegarder un rapport ;
- Créer un journal d'activités ;
- Enregistrer des notes ;
- Partager un document avec une autre application.

TinyDB n'est plus la meilleure solution. Dans ces situations, il est préférable d'utiliser des fichiers. Les fichiers sont universels. Ils peuvent être copiés, sauvegardés, envoyés par e-mail ou ouverts avec d'autres applications.

Dans cette partie, nous allons découvrir comment les utiliser dans Kodular.

C. QU'EST-CE QU'UN FICHIER ?

Un fichier est un document enregistré dans la mémoire d'un appareil. Il possède généralement un nom, un emplacement et un contenu. Chaque ligne correspond à une information.

Tu utilises déjà des fichiers tous les jours tels qu'une photo, une musique, une vidéo, un document Word, un fichier PDF, un fichier texte.

Chaque fichier possède une extension qui indique son type.

Les fichiers permettent de :

- Sauvegarder des informations ;
- Transférer des données ;
- Partager un document ;
- Effectuer une copie de sécurité ;
- Conserver un historique.

Ils sont utilisés dans pratiquement toutes les applications professionnelles.

D. TINYDB OU FICHIER ?

Les deux solutions permettent de sauvegarder des données.

Mais elles ne répondent pas au même besoin.

TinyDB	**Fichier**
Données internes	Données partageables
Invisible pour l'utilisateur	Visible si autorisé
Très rapide	Plus souple
Facile à utiliser	Idéal pour les échanges

E. LE COMPOSANT FILE

Kodular possède un composant invisible nommé **File**. Il se trouve dans la catégorie **Storage**.

Il permet notamment de :

- Créer un fichier ;
- Ecrire dans un fichier ;
- Lire un fichier ;
- Supprimer un fichier.

Selon la configuration choisie, les fichiers peuvent être stockés :

- Dans le dossier privé de l'application ;
- Dans un dossier partagé du téléphone (si les autorisations nécessaires sont accordées).

Pour un débutant, il est recommandé de commencer par le stockage privé de l'application. Il est plus simple et plus sécurisé.

F. CREER, ECRIRE ET LIRE UN FICHIER TEXTE

Avant de créer un fichier, il faut avant tout choisir le nom qu'on va lui donner. Un nom explicite facilite la maintenance. On pourra, par exemple, l'appeler Courses.txt, Contacts.txt, …

Dans Kodular, il n'existe pas vraiment de commande pour créer un fichier vide. En effet, c'est lors de l'écriture de son contenu que tout se passe.

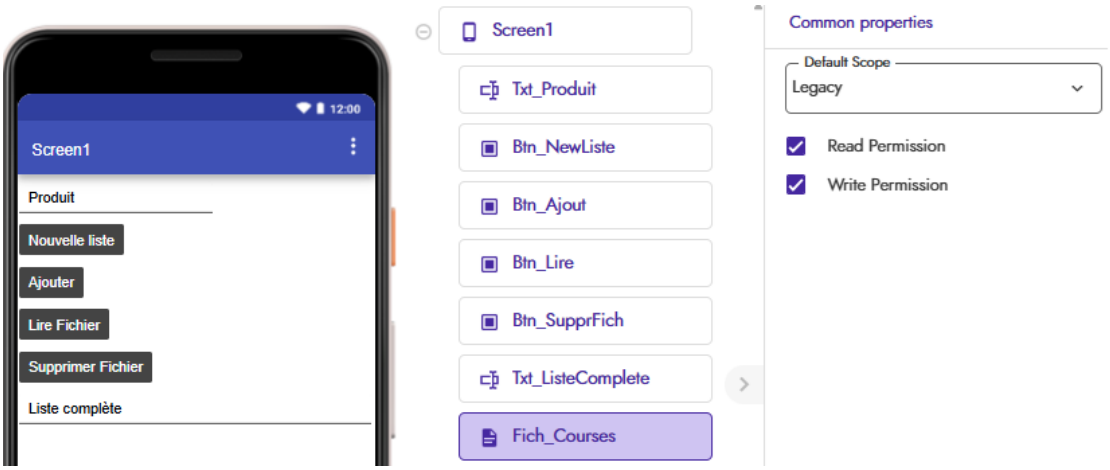
Il existe deux commandes pour écrire dans un fichier. La première crée le fichier, en supprimant le cas échéant un fichier existant ayant le même nom et y place ensuite le texte (éventuellement vide) indiqué. La seconde ajoute le texte donné en argument à la fin d'un fichier texte existant.

Supposons que l'on veuille sauver dans un fichier, la liste des produits que l'on doit acheter la prochaine fois que l'on ira au supermarché.

Pour ce faire, on va créer une application qui a :

- Une zone de texte pour indiquer le produit suivant à acheter
- Les boutons *Nouvelle liste*, *Ajout*, *Lire fichier* et *Supprimer fichier*
- Une zone de texte qui permettra d'afficher le contenu du fichier

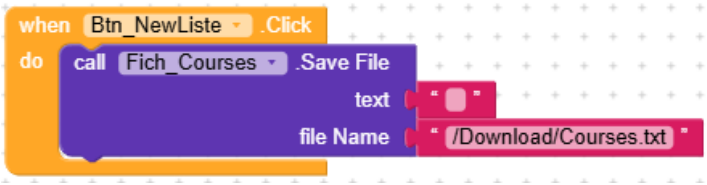
Il faudra aussi faire en sorte que l'écran soit "scrollable" afin de pouvoir voir le contenu de la liste même si elle dépasse la taille de l'écran.



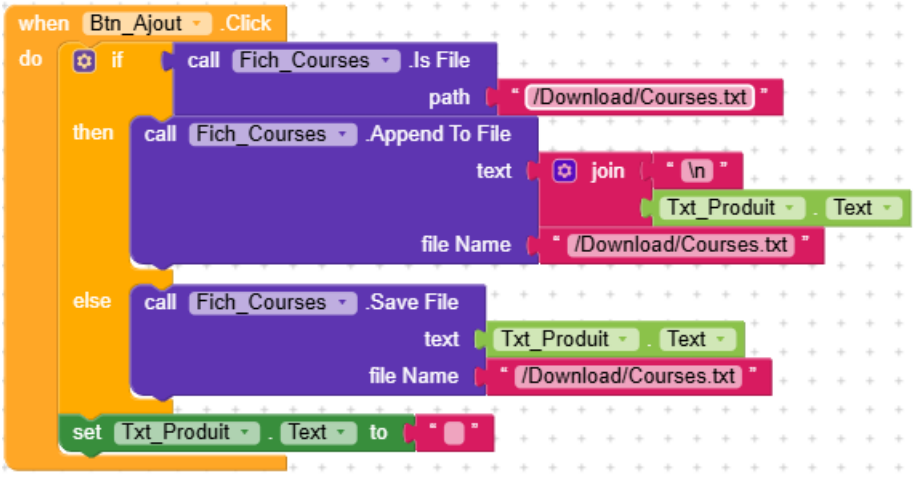
Que peut-on constater dans cet exemple ?

- L'option **Scrollable** du *Screen1* est cochée
- Les options **Multiline** et **Read Only** de la zone *Txt_ListeComplete* sont cochées
- L'option **Default Scope** de *Fich_Courses* (**Storage** / **File**) est à **Shared**. Le scope **Legacy** correspond au stockage partagé, accessible en dehors du stockage privé de ton application. Dans cet exemple, on le sauve dans le dossier *Téléchargements* (*/Downloads*) du smartphone.

Le bouton *Nouvelle liste* permet de vider le fichier. En fait, pour cela, il crée simplement le fichier *Courses.txt* dans le dossier *Download* en y mettant du texte vide :

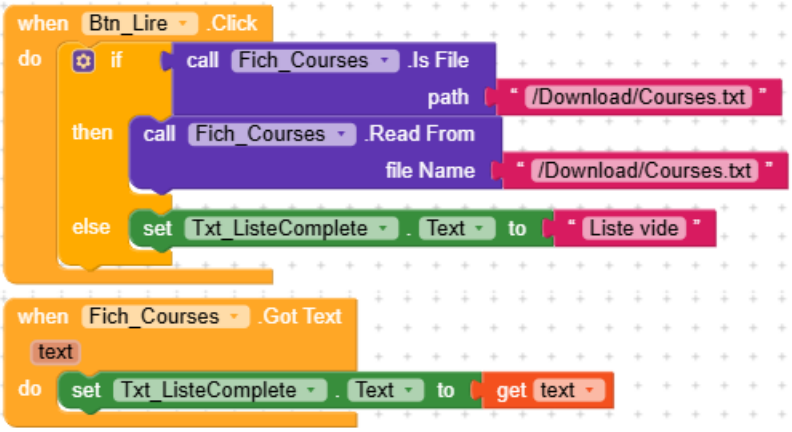


Le bouton *Ajouter* permet d'ajouter le contenu de la zone *Txt_Produit* dans le fichier. Si ce dernier existe déjà, il l'ajoute simplement à la fin du fichier ; dans le cas contraire, il crée le fichier lors de l'ajout.



Remarquons encore que lors de l'ajout à la fin du fichier existant, on fait précéder le texte *Txt_Produit* des caractères **\n**. Ceux-ci correspondent au code *Passage à la ligne* qui permet d'avoir chaque produit sur une ligne différente.

Quant à la lecture et l'affichage du contenu du fichier, cela se fait en deux étapes distinctes. La première étape lit le fichier, charge le contenu en mémoire et déclenche l'événement **Got Text**. Ce dernier permet de récupérer le texte lu et, éventuellement, l'afficher à l'écran.



G. MODIFIER UN FICHIER TEXTE

Si on souhaite modifier le contenu d'un fichier texte, il faut :

- Lire le fichier.
- Modifier les données en mémoire.
- Réécrire le fichier.

H. SUPPRIMER UN FICHIER TEXTE

Lorsque le fichier n'est plus utile, il peut être supprimé au moyen de la commande :



I. LES FICHIERS CSV

Si les fichiers Texte présentent plusieurs avantages tels que leur légèreté, leur compatibilité avec de nombreux logiciels, leur lecture aisée, etc., il existe un autre type de fichiers régulièrement utilisés. Ce sont les fichiers CSV qui signifie **Comma Separated Values** (valeurs séparées par des séparateurs).

Exemples :

- Dupont;Pierre;17
- Martin;Julie;15
- Leroy;Marc;18

Chaque ligne représente une personne et chaque colonne contient une information.

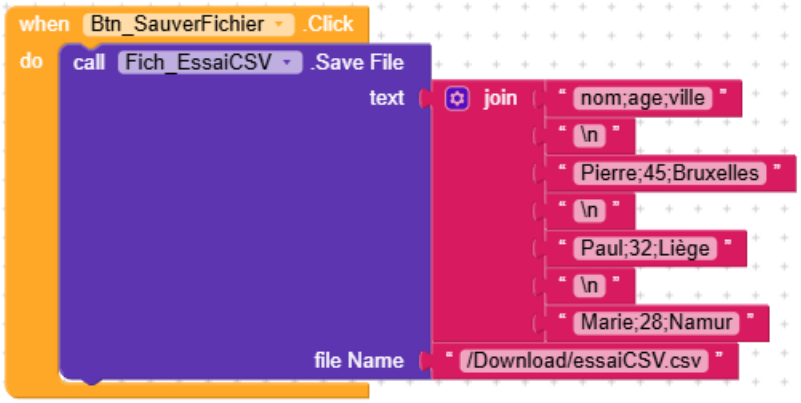
Les fichiers CSV sont largement utilisés pour Excel, LibreOffice Calc et les bases de données. Il est en effet très facile d'importer ces données dans un tableur.

Les commandes utilisées pour les créer, … sont les mêmes que les commandes utilisées pour les fichiers Texte. En effet, un fichier CSV est simplement un fichier Texte dont les valeurs sont séparées par un caractère (généralement ; ou ,) et dont l'extension est CSV. La manipulation de leurs données diffère toutefois.

Supposons que l'on veuille créer un fichier CSV contenant :

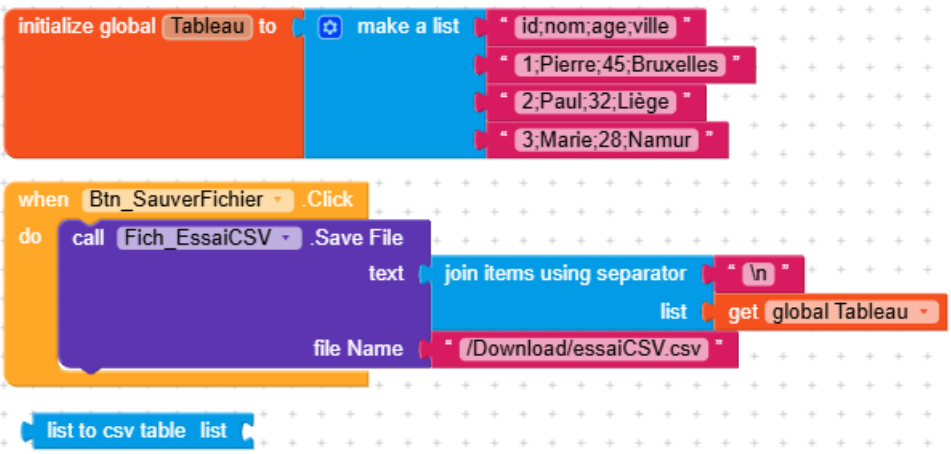
```
nom;age;ville
Pierre;45;Bruxelles
Paul;32;Liège
Marie;28;Namur
```

On peut le faire au moyen de la commande :



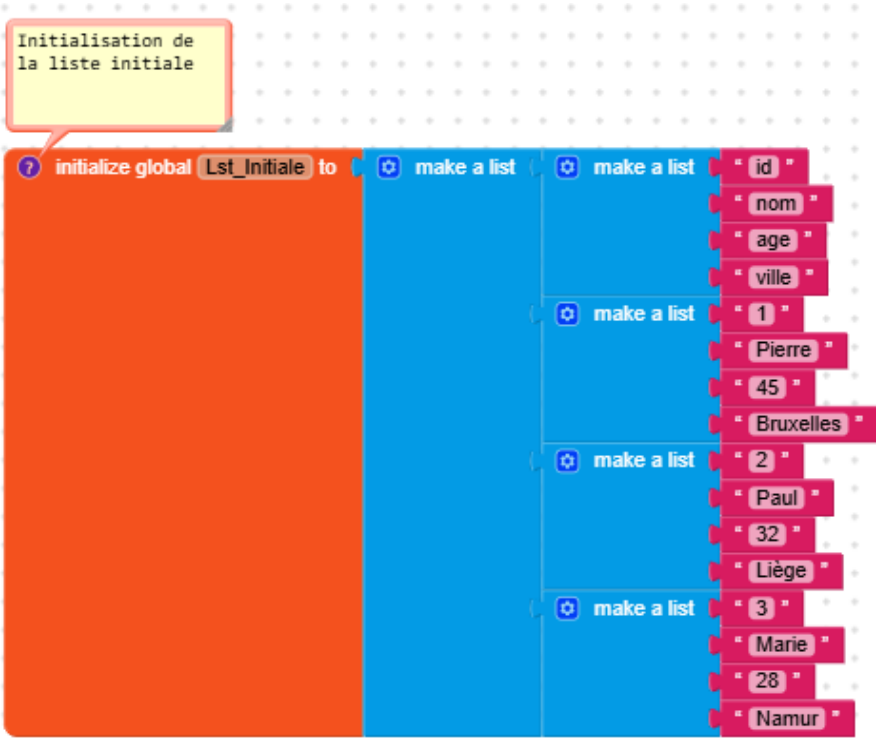
Ce fichier pourra être ouvert avec Excel via la commande **Données / Fichier texte**.

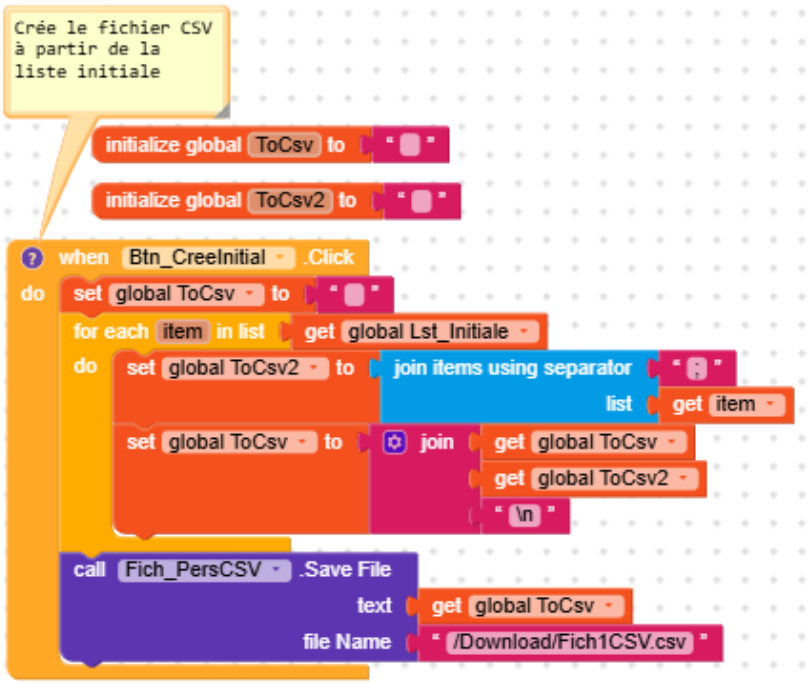
Mais il y a une méthode beaucoup plus intéressante avec les listes :



Attention, il ne faut surtout pas utiliser la commande **List to csv table** car cette commande ajoute des guillemets au début et à la fin de chaque ligne. Ce qui empêche Excel de récupérer correctement les données en tenant compte des séparateurs de champs.

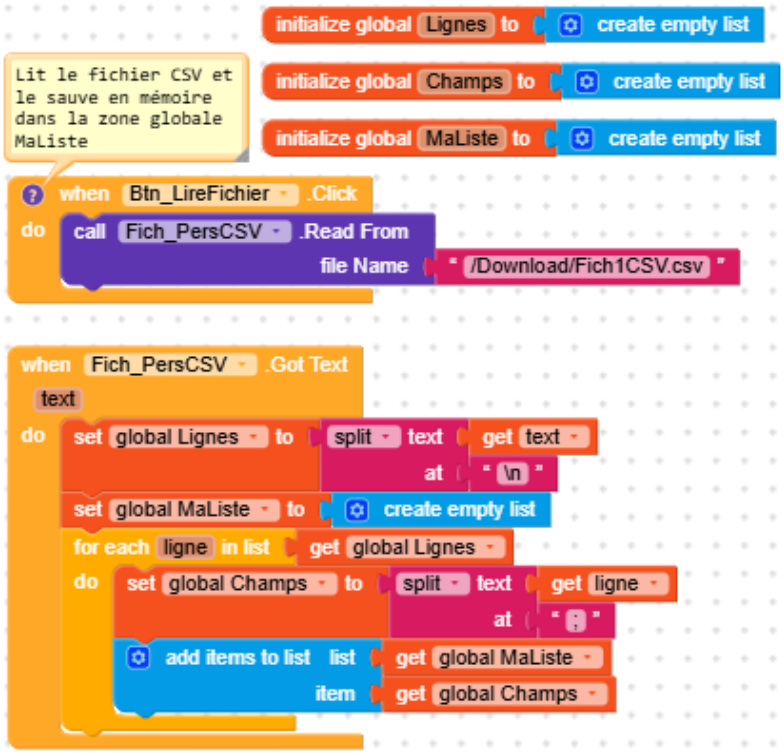
Ou mieux encore, même si plus long, si l'on considère que chaque valeur doit être dans une zone séparée, on peut créer une liste (id, nom, âge, ville) imbriquée dans une liste (personne). Cette façon de faire permettra de pouvoir modifier plus facilement la valeur d'une zone.



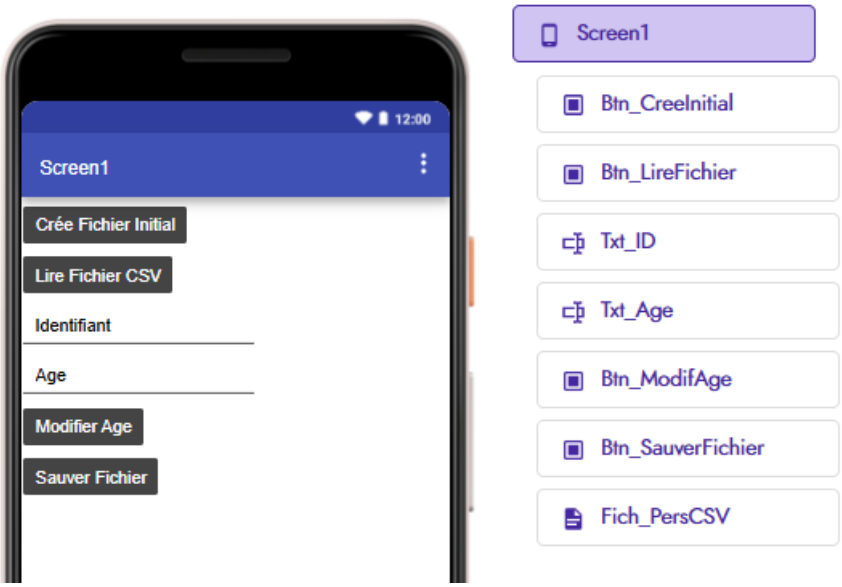


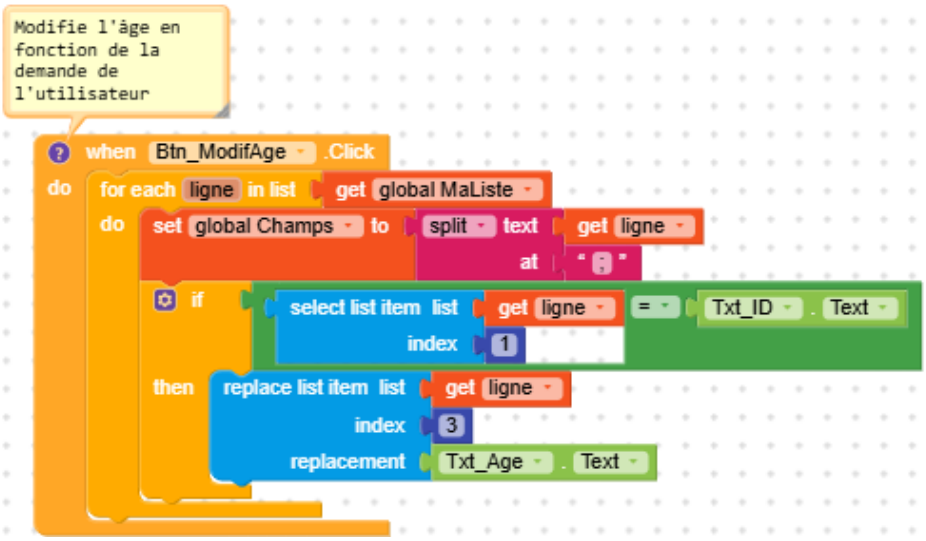
Toutes les commandes ci-avant permettent de créer un fichier CSV. Voyons à présent comment récupérer un fichier CSV sous forme de liste de listes. Ce qui permettra ensuite de modifier facilement une valeur comme dans un petit tableau.

La première étape consiste à lire le fichier. La deuxième permet de transformer le CSV en lignes et chaque ligne en colonnes.

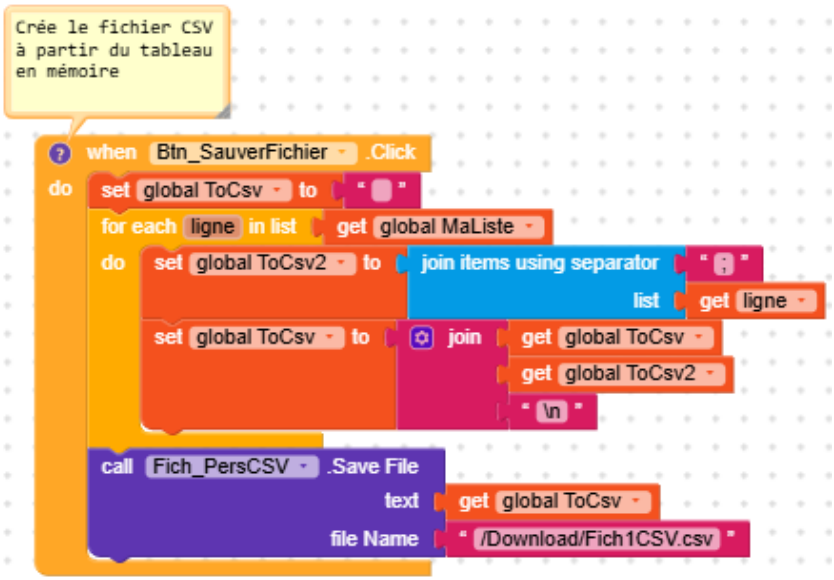


Supposons qu'il faille changer l'âge de Paul car il a 33 ans à présent. Paul est à la 2ème position dans la liste, mais il est préférable de l'identifier au moyen du contenu de la zone *ID*. Ainsi, peu importe sa position, l'application traitera bien la bonne personne. Nous utiliserons toutefois l'index 3 pour identifier la zone *Age* (Kodular commence les index de liste à 1, pas à 0).





Et enfin, voici les instructions pour créer le fichier CSV à partir du tableau en mémoire :



J. PARTAGER UN FICHIER

Une fois créé, un fichier peut être :

- Envoyé par e-mail ;
- Partagé avec une messagerie ;
- Sauvegardé dans un espace cloud ;
- Copié vers un ordinateur.

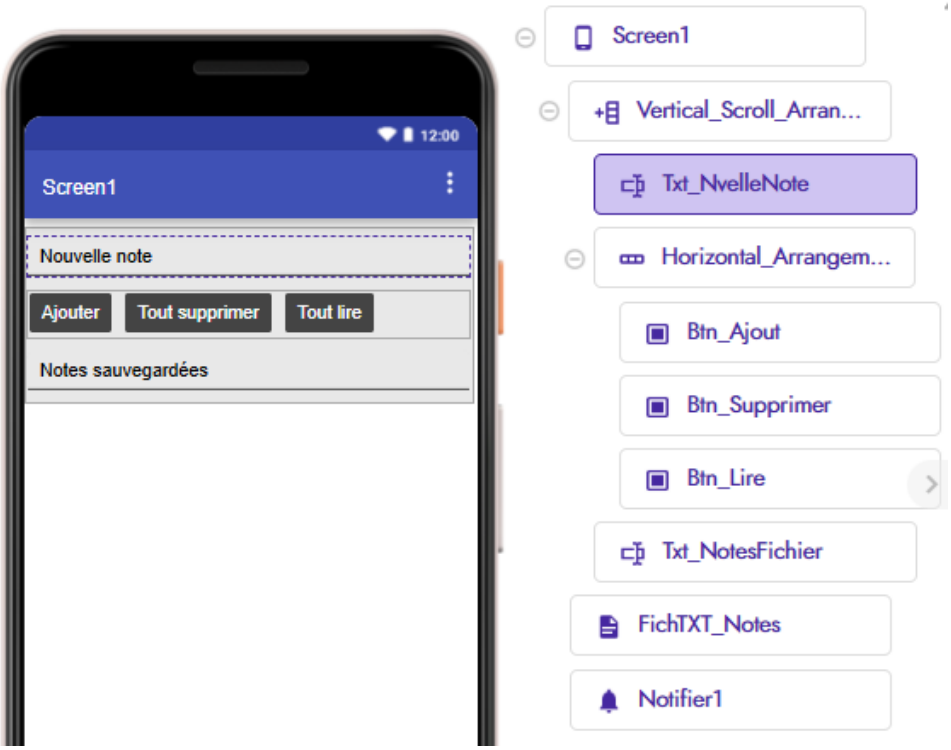
Cette possibilité rend les applications beaucoup plus utiles.

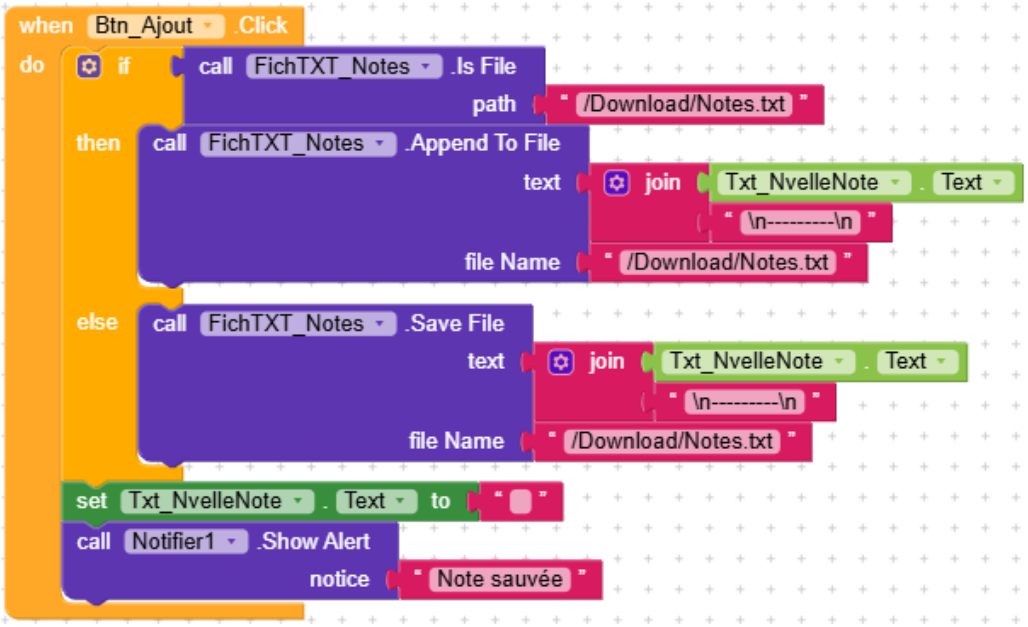
K. EXERCICE : BLOC-NOTES

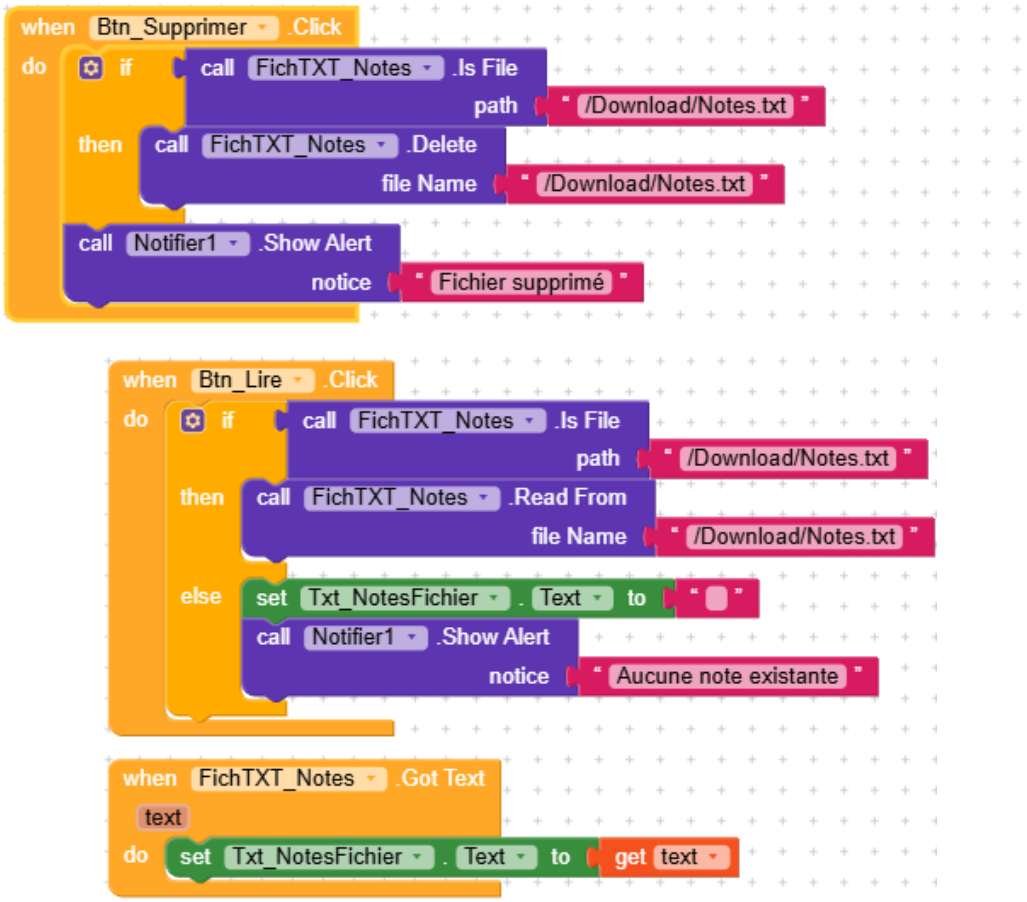
Créer une application contenant :

- Une zone verticale **Scrollable** ;
- Une **Text Box Multiline** *Nouvelle note* ;
- Un bouton *Ajouter* pour ajouter la note à la fin du fichier **Texte** ;

- Un bouton *Tout supprimer* pour vider le fichier ;
- Un bouton *Tout lire* pour lire toutes les notes du fichier et les afficher à l'écran ;
- Une zone d'affichage de toutes les notes présentes dans le fichier.





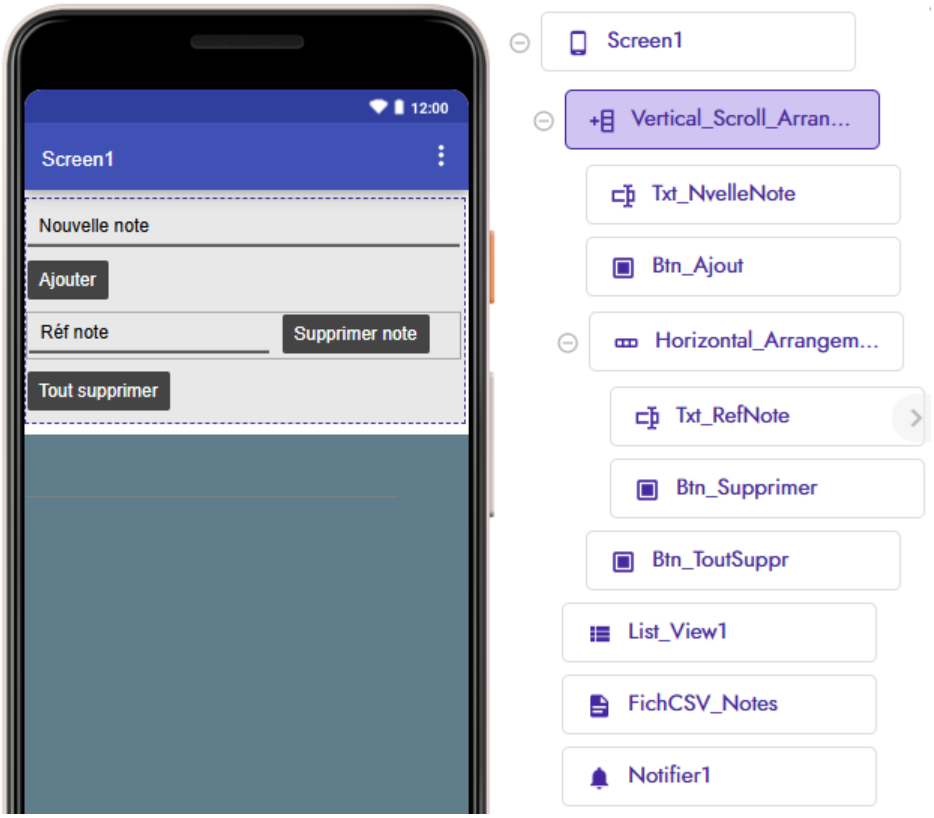


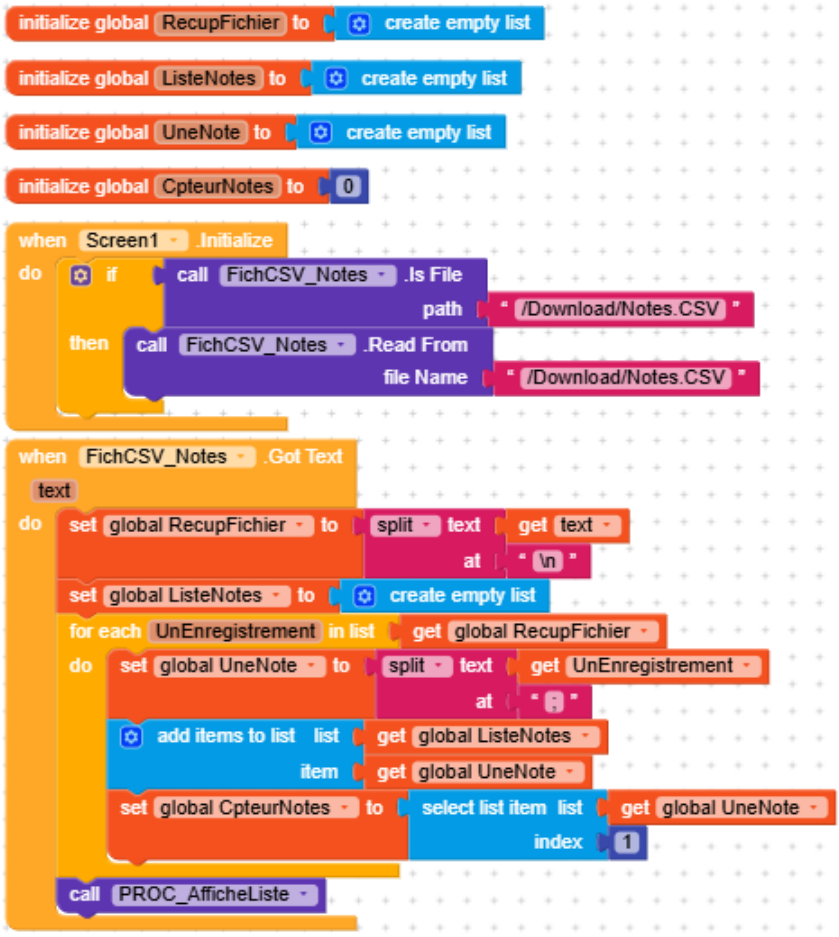
L. EXERCICE : BLOC-NOTES CSV

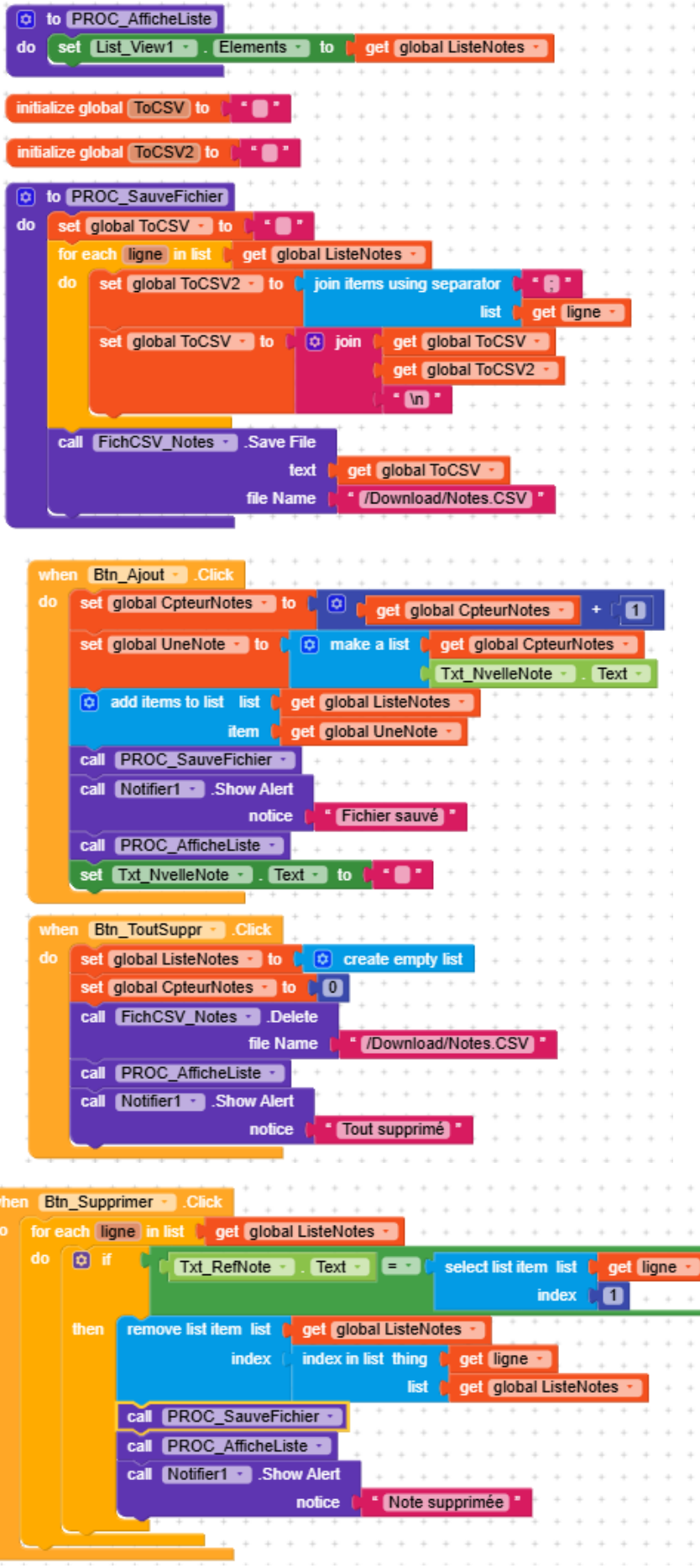
Créer une application contenant :

- Une zone verticale **Scrollable** ;
- Une **Text Box Multiline** *Nouvelle note* ;
- Un bouton *Ajouter* pour ajouter la note à la fin du fichier **CSV** ; Attention, dans le fichier CSV, il faut que toutes les notes soient numérotées automatiquement à partir de 1 ;
- Une zone *Note à supprimer* et le bouton *Supprimer* afin de supprimer la note correspondant au numéro indiqué.
- Un bouton *Tout supprimer* pour supprimer le fichier ;
- Lors du chargement de l'application et après chaque changement du fichier, le fichier est relu et réaffiché complètement dans une zone d'affichage (notes et leur numéro).

Le fichier pourra ensuite être ouvert avec un tableur.


12:00
Screen1
Nouvelle note
Ajouter
Réf note
Supprimer note
Tout supprimer
Screen1
Vertical_Scroll_Arran...
Txt_NvelleNote
Btn_Ajout
Horizontal_Arrangem...
Txt_RefNote
Btn_Supprimer
Btn_ToutSuppr
List_View1
FichCSV_Notes
Notifier1


initialize global RecupFichier to create empty list
initialize global ListeNotes to create empty list
initialize global UneNote to create empty list
initialize global CpteurNotes to 0
when Screen1 .Initialize
do if call FichCSV_Notes .Is File path "/Download/Notes.CSV"
then call FichCSV_Notes .Read From file Name "/Download/Notes.CSV"
when FichCSV_Notes .Got Text
text
do set global RecupFichier to split text get text at "\n"
set global ListeNotes to create empty list
for each UnEnregistrement in list get global RecupFichier
do set global UneNote to split text get UnEnregistrement at ";"
add items to list list get global ListeNotes item get global UneNote
set global CpteurNotes to select list item list get global UneNote index 1
call PROC_AfficheListe


to PROC_AfficheListe
do set List_View1 . Elements to get global ListeNotes
initialize global ToCSV to " "
initialize global ToCSV2 to " "
to PROC_SauveFichier
do set global ToCSV to " "
for each ligne in list get global ListeNotes
do set global ToCSV2 to join items using separator " ; "
list get ligne
set global ToCSV to join get global ToCSV
get global ToCSV2
" \n "
call FichCSV_Notes .Save File
text get global ToCSV
file Name " /Download/Notes.CSV "
when Btn_Ajout .Click
do set global CpteurNotes to get global CpteurNotes + 1
set global UneNote to make a list get global CpteurNotes
Txt_NvelleNote . Text
add items to list list get global ListeNotes
item get global UneNote
call PROC_SauveFichier
call Notifier1 .Show Alert
notice " Fichier sauvé "
call PROC_AfficheListe
set Txt_NvelleNote . Text to " "
when Btn_ToutSuppr .Click
do set global ListeNotes to create empty list
set global CpteurNotes to 0
call FichCSV_Notes .Delete
file Name " /Download/Notes.CSV "
call PROC_AfficheListe
call Notifier1 .Show Alert
notice " Tout supprimé "
when Btn_Supprimer .Click
do for each ligne in list get global ListeNotes
do if Txt_RefNote . Text = select list item list get ligne
index 1
then remove list item list get global ListeNotes
index index in list thing get ligne
list get global ListeNotes
call PROC_SauveFichier
call PROC_AfficheListe
call Notifier1 .Show Alert
notice " Note supprimée "

M. BONNES PRATIQUES

- Choisis des noms de fichiers explicites.
- Utilise le format **.txt** pour les données simples.
- Préfère le format **.csv** pour les tableaux.
- Vérifie qu'un fichier existe avant de le lire.
- Demande une confirmation avant une suppression.
- Sauvegarde régulièrement les données importantes.

N. RESUME

Dans ce chapitre, tu as appris à :

- Créer un fichier ;
- Ecrire dans un fichier ;
- Lire un fichier ;
- Modifier son contenu ;
- Supprimer un fichier ;
- Utiliser les formats **TXT** et **CSV**.

Les fichiers complètent parfaitement TinyDB. Ils permettent de conserver des informations, de les partager et de les échanger avec d'autres applications.

O. CE QUE NOUS VERRONS DANS LA PARTIE 4

Dans la prochaine partie, tu découvriras **SQLite**, la base de données intégrée à Android. Tu apprendras à enregistrer, rechercher, modifier et supprimer des milliers d'informations de manière structurée et très performante. Tu comprendras également quand choisir **TinyDB**, les **fichiers** ou une **base de données** selon les besoins de ton application.

4. DECOUVRIR SQLITE : LA BASE DE DONNEES INTEGREE A ANDROID

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre ce qu'est une base de données ;
- Découvrir le fonctionnement de **SQLite** ;
- Savoir quand utiliser **SQLite** plutôt que **TinyDB** ou des fichiers ;
- Comprendre les notions de table, d'enregistrement et de champ ;
- Comprendre le langage SQL de base ;
- Préparer tes futures applications utilisant des bases de données.

B. INTRODUCTION

Au fil des chapitres précédents, tu as appris plusieurs façons de conserver des informations.

Tu sais désormais utiliser :

- Les variables ;

- **TinyDB ;**
- Les fichiers Texte ;
- Les fichiers CSV.

Ces solutions sont excellentes pour de nombreuses applications. Mais imaginons maintenant que tu développes :

- Un catalogue contenant 20 000 produits ;
- Un carnet d'adresses de 5 000 contacts ;
- Une bibliothèque de 50 000 livres ;
- Un système de réservation ;
- Une gestion des stocks.

Dans ce cas, **TinyDB** devient moins adapté. Il faut un système capable de retrouver très rapidement une information parmi des milliers d'autres. C'est exactement le rôle d'une base de données.

C. QU'EST-CE QUE SQLITE ?

SQLite est une base de données légère intégrée à Android.

Elle ne nécessite :

- Aucun serveur ;
- Aucune connexion Internet ;
- Aucune installation particulière.

Toute la base est enregistrée dans un simple fichier. C'est l'une des raisons pour lesquelles elle est utilisée par des millions d'applications Android.

D. QU'EST-CE QU'UNE BASE DE DONNEES ?

Auparavant, l'information était conservée sur papier dans des dossiers qui étaient rangés dans des classeurs. Ces derniers contenaient l'ensemble des dossiers des livres et des lecteurs d'une bibliothèque par exemple. Ils correspondaient à ce qu'on appelle aujourd'hui une base de données.

On pouvait extraire de l'information de ces classeurs. Pour connaître le nombre d'emprunts de l'année de chaque lecteur, il fallait sortir un par un les dossiers de chaque lecteur et faire le total de tous les emprunts de l'année.

En informatique, une base de données est une collection de renseignements ou de données classés par sujet. Un classeur contenant tous les dossiers des clients d'une entreprise constitue la base de données des clients.

Prenons pour exemple une base de données contenant l'ensemble des renseignements sur la gestion d'une bibliothèque.

E. NOTION DE TABLE

L'organisation de cette base de données pourrait être telle qu'on aurait un dossier "Lecteurs", un dossier "Livres" et un dossier "Emprunts". En informatique, on dira que la base de données "Bibliothèque" contient les ***tables*** "Lecteurs", "Livres" et "Emprunts".

En d'autres mots, une base de données (DB) est un grand fichier composé de

plusieurs tables.

Une table est une sorte de tableau à deux dimensions contenant des informations sur un même sujet. Ainsi le fichier "Livres" contiendra les informations sur les livres, mais pas sur les lecteurs.

F. NOTION D'ENREGISTREMENT ET DE CHAMP

Une table reprend donc les données relatives à un type d'informations que l'on gère. Par exemple, la table "Lecteurs" contiendra les données des clients de la bibliothèque. Chaque "fiche client" aura la même structure qui pourrait être :

```
                    Signalétique lecteur
Référence... : ____
Nom......... : _____________________________
Prénom...... : _____________________________
Naissance... : __/__/____
Adresse..... : _____________________________
Ville....... : _____________________________  Code postal : ____
Tél......... : ____________
Commentaires : ___________________________________________
               ___________________________________________
```

Un ***enregistrement*** correspond à une "fiche client". En d'autres mots, il reprend toutes les informations relatives à une entité déterminée (= un client) de la table.

Un enregistrement est lui-même décomposé en petites zones appelées ***champs***. Un champ désigne donc chacune des données stockées dans un enregistrement.

Remarques :

En informatique, il existe souvent deux manières de visualiser les données : soit sur une fiche (cf. ci-avant), soit dans un tableau. On admettra que la structure "tableau" est exactement la même que celle de la fiche si ce n'est que les données sont écrites sur une seule ligne. Dans cette représentation, un enregistrement correspond à une ligne et un champ à une colonne.

Référence	Nom	Prénom	Naissance	Adresse	Ville	Code postal	Tél.	Commentaires
___	.....	.....	__/__/____	.....	.....	____	.....	.....
___	.....	.....	__/__/____	.....	.....	____	.....	.....
___	.....	.....	__/__/____	.....	.....	____	.....	.....
___	.....	.....	__/__/____	.....	.....	____	.....	.....

Lorsque l'on crée une base de données, il est très important de bien réfléchir tant à sa structure générale, qu'à la structure de chacune de ses tables. En effet, comme nous le verrons par la suite, toutes les opérations que l'on pourra effectuer par la suite dépendront toujours de la manière dont les données ont été structurées. Certaines opérations seront rendues très complexes, voire impossibles, si les différentes structures ont été mal dessinées. La définition des tables est donc un travail très important qui vaut la peine qu'on s'y attarde. Tout le temps "perdu" à ce niveau se rattrapera largement par la suite.

G. NOTION DE LIENS ENTRE LES TABLES

Dans le cadre de l'informatisation de la gestion de la bibliothèque reprise dans notre exemple, on établit la liste des listes de contrôle que le bibliothécaire pourrait avoir besoin dans le futur. Un des souhaits possibles serait de pouvoir vérifier tous les livres empruntés depuis le début de l'année. Nous, informaticiens, devons-nous poser la question suivante : "Que faut-il comme informations dans la base de données pour pouvoir répondre à sa demande ?". La réponse est évidemment celle-ci : il faut une table reprenant tous les emprunts effectués depuis le début de l'année (voire depuis la création de la bibliothèque). La question découlant de cette réponse est : "Quels champs devons-nous y définir ?" Il faut, bien évidemment, retenir la date de l'emprunt, ainsi que les informations relatives au lecteur et au livre. Mais est-il nécessaire de sauver dans cette table toutes les données du lecteur et du livre ? Non, il suffit d'y retenir les références du lecteur et du livre ! En effet, toutes les autres données se trouvent déjà dans les autres tables de la base de données. Il nous suffit d'établir un lien entre les tables "Emprunts" et "Lecteurs", ainsi qu'un lien entre les tables "Emprunts" et "Livres" pour avoir toutes les informations nécessaires.

En d'autres mots, les liens entre les tables évitent d'avoir à répéter les informations d'une table dans une autre. Et puisque l'on parle de répétition des données, insistons sur le fait qu'il est très important d'éviter toute répétition des données dans une base de données et ce pour diverses raisons :

- Eviter de consommer de la place mémoire inutilement
- Eviter une perte de temps lors des mises-à-jour des informations redondantes (ex.: adresse d'un lecteur, …)
- Eviter les incohérences entre les données si l'on oublie un des champs dupliqués lors des mises-à-jour

Précisons que le lien, physique ou logique, se fait grâce à un champ commun qui est une clé primaire dans l'une des deux tables.

H. NOTION DE CLE PRIMAIRE

Si nous reprenons notre exemple de table "Lecteurs" et que nous recherchons les informations concernant une personne précise, cela peut prendre beaucoup de temps si les données ne sont pas triées. En effet, si rien n'est précisé au système, il classe les informations dans l'ordre suivant lequel on les a introduites. Par contre, si on lui a demandé de trier les données sur le champ *Nom*, on retrouvera les informations de la personne voulue très rapidement.

Un problème subsiste toutefois. En effet, comment faire si deux personnes ont le même nom ? Pour contourner ce problème, les informaticiens ont défini la notion de ***clé primaire***.

Une clé primaire est une zone (ou un ensemble de zones) qui joue trois rôles :

- Le premier est d'assurer une identification unique de chaque enregistrement de la table.
- Le deuxième est de préciser l'ordre suivant lequel les données seront triées.
- Le troisième consiste à établir des liens entre les diverses tables de la base de données.

Dans notre exemple, le champ *Référence* sera utilisé comme clé primaire. Cette

référence sera définie comme étant unique et ne sera donc jamais attribuée à deux lecteurs.

I. NOTION D'INDEXATION DES DONNEES

Si une clé primaire est souvent indispensable pour toutes les tables, elle n'est toutefois pas suffisante. En effet, il ne faut pas oublier que le rôle principal d'une clé primaire est d'identifier de manière unique un enregistrement dans une table. C'est pour cette raison que l'on attribue généralement une référence (numérique ou autre) à chaque enregistrement ; référence qui sert de clé primaire. Mais qu'arrive-t-il lorsque l'on recherche un enregistrement dont on ne connaît pas cette référence ? Il suffit alors, simplement, de rechercher l'enregistrement en comparant le contenu d'autres champs (ex.: le nom d'une personne ou le titre d'un livre). Le problème, c'est que dans ce cas, le système doit lire tous les enregistrements de la table pour trouver celui que l'on a demandé. Ce qui peut prendre beaucoup de temps. Pour éviter cette perte de temps, on va créer un index sur les champs des tables que l'on utilisera lors des recherches. On pourrait donc indexer un champ *Ville* pour retrouver rapidement tous les clients de Bruxelles.

Attention, contrairement à la clé primaire, les index supplémentaires autorisent les doublons (= valeur identique pour deux enregistrements).

J. LES OPERATIONS PRINCIPALES

Une base de données permet principalement de :

- Ajouter une ligne ;
- Rechercher une ligne ;
- Modifier une ligne ;
- Supprimer une ligne
- Trier les données et/ou les résultats ;
- Sélectionner les données.

Ces opérations sont présentes dans pratiquement toutes les applications.

K. METHODOLOGIE A SUIVRE POUR LA CONCEPTION D'UNE BASE DE DONNEES

Lorsque l'on doit créer une nouvelle base de données, ou même si l'on veut adapter une base de données existante, il est conseillé de suivre une certaine méthodologie. Il en existe plusieurs, en voici donc une qui reprend la base de plusieurs de celles-ci.

Il est conseillé de travailler en suivant les étapes suivantes :

1. Analyser les souhaits, besoins et situation actuelle du client
2. Identifier les entités en présence
3. Lister leurs attributs
4. Définir leurs identifiants
5. Définir les liens entre les entités et ainsi établir le schéma relationnel
6. Optimiser au mieux la base de données

L. ANALYSE DES SOUHAITS, BESOINS ET SITUATION ACTUELLE DU CLIENT

La première étape consiste bien évidemment à rencontrer le client pour savoir ce qu'il souhaite, quel est son environnement de travail, quels sont ses besoins

actuels mais aussi futurs, … Existe-t-il déjà une base de données ?

Attention, on ne doit pas se contenter de ce qu'il dit, il faut se mettre à sa place, tenter de comprendre son métier et imaginer tout ce que l'informatique peut lui apporter.

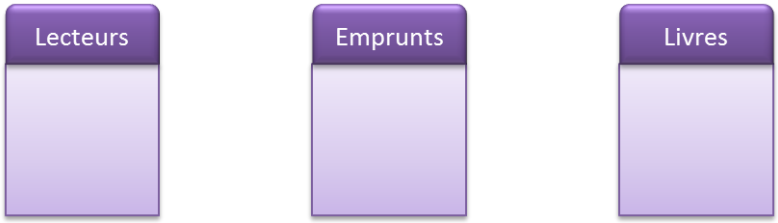
Il faut ensuite lui faire une présentation de ce que l'on compte réaliser au niveau informatique afin de voir si l'on a bien compris ses besoins et souhaits.

M. IDENTIFIER LES ENTITES

La deuxième étape consiste à identifier les entités.

Une entité est un ensemble d'objets de même structure au sujet desquels on conserve de l'information dans la base de données. Par même structure, on entend les éléments pour lesquels on enregistre les mêmes informations. A chaque entité correspondra au moins une table.

Dans notre exemple de gestion d'une bibliothèque, on aurait plusieurs entités dont, par exemple, les livres, les lecteurs. Mais on pourrait en avoir d'autres en fonction du niveau d'informatisation de la gestion tels que le personnel, etc.



Dans un magasin, on pourrait avoir des entités telles que les clients, les articles, les fournisseurs, etc.

N. LES ATTRIBUTS DES ENTITES

Par attribut, on entend les informations liées aux entités. En fait, les attributs correspondent aux champs des tables.

Quelles sont les informations nécessaires et utiles pour les traitements informatiques à développer ? Dans quelle table doit-on les stocker pour pouvoir les retrouver facilement ?

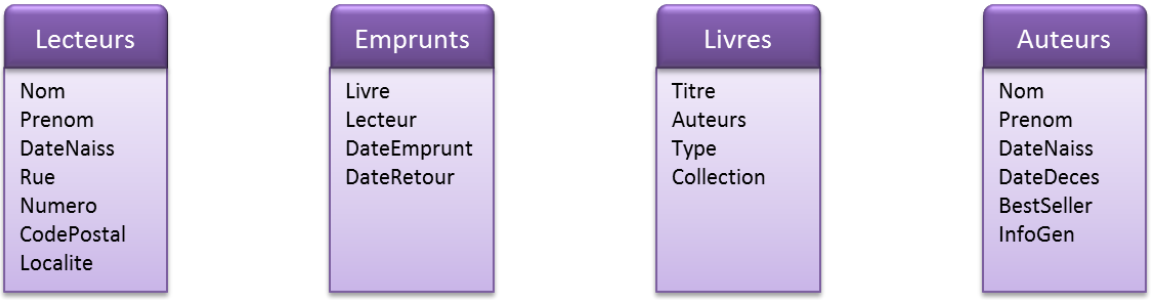
C'est aussi lors de cette étape que l'on va éventuellement définir de nouvelles entités auxquelles on n'avait pas pensé lors de l'étape précédente.

En effet, il faut veiller à ce que la base de données contienne toutes les informations nécessaires, mais aussi évite toute redondance des informations qui y sont contenues et reste cohérente.

Si nous prenons l'exemple de l'entité Livres de la base de données *Bibliothèque*, il pourrait s'avérer nécessaire d'y ajouter de nouvelles entités qui contiendraient :

- Les Types des livres (ex.: Roman, biographie, …) afin de garder une certaine cohérence des données pour tous les livres (toujours le même nom avec la même orthographe afin de faciliter de futures recherches et/ou sélections)
- Les Auteurs avec leur biographie. Cette entité séparée permettra d'éviter les redondances d'informations telles que les best-sellers de l'auteur au

niveau de chaque livre.



Petit conseil : Lors de la définition des champs, il faut veiller à faire des champs ne contenant qu'une seule information. On ne créera pas le champ Adresse, mais les champs Rue, Numéro, Boîte, Code Postal, Localité, … Cela facilitera fortement les traitements ultérieurs de recherche, …

O. LES IDENTIFIANTS DES ENTITES

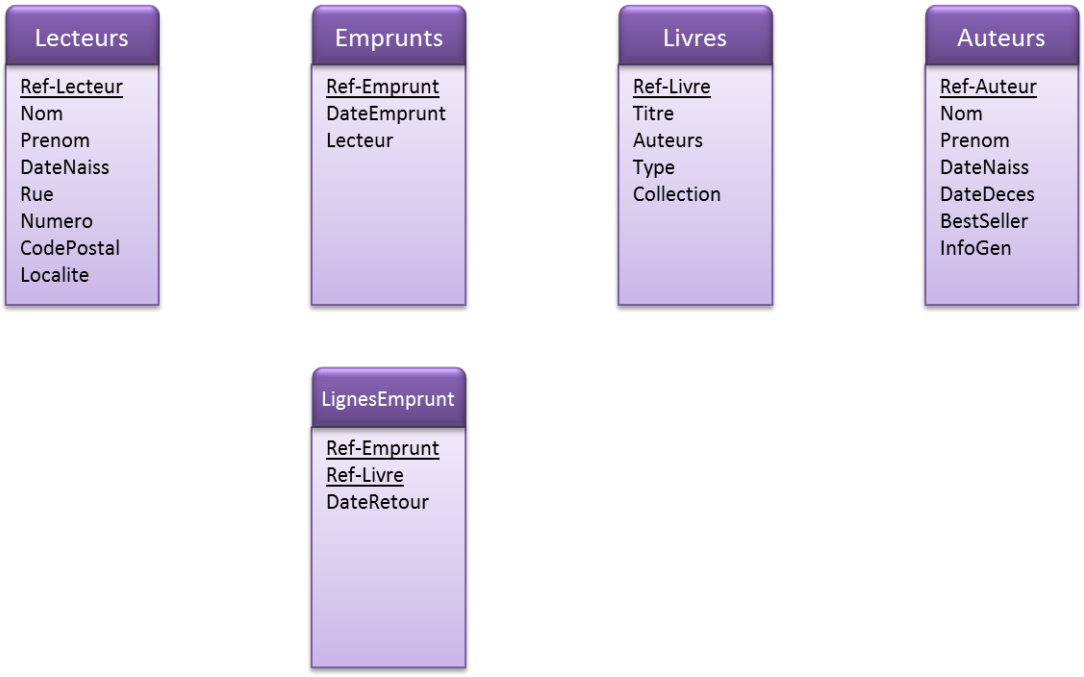
Chaque élément d'une entité doit pouvoir être identifiable de manière unique. C'est pourquoi toutes les entités doivent posséder un attribut sans doublon (c'est-à-dire sans qu'il soit possible d'avoir une deuxième occurrence dans la même entité).

Cet identifiant correspond généralement à une référence numérique ou, plus rarement, alphanumérique.

Dans notre exemple, on associera un numéro à chaque lecteur. En effet, nous ne pouvons pas utiliser leurs noms et prénoms comme identifiant unique car il est possible qu'il y ait plusieurs lecteurs avec les mêmes noms et prénoms.

Cet identifiant servira aussi de clé primaire.

Attention, lorsque l'on choisit l'identifiant d'une table, il faut éviter les identifiants susceptibles de changer avec le temps tels que les numéros des cartes d'identité, les âges, les adresses, …



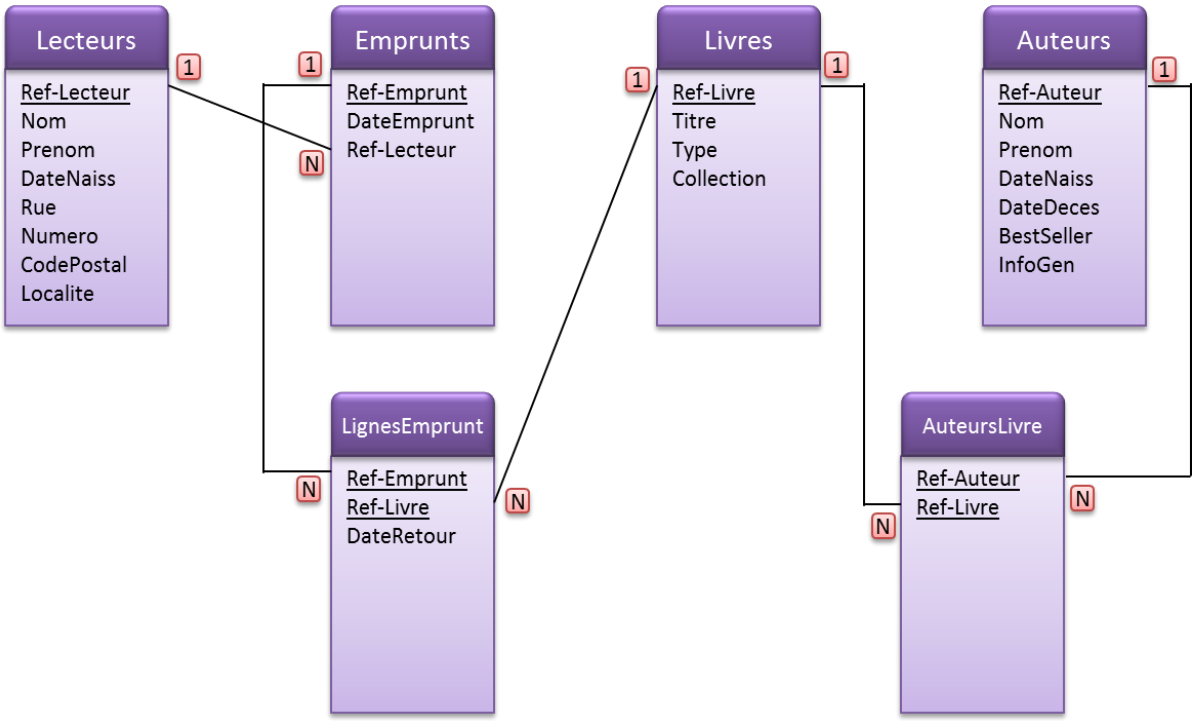
P. LES LIENS ENTRE LES ENTITES, LE SCHEMA RELATIONNEL

Il faut à présent définir les liens entre les différentes entités/tables. Dans notre exemple, on créera différentes associations :
- Un lecteur empruntera 0, un ou plusieurs livres
- Un livre sera emprunté par 0, un ou plusieurs lecteurs
- Un livre sera écrit par un ou plusieurs auteurs
- Un auteur écrira un ou plusieurs livres
- Lors d'un emprunt, plusieurs livres peuvent être empotés

On remarquera aussi que s'il n'y a pas de lien direct entre les lecteurs et les livres, un lien indirect existe via les emprunts.

De plus, sur le schéma relationnel, on notera aussi la cardinalité des relations. Celle-ci indique le nombre maximum de fois qu'un élément peut être concerné par l'association.

Enfin, on remarquera aussi que les champs utilisés au niveau des liens sont généralement la clé primaire d'une entité d'une part et son équivalent dans la seconde entité d'autre part.

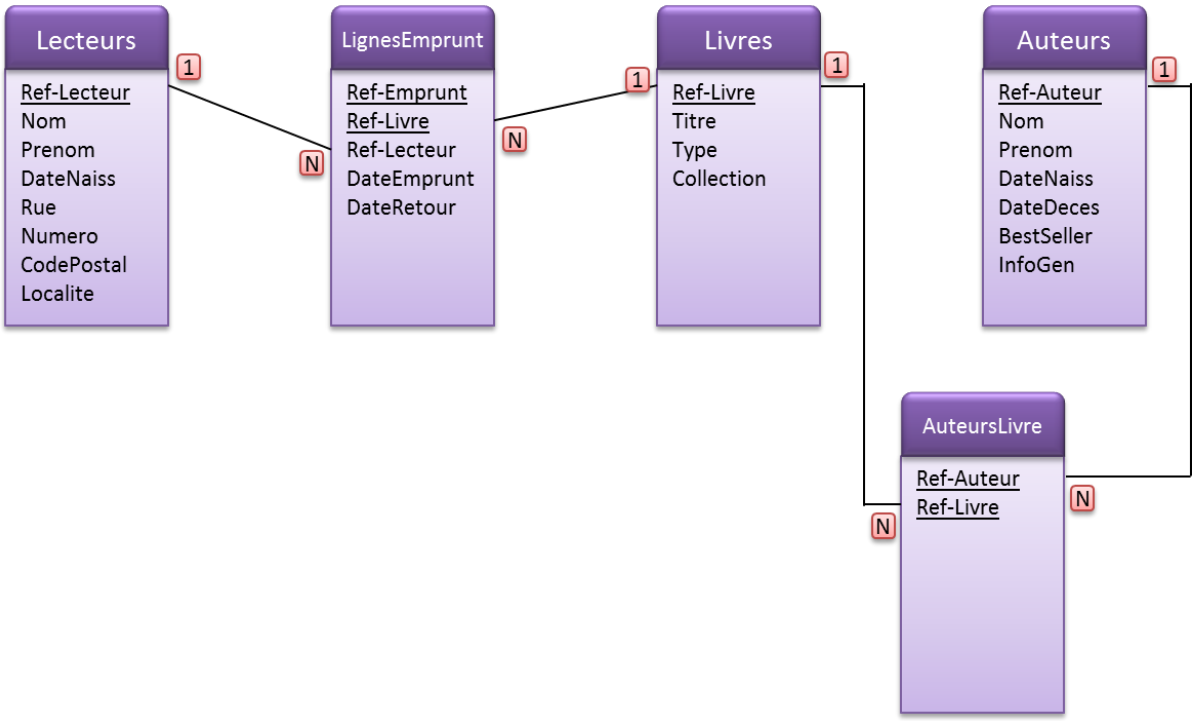


Q. LES OPTIMISATIONS POSSIBLES

Pour les bases de données relationnelles, l'optimisation qui vise à accélérer les requêtes peut passer par :

- L'ajout d'index aux tables sur les champs régulièrement utilisés lors des recherches ou jointures.
- L'ajout de champs calculés ou de certaines redondances pour éviter des jointures coûteuses. Mais, dans ce cas, il faudra veiller à ce que la cohérence entre les données soit respectée.
- La suppression des contraintes d'unicité au sein d'une table. Par exemple,

les tables Emprunts et LignesEmprunt seraient fusionnées et on ne garderait que la table des lignes des emprunts.



R. TINYDB, FICHIERS OU SQLITE ?

Le choix dépend du projet :

Solution	**Idéale pour**
Variables	Données temporaires
TinyDB	Petites quantités de données
Fichiers TXT/CSV	Exportation et partage
SQLite	Grandes quantités de données structurées

Kodular permet d'utiliser **SQLite**, mais sa mise en œuvre est plus exigeante que **TinyDB**. En effet, presque toutes les commandes liées à **SQLite** devront se faire au moyen du langage **SQL** car elles n'existent pas sous forme de **Blocks**. Dans la plupart des projets destinés aux débutants, **TinyDB** est souvent suffisant. **SQLite** devient particulièrement intéressant lorsque :

- Les données sont nombreuses ;
- Les recherches sont complexes ;
- Les performances sont importantes.

C'est pourquoi il est conseillé de bien maîtriser **TinyDB** avant de passer à **SQLite**.

S. LES COMMANDES SQL

Supposons que l'on ait la base de données suivante :

Nom	Type
Ref_Lecteur	int(10)
Nom	varchar(50)
Prenom	varchar(40)
DateNaiss	date
Rue	varchar(50)
Numero	int(11)
CodePostal	int(11)
Localite	varchar(50)

Nom	Type
Ref_Livre	int(10)
Titre	varchar(80)
Type	enum('BD', 'Fiction', 'Humour', 'Nouvelle', 'Polic...
Collection	varchar(50)
AnneeEdition	int(10)
NbrImpression	bigint(20)

Nom	Type
Ref_Auteur	int(10)
Nom	varchar(50)
Prenom	varchar(40)
DateNaiss	date
DateDeces	date
BestSeller	varchar(80)
InfoGen	varchar(150)

Nom	Type
RefEmprunt	int(11)
RefLivre	int(11)
RefLecteur	int(11)
DateEmprunt	date
DateRetour	date

Nom	Type
Ref_Auteur	int(10)
Ref_Livre	int(10)

Avec les données :

Dans la table Lecteurs :

Ref-Lecteur	Nom	Prenom	DateNaiss	Rue	Numero	CodePostal	Localite
1	Hofmans	Pierre	1980-02-24	Marienborre	44	1120	Bruxelles
2	Hofmans	Bernard	1980-02-24	Marienborre	44	1120	Bruxelles
3	Tare	Guy	1995-03-10	Mutsaerd	14	1020	Bruxelles
4	Entaille	Eve	1992-08-24	Chaussée de Mons	1043	1070	Bruxelles

Dans la table Auteurs :

Ref-Auteur	Nom	Prenom	DateNaiss	DateDeces	BestSeller	InfoGen
1	Higgins Clark	Mary	1927-12-24	0000-00-00	Un jour tu verras...	
2	Higgins Clark	Carol	1956-07-28	0000-00-00	Ce soir je veillerai sur toi	A écrit plusieurs livres avec sa mère Mary
3	Lenteric	Bernard	1944-01-01	2009-03-25	La Nuit des enfants rois	
4	Christie	Agatha	1890-09-15	1976-01-12	Les dix petits nègres	Son personnage fétiche est Hercule Poirot

Dans la table Livres :

Ref-Livre	Titre	Type	Collection	AnneeEdition	NbrImpression
1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000
2	Un jour tu verras	Policier	Le Grand Livre du Mois	1993	180000
3	La nuit des enfants rois	Roman	Livre de Poche	1981	214000
4	Substance B	Roman	Livre de Poche	1986	250000
5	Le train bleu	Policier	Livre de Poche	1965	512000
6	Hercule Poirot contre ABC	Policier	Livre de Poche	1970	351000

Dans la table AuteursLivre :

Ref-Auteur	Ref-Livre
1	1
1	2
2	1
3	3
3	4
4	5
4	6

Dans la table LignesEmprunt :

RefEmprunt	RefLivre	RefLecteur	DateEmprunt	DateRetour
1	1	1	2017-01-02	2017-01-09
2	1	2	2017-01-10	0000-00-00

Les créations des tables se feront au moyen de commandes SQL semblables à

```
CREATE TABLE IF NOT EXISTS Livres (
   ref_Livre      INTEGER PRIMARY KEY AUTOINCREMENT,
   titre          TEXT NOT NULL,
   type           TEXT NOT NULL
                  CHECK (type IN('BD', 'Fiction', 'Humour', 'Nouvelle',
                                 'Policier', 'Roman')),
   collection     TEXT,
   anneeedition   INTEGER,
   nbrImpression  INTEGER NOT NULL DEFAULT 0,
);

CREATE INDEX IF NOT EXISTS idx_livres_titre
  ON Livres(titre);
```

S'il faut ajouter une colonne à une table existante, on exécutera une commande semblable à :

```
ALTER TABLE Livres
   ADD COLUMN editeur TEXT;
```

S'il faut renommer une colonne :

```
ALTER TABLE Livres
   RENAME COLUMN titre TO nom_livre;
```

Pour l'ajout d'enregistrement :

```
INSERT INTO Livres
    (titre, type, collection, anneeedition, nbrimpression)
VALUES
    ('Ce soir, je veillerai sur toi', 'Policier', 'Le Grand Livre du Mois',
2001, 450000),
    ('Un jour tu verras', 'Policier', 'Le Grand Livre du Mois', 1993, 180000);
```

Pour la lecture d'enregistrements d'une table, on utilise la commande **SELECT**. La version la plus courante est ***SELECT * FROM Livres***. Analysons-la.

- **SELECT** : En SQL, le premier mot indique le type d'opération à effectuer. **SELECT** demande d'afficher ce que contient une table.
- ***** : Après le **SELECT**, on doit indiquer les champs de la table que l'on désire récupérer. Les noms des champs doivent être écrits les uns après les autres et séparés par une virgule (ex.: SELECT titre, collection FROM livres). Si l'on désire tous les champs de la table, on peut remplacer la liste des noms par une étoile.
- **FROM** : C'est un mot réservé du SQL qui indique que le paramètre suivant sera le nom de la table à utiliser pour la commande
- Livres : Nom de la table à lire

La clause DISTINCT

Le bibliothécaire souhaite envoyer des toutes-boîtes dans les communes dans lesquelles au moins un de ses lecteurs habite. Pour cela, il exécute la commande SQL SELECT codepostal, localite FROM lecteurs. Celle-ci lui affiche bien toutes les communes à l'écran, malheureusement elle affiche plusieurs fois les mêmes… N'y a-t-il pas moyen d'éviter l'affichage des doublons ?

Les concepteurs du SQL ont solutionné ce problème en créant la clause **DISTINCT** que l'on peut ajouter juste après le mot **SELECT**. Cette clause indique qu'il ne

faut pas afficher les enregistrements dont l'ensemble des champs à afficher est identique à un autre enregistrement déjà affiché.

Exemple : `SELECT DISTINCT codepostal, localite FROM Lecteurs`

La clause *WHERE*

Cette clause, ajoutée à la suite de la commande **SELECT**, permet de filtrer les enregistrements en limitant la sélection en fonction de certaines conditions. Pour ce faire, on utilisera les opérateurs de comparaisons **<**, **>**, **=**, **<=**, **>=** et **<>** (ou **!=**).

Supposons que je ne souhaite avoir que les livres "policier" :

```
SELECT *
FROM livres
WHERE TYPE = 'Policier'
```

Notons que les chaînes de caractères doivent être placées entre apostrophes ou guillemets pour les délimiter (ex.: 'Policier') ; ce qui n'est pas nécessaire pour les nombres.

Il est également possible de combiner plusieurs conditions grâce aux mots **AND** et **OR**, voire des parenthèses, ajoutés entre les différentes conditions :

```
SELECT *
FROM livres
WHERE TYPE = 'Policier'
AND NbrImpression >200000
```

Le langage SQL étant assez développé, il permet de faire des sélections plus complexes grâce notamment aux mots réservés **NOT** et **LIKE**.

NOT, comme nous l'avons déjà vu dans d'autres langages, permet d'inverser le résultat d'une condition. Si le résultat était True, il vaudra False et inversement.

LIKE permet d'utiliser des wildcards dans la zone de recherche : le caractère _ sera remplacé par 1 et 1 seul caractère et le caractère % sera remplacé par 0, 1 ou plusieurs caractères.

```
SELECT *
FROM livres
WHERE Collection LIKE "%livre%"
```

Toutes les collections contenant le mot livre

```
SELECT *
FROM livres
WHERE Collection LIKE "livre%"
```

Toutes les collections commençant par le mot livre

```
SELECT *
FROM livres
WHERE Collection LIKE "l_vre%"
```

Toutes les collections commençant par la lettre l, suivie d'un caractère et des lettres vre

La clause *ORDER BY*

Cette clause, ajoutée à la suite de la commande **SELECT**, permet de trier les résultats sur le contenu de certains champs de manière croissante (**ASC**) ou décroissante (**DESC**).

Supposons que je souhaite avoir tous les livres écrits après 1965, triés par type (croissant) et nombre d'impressions (décroissant) :

```
SELECT *
FROM livres
WHERE AnneeEdition >1965
ORDER BY TYPE ASC , NbrImpression DESC
```

La modification des enregistrements d'une table se fait au moyen de la commande **UPDATE** dont la syntaxe est la suivante :

```
UPDATE table
SET champ1 = valeur1, champ2 = valeur2, …
WHERE condition
```

La suppression d'enregistrements se fait très, voire trop, facilement. En effet, il suffit de taper la commande **DELETE FROM table WHERE condition**. Il faudra donc veiller tout particulièrement à la condition. En cas d'erreur, on risque d'effacer à tout jamais des enregistrements que l'on ne voulait pas effacer.

Les Jointures

Tout cela, c'est très bien… Mais, nous, nous avons besoin de données qui se trouvent dans plusieurs tables. Comment pouvons-nous regrouper les tables lors d'un **SELECT** ? Eh bien, c'est relativement simple. Nous allons utiliser les clauses **JOIN** !

Il existe plusieurs types de jointures, nous allons en étudier 3 dans ce cours.

L'équijointure

La première, l'équijointure, consiste à opérer une jointure entre les tables avec une condition d'égalité.

Cette jointure se faisait auparavant au moyen de la commande **SELECT** telle qu'on l'a déjà étudiée. Il suffisait de préciser les différentes tables souhaitées au niveau du **FROM** et les critères de jointure au niveau du **WHERE**.
Attention, le nom des champs commun à plusieurs tables doivent être préfixés du nom de la table liée (ex.: Livres.RefLivre).

Exemple 1 :

```
SELECT *
FROM `livres` , auteurs, auteurslivre
WHERE livres.reflivre = auteurslivre.reflivre
AND auteurslivre.refauteur = auteurs.refauteur
```

RefLivre	Titre	Type	Collection	AnneeEdition	NbrImpression	RefAuteur	Nom	Prenom	DateNaiss	DateDeces	BestSeller	InfoGen	RefAuteur	RefLivre
1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000	1	Higgins Clark	Mary	1927-12-24	0000-00-00	Un jour tu verras...		1	1
2	Un jour tu verras	Policier	Le Grand Livre du Mois	1993	180000	1	Higgins Clark	Mary	1927-12-24	0000-00-00	Un jour tu verras...		1	2
1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000	2	Higgins Clark	Carol	1956-07-28	0000-00-00	Ce soir je veillerai sur toi	A écrit plusieurs livres avec sa mère Mary	2	1
3	La nuit des enfants rois	Roman	Livre de Poche	1981	214000	3	Lenteric	Bernard	1944-01-01	2009-03-25	La Nuit des enfants rois		3	3
4	Substance B	Roman	Livre de Poche	1986	250000	3	Lenteric	Bernard	1944-01-01	2009-03-25	La Nuit des enfants rois		3	4
5	Le train bleu	Policier	Livre de Poche	1965	512000	4	Christie	Agatha	1890-09-15	1976-01-12	Les dix petits nègres	Son personnage fétiche est Hercule Poirot	4	5
6	Hercule Poirot contre ABC	Policier	Livre de Poche	1970	351000	4	Christie	Agatha	1890-09-15	1976-01-12	Les dix petits nègres	Son personnage fétiche est Hercule Poirot	4	6

Le résultat affiche 7 lignes de données. En effet, cette jointure recherche pour chaque enregistrement de la table Livres, les données correspondantes dans les autres tables. S'il ne trouve pas de données, cet enregistrement ne sera pas affiché. Par contre, dans le cas où il en trouve plusieurs, l'enregistrement sera affiché en plusieurs exemplaires (ex.: Le livre "Ce soir, je veillerai sur toi").

Exemple 2 :

```
SELECT *
FROM `lignesemprunt` , livres, lecteurs
WHERE lignesemprunt.reflivre = livres.reflivre
AND lignesemprunt.reflecteur = lecteurs.reflecteur
```

RefEmprunt	RefLivre	RefLecteur	DateEmprunt	DateRetour	RefLivre	Titre	Type	Collection	AnneeEdition	NbrImpression	RefLecteur	Nom	Prenom	DateNaiss	Rue	Numero	CodePostal	Localite
1	1	1	2017-01-02	2017-01-09	1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000	1	Hofmans	Pierre	1980-02-24	Marienborre	44	1120	Bruxelles
2	1	2	2017-01-10	0000-00-00	1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000	2	Hofmans	Bernard	1980-02-24	Marienborre	44	1120	Bruxelles

Dans ce cas-ci, seules 2 lignes sont affichées car les autres livres n'ont jamais été empruntés. Aucun enregistrement lié n'est donc présent dans la table des emprunts.

Précisons encore qu'il est possible d'ajouter des critères de sélection supplémentaires afin de limiter le nombre d'enregistrements affichés.

Exemple 3 :

```
SELECT *
FROM `lignesemprunt` , livres, lecteurs
WHERE lignesemprunt.reflivre = livres.reflivre
AND lignesemprunt.reflecteur = lecteurs.reflecteur
AND dateretour = "0000-00-00"
```

Dans ce cas, seuls les livres empruntés non rendus seront affichés.

Les jointures réalisées avec la clause WHERE étant quelque peu limitées, une nouvelle clause est apparue : la clause JOIN. Etudions-la.

Exemple 4 :

```
SELECT *
FROM `lignesemprunt` E
JOIN livres LI ON E.reflivre = LI.reflivre
JOIN lecteurs LE ON E.reflecteur = LE.reflecteur
WHERE E.dateretour = "0000-00-00"
```

Comme pour l'exemple 3, seuls les livres empruntés non rendus seront affichés.

Les jointures se font au moyen des clauses **JOIN table ON critère** où table indique la table liée et critère indique les critères de jointure entre deux tables.



Notons qu'afin de faciliter la lecture des commandes, on peut utiliser le surnommage qui consiste à remplacer dans la commande le nom d'une table par un alias (ex.: LignesEmprunt est remplacé par E, Livres par LI, Lecteurs par LE).

Les jointures externes LEFT

Si l'équijointure est la plus utilisée, il se peut qu'elle ne réponde pas aux besoins dans certaines situations. En effet, elle ne sélectionne que les enregistrements dont des données sont présentes dans chaque table de la jointure. Or, il se pourrait que l'on ait besoin d'afficher les enregistrements dès que des données sont présentes dans au moins une des tables. Prenons l'exemple de deux tables : la première contient les données signalétiques des

clients et la deuxième les numéros des téléphones. Si un client ne possède pas de téléphone, une équijointure entre ces deux tables n'affichera pas le client en question. C'est pourquoi, on a mis au point les jointures externes.

Etudions la première : la jointure externe **LEFT**.

Lorsque l'on ajoute le mot **LEFT** à la gauche de **JOIN**, cela signifie que la table principale est la première table citée et que tous les enregistrements qui y sont présents et qui répondent à la condition **WHERE** seront affichés et ce même s'ils n'ont pas de correspondance dans la seconde table. Ainsi, on pourrait afficher tous les clients qu'ils aient ou non un numéro de téléphone.

Exemple :

```
SELECT *
FROM `lignesemprunt` E
LEFT JOIN livres LI ON E.reflivre = LI.reflivre
```

RefEmprunt	RefLivre	RefLecteur	DateEmprunt	DateRetour	RefLivre	Titre	Type	Collection	AnneeEdition	NbrImpression
1	1	1	2017-01-02	2017-01-09	1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000
2	1	2	2017-01-10	0000-00-00	1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000

Dans cet exemple, on affiche tous les emprunts (en cours ou non).

Les jointures externes RIGHT

La seconde jointure externe étudiée est la jointure externe **RIGHT**.

Dans ce cas-ci, on ajoute le mot RIGHT à la gauche de **JOIN**, cela signifie que la table principale est la seconde table citée et que tous les enregistrements qui y sont présents et qui répondent à la condition **WHERE** seront affichés et ce même s'ils n'ont pas de correspondance dans la première table.

Exemple :

```
SELECT *
FROM `lignesemprunt` E
RIGHT JOIN livres LI ON E.reflivre = LI.reflivre
```

RefEmprunt	RefLivre	RefLecteur	DateEmprunt	DateRetour	RefLivre	Titre	Type	Collection	AnneeEdition	NbrImpression
1	1	1	2017-01-02	2017-01-09	1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000
2	1	2	2017-01-10	0000-00-00	1	Ce soir je veillerai sur toi	Policier	Le Grand Livre du Mois	2001	450000
NULL	*NULL*	*NULL*	*NULL*	*NULL*	2	Un jour tu verras	Policier	Le Grand Livre du Mois	1993	180000
NULL	*NULL*	*NULL*	*NULL*	*NULL*	3	La nuit des enfants rois	Roman	Livre de Poche	1981	214000
NULL	*NULL*	*NULL*	*NULL*	*NULL*	4	Substance B	Roman	Livre de Poche	1986	250000
NULL	*NULL*	*NULL*	*NULL*	*NULL*	5	Le train bleu	Policier	Livre de Poche	1965	512000
NULL	*NULL*	*NULL*	*NULL*	*NULL*	6	Hercule Poirot contre ABC	Policier	Livre de Poche	1970	351000
NULL	*NULL*	*NULL*	*NULL*	*NULL*	7	Tintin en Amérique	BD	Casterman	1930	352000

Dans cet exemple, on affiche tous les emprunts (en cours ou non), ainsi que tous les livres jamais empruntés.

T. EXERCICE : CONCEPTION D'UNE BASE DE DONNEES POUR UNE BIBLIOTHEQUE

Créer une bibliothèque SQLite dans Kodular

Tu vas construire une base de données pour ta bibliothèque personnelle, utilisable sur Android depuis ton application Kodular. Elle permettra de stocker tes livres, de les rechercher, de modifier leurs informations et d'indiquer ceux

que tu as lus.

1. Structure de la base de données

Je te propose de commencer par une seule table Livres. Pour une bibliothèque personnelle, elle suffit et sera plus simple à gérer.

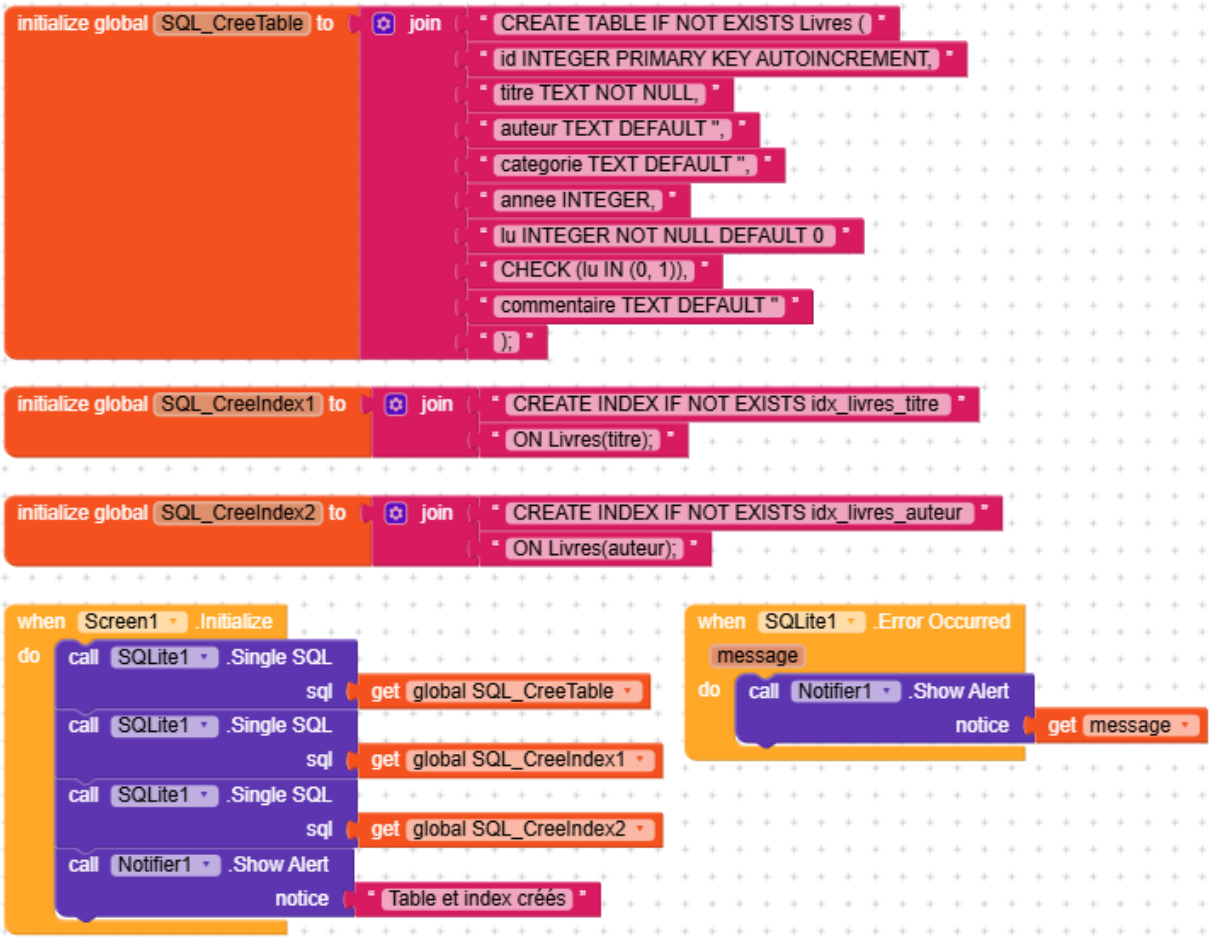
Base SQLite locale sur Android - Table Livres

Champ	Type	Description
id	INTEGER	Identifiant unique, automatique
titre	TEXT	Titre du livre (obligatoire)
auteur	TEXT	Auteur ou autrice
categorie	TEXT	Roman, BD, histoire, etc.
annee	INTEGER	Année de publication, facultative
lu	INTEGER	0 = non lu ; 1 = lu
commentaire	TEXT	Notes personnelles, facultatives

2. Le SQL complet

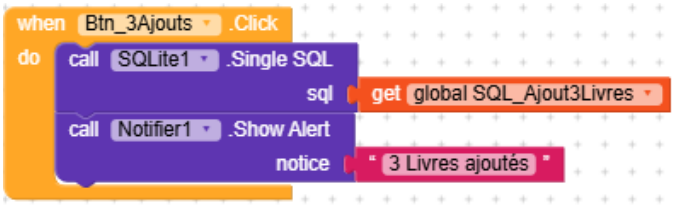
Ces commandes créent la table, ajoutent un index pour accélérer les recherches et insèrent quelques livres de test.

A. Créer la table et les index au démarrage de l'application



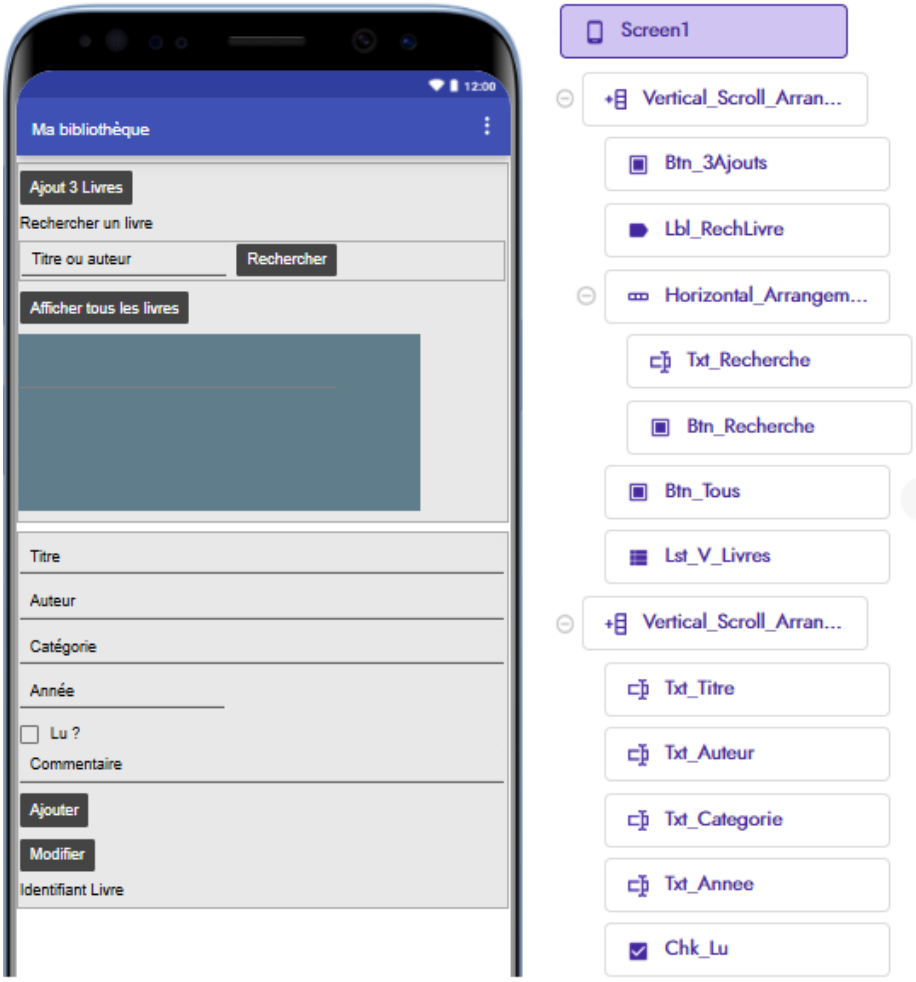
B. Ajouter des livres de test

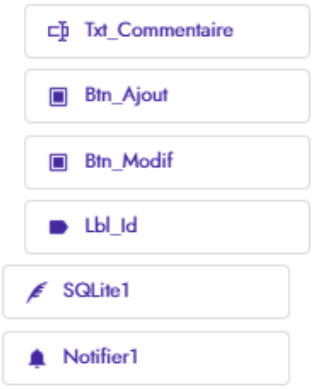
Attention : ces commandes d'insertion sont à exécuter une seule fois pour les données de démonstration. Sinon, les mêmes livres seront ajoutés à nouveau.





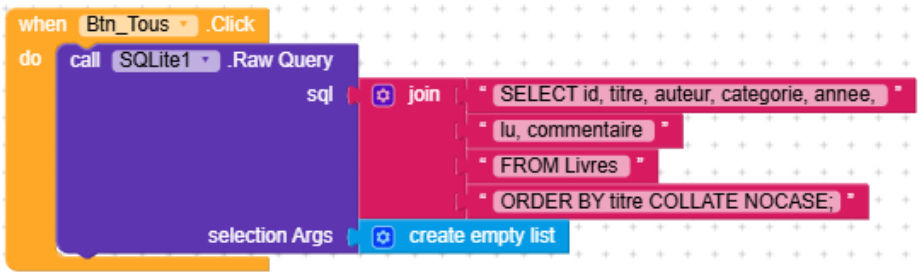
3. Construire l'écran dans Kodular

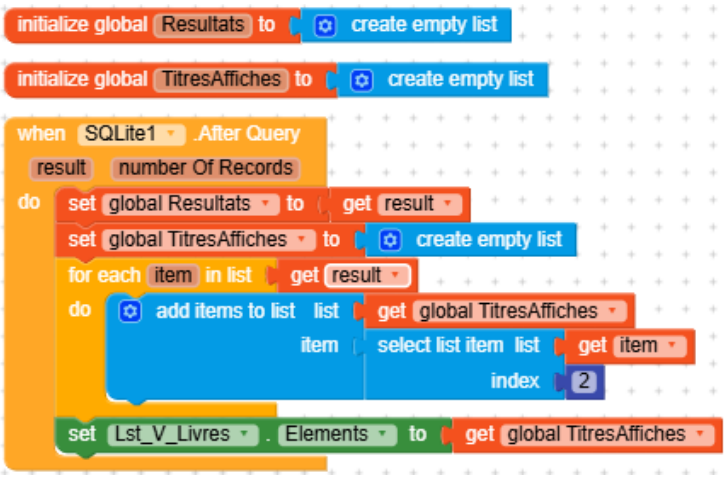




- Bouton *Ajout 3 Livres* afin d'ajouter les 3 livres après la création de la table
- Text Box *Titre ou auteur* et bouton *Rechercher* qui permettent de rechercher et d'afficher dans la liste les titres des livres correspondants au critère de recherche encodé
- Bouton *Tous les livres* afin d'afficher la liste complète des titres de tous les livres de la DB
- La Liste des titres des livres
  Si on clique sur un de ses éléments, tous les détails du livre sélectionné doivent apparaître dans les zones en-dessous
- Les boutons *Ajouter* et *Modifier* permettent d'ajouter un nouveau livre ou de modifier un livre existant dans la DB en tenant compte des zones de détail

4. Les blocks pour afficher tous les livres





Le block **Raw Query** exécute cette requête de manière asynchrone. Les résultats

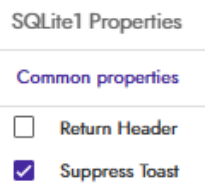
arrivent ensuite dans l'événement **SQLite1.After Query**, avec la liste des résultats et le nombre d'enregistrements.

Pour afficher les résultats dans la **List View**, on crée deux variables globales :

- **global** *Resultats* : la liste des lignes renvoyées par **SQLite**.
- **global** *TitresAffiches* : la liste des titres visibles dans la **List View**.

Dans **when SQLite1.After Query (result, numberOfRecords)** :

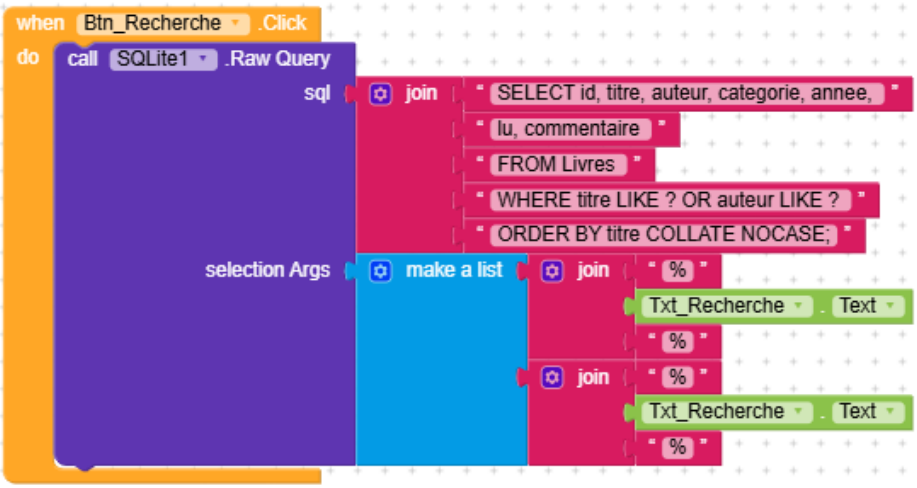
1. Affecte **result** à **global** *Resultats*.
2. Vide **global** *TitresAffiches*.
3. Parcours **result** avec un block **for each item in list**.
4. Pour chaque ligne, récupère le titre, qui est le deuxième élément de la ligne puisque la requête commence par id, titre, auteur…
5. Ajoute ce titre à **global** *TitresAffiches*.
6. Affecte **global** *TitresAffiches* à *Lst_V_Livres.Elements*.



Attention, il faut désactiver la propriété **Return Header** du **SQLite**.

Les résultats sont des listes imbriquées : une liste de livres contenant une liste par livre. Si tu modifies l'ordre des colonnes dans le **SELECT**, il faudra adapter les indices dans les blocks.

5. Les blocks pour rechercher un livre



Le block **Raw Query** exécute cette requête de manière asynchrone. Les résultats arrivent ensuite dans l'événement **SQLite1.After Query**, avec la liste des résultats et le nombre d'enregistrements. Le même **Block** qu'au point précédent sera utilisé pour l'affichage.

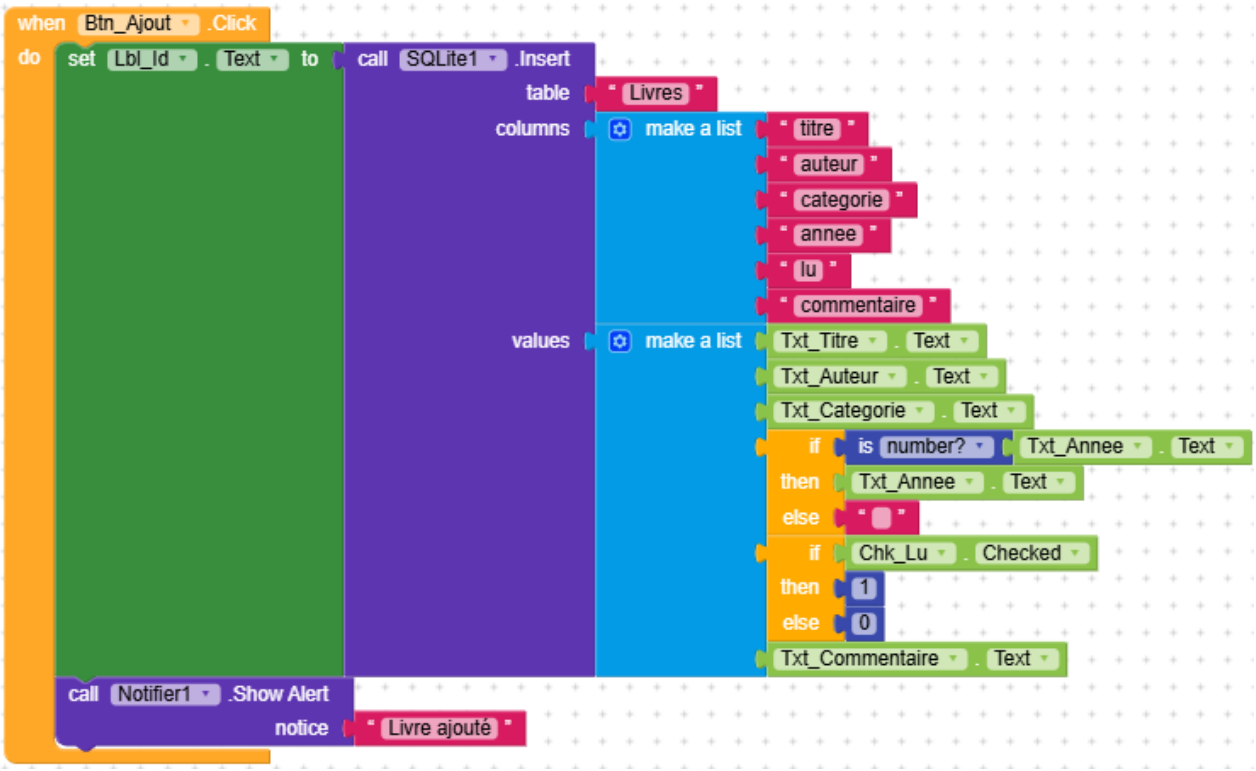
La clause **COLLATE NOCASE** signifie qu'il ne doit pas faire de différence entre les majuscules et les minuscules.

Dans le paramètre **selection Args**, on passe une liste de deux textes :

- *% + txtRecherche.Text + %*
- *% + txtRecherche.Text + %*

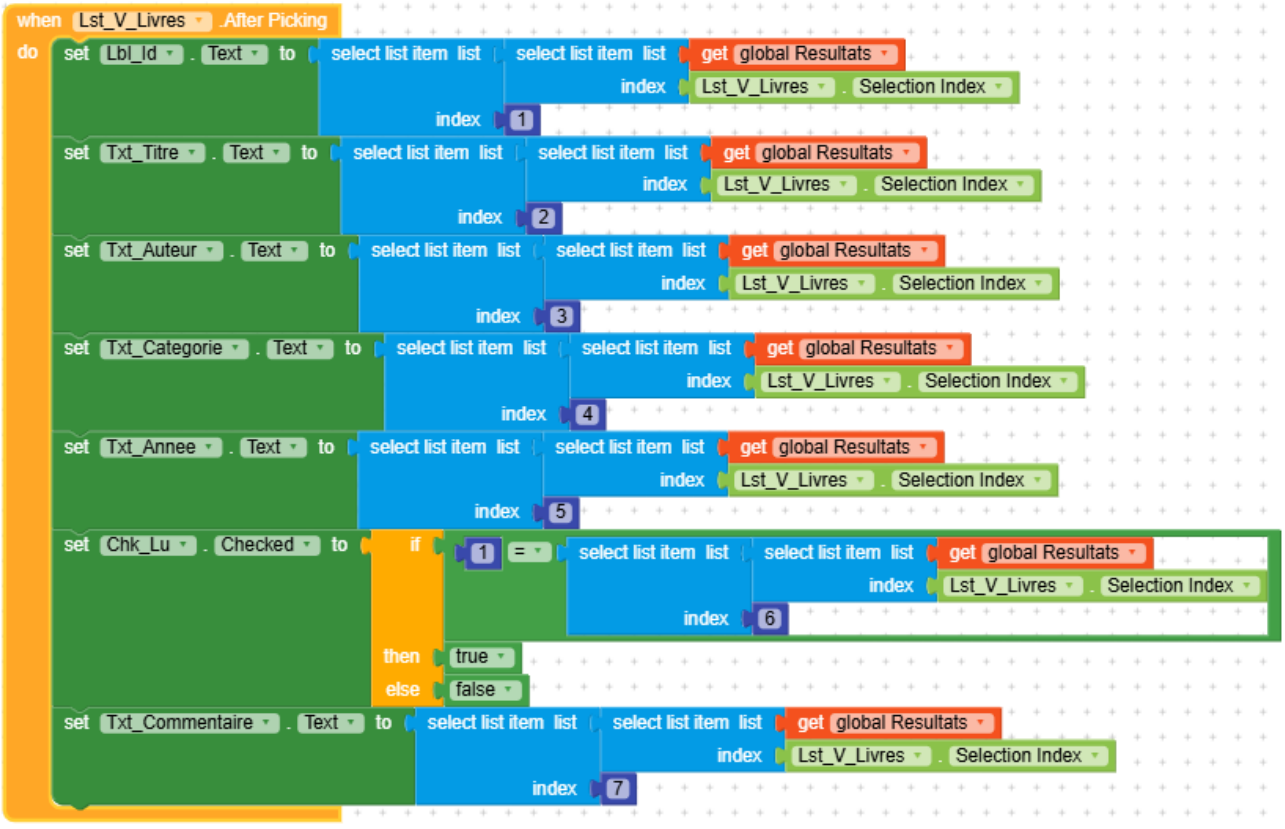
Le caractère % permet de rechercher un mot même s'il est au milieu du titre ou du nom de l'auteur.

6. Ajouter un livre



Vérifie que Txt_Annee.Text contient un nombre valide avant l'insertion.
Le block **Insert** renvoie l'identifiant de la ligne créée.

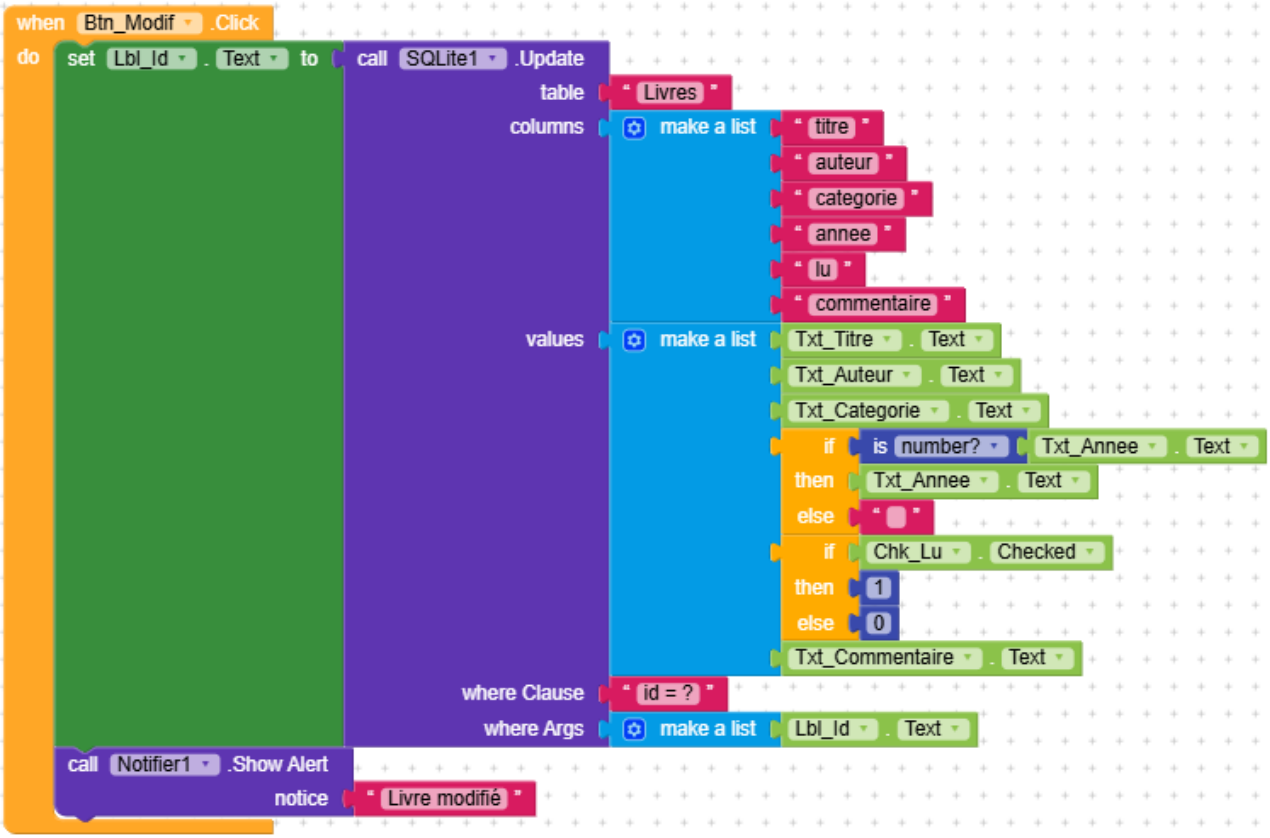
7. Sélectionner un livre dans la liste et en afficher les détails



Dans **when Lst_V_Livres.After Picking** :

1. Récupère l'index sélectionné.
2. Prends la ligne correspondante dans **global** *Resultats*.

8. Modifier le livre actuellement affiché en détail



5. SUPABASE : LA BASE DE DONNEES EN LIGNE

A. OBJECTIFS DE CE CHAPITRE

A la fin de ce chapitre, tu seras capable de :

- Comprendre ce qu'est une Supabase ;
- Savoir dans quels cas utiliser une base de données en ligne ;
- Connecter une application Kodular à Supabase ;
- Enregistrer et récupérer des données sur Internet ;
- Synchroniser automatiquement plusieurs appareils.

B. INTRODUCTION

Jusqu'à présent, toutes les données que nous avons enregistrées étaient stockées sur le téléphone de l'utilisateur. Si un utilisateur crée une liste de tâches sur son smartphone, cette liste reste uniquement sur cet appareil. Mais imaginons une autre situation. Paul crée une liste de courses sur son téléphone. Quelques secondes plus tard, Sophie ouvre la même application sur son propre téléphone. Elle voit immédiatement la liste créée par Paul. Comment est-ce possible ? Les informations ne sont plus stockées uniquement sur un téléphone.

Elles sont enregistrées sur Internet, dans une base de données accessible par tous les appareils autorisés. C'est exactement ce que permet **Supabase.**

Nous allons construire un exemple concret : une mini-application Android appelée *Ma bibliothèque* qui permet d'enregistrer un livre, de rechercher un livre et de modifier son titre. Tu pourras ensuite reprendre les mêmes principes pour ta véritable bibliothèque.

C. COMPRENDRE LE PRINCIPE

Il faut distinguer trois éléments :



Par exemple, lorsque tu appuies sur le bouton *Rechercher*, **Kodular** demande à **Supabase** de trouver le livre voulu. **Supabase** renvoie les résultats que **Kodular** affiche à l'écran.

D. PREMIERE ETAPE : CREER TON PROJET SUPABASE

- Ouvre le site **Supabase** (https://supabase.com) et crée un compte via le bouton **SignIn** (Il affiche parfois des erreurs mais le fait néanmoins).
- Crée un nouveau projet, par exemple *MaBibliotheque*.
- Choisis un mot de passe pour la base de données et conserve-le en lieu sûr.
- Attends que le projet soit prêt. Tu disposeras alors d'une base **PostgreSQL** en ligne.

Dans **Supabase**, tu trouveras notamment le tableau de bord de la base de données, l'éditeur SQL et les paramètres de l'API. Les tables **PostgreSQL** peuvent être interrogées par l'API utilisée par le composant Kodular.

E. CREER NOTRE PREMIERE TABLE : LIVRES

Pour commencer, nous allons utiliser une seule table livres, avec seulement quatre champs :

Champ	Type	Rôle
id	entier	Identifiant unique du livre
titre	texte	Titre du livre
auteur	texte	Nom de l'auteur
annee	entier	Année de publication

id	titre	auteur	annee
1	Le Petit Prince	Saint-Exupéry	1943
2	Vingt mille lieues sous les mers	Jules Verne	1870

Comment créer la table ?

Dans le tableau de bord **Supabase** :

- Ouvre **SQL Editor**.
- Crée une nouvelle requête.
- Copie le code **SQL** :

```
create table public.livres (
    id bigint generated by default as identity primary key,
    titre text not null,
    auteur text,
    annee integer
);
```

- Clique sur **Run**.

Ce code crée la table et configure *id* pour générer automatiquement un identifiant unique pour chaque livre.

Important : la table existe maintenant, mais cela ne signifie pas que ton application **Kodular** a déjà le droit de la consulter. Il faudra configurer les autorisations avant de faire les essais.

F. PREPARER LES AUTORISATIONS

Supabase protège l'accès aux données avec les permissions SQL et les règles de sécurité RLS (Row Level Security). Ces règles déterminent qui peut consulter ou modifier les lignes.

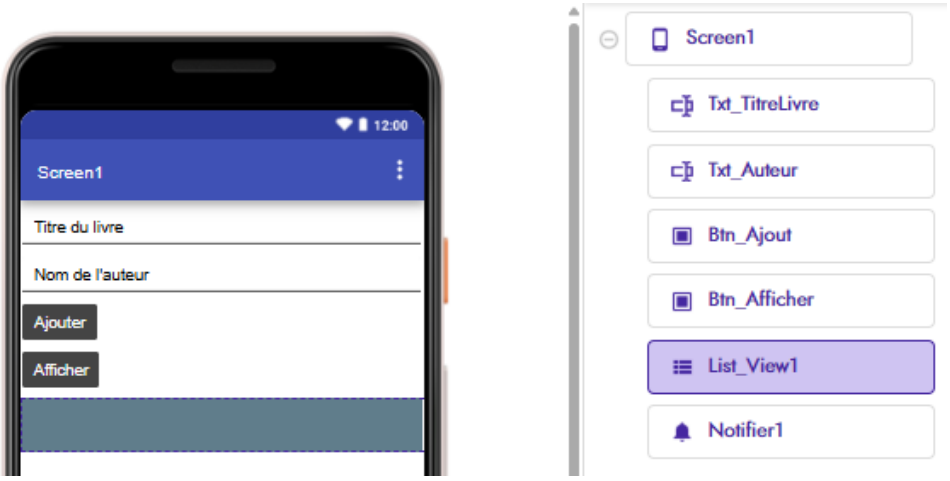
Pour notre premier essai, je te conseille de garder les données de démonstration séparées de ta véritable bibliothèque et de ne pas ouvrir les droits d'écriture à tout le monde.

Nous configurerons les permissions adaptées au mode d'accès choisi avant de connecter **Kodular**. La clé publique de l'API ne remplace pas ces autorisations et une clé secrète ne doit jamais être intégrée à l'application Android.

G. CONSTRUIRE L'ECRAN DANS KODULAR

Ouvre **Kodular** et crée un projet nommé MaBibliotheque.

Dans le **Designer**, ajoute les composants suivants :



Ajoute ensuite le composant non visible **Supabase Database**, dans la catégorie **Storage** si elle est disponible dans ta version de **Kodular**. Ce composant réalise les opérations de base de données par l'**API REST** de **Supabase**.

H. RELIER KODULAR A SUPABASE

Sélectionne le composant **Supabase Database** dans le **Designer**.

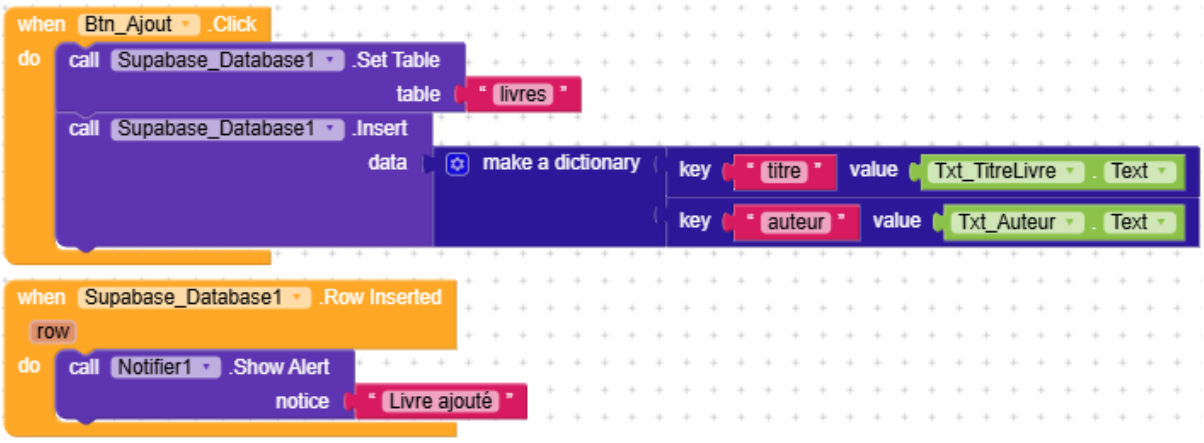
Configure ses propriétés :

Propriété	**Valeur à renseigner**
Default Schema	public
Project URL	L'URL de ton projet Supabase
Publishable API Key	La clé publique de ton projet

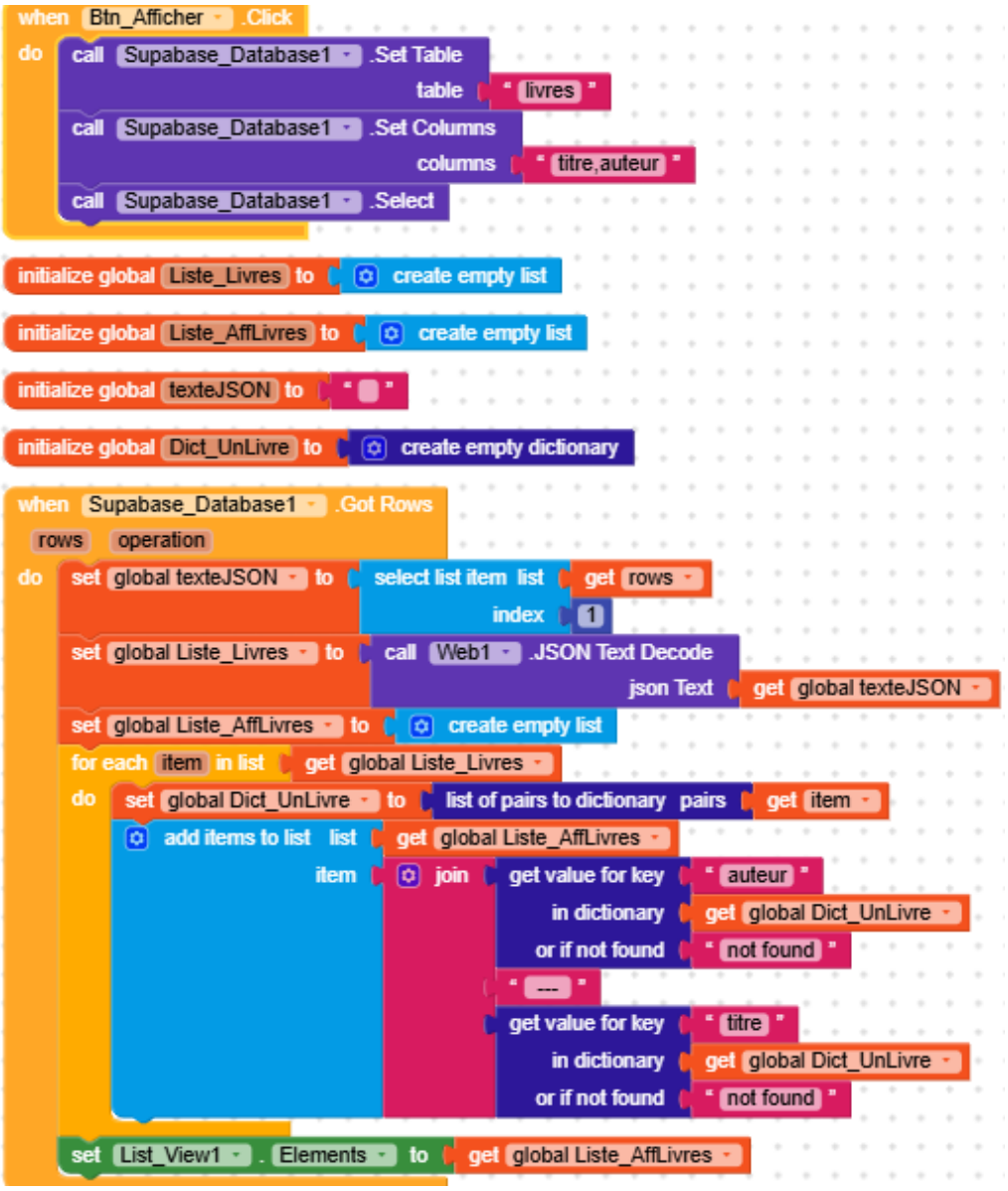
Tu trouveras l'URL et la clé publique dans les informations de connexion ou d'API du tableau de bord **Supabase**. Ne mets pas de clé secrète dans **Kodular**.

I. COMPRENDRE LES BLOCS AVEC TROIS OPERATIONS

Exemple A : Ajouter un livre (avec le titre et l'auteur saisis dans les **TextBox**)



Exemple B : Afficher l'auteur et le titre des livres quand tu appuies sur le bouton *Afficher*



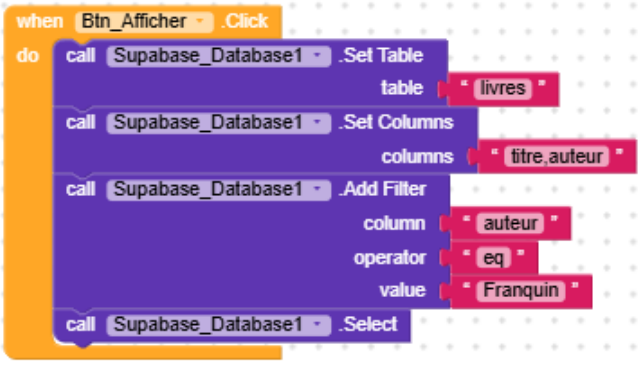
La commande **Supabase.Set Columns** permet de dire les champs que l'on souhaite récupérer de la table.

La commande **Select** permet de lire la table en tenant compte des paramètres précédemment fixés. Elle génère ensuite l'événement **Got Rows.**

La commande **Got Rows** renvoie une liste d'un élément reprenant tous les enregistrements résultats du query exécuté. Cet élément est sous le format **JSON.** Ce qui explique la commande **JSON Text Decode.** Celle-ci renvoie une liste que l'on doit transformer en dictionnaire via la commande **list of pairs to dictionary pairs** afin de pouvoir rechercher les éléments souhaités grâce à leur clé.

Le composant **Web** de **Kodular** possède des blocs de décodage **JSON** adaptés à ce traitement.

Remarquons que l'on aurait pu également appliquer une sélection pour éviter d'afficher la liste de tous les livres. L'exemple suivant montre comment n'afficher que les livres de Franquin :

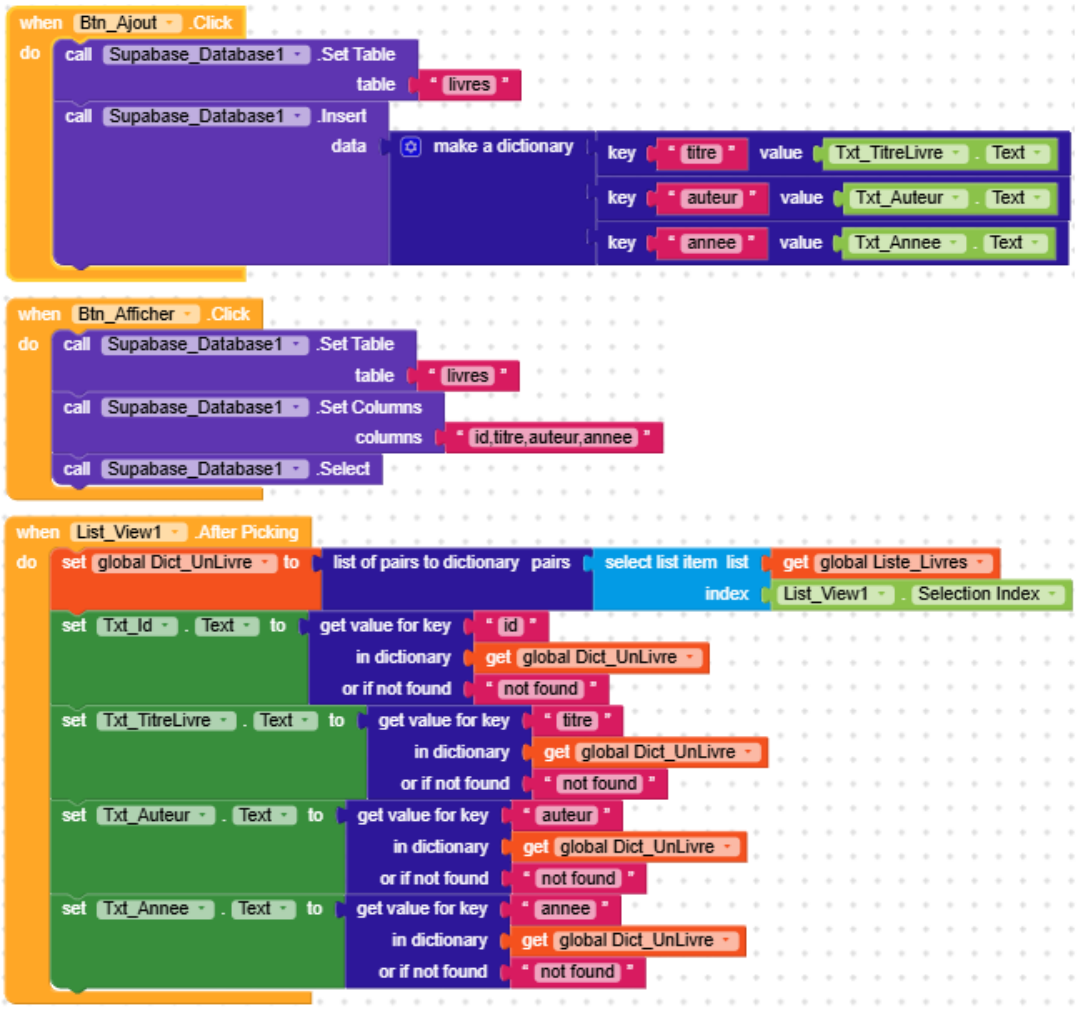


Exemple C : Modifier un livre

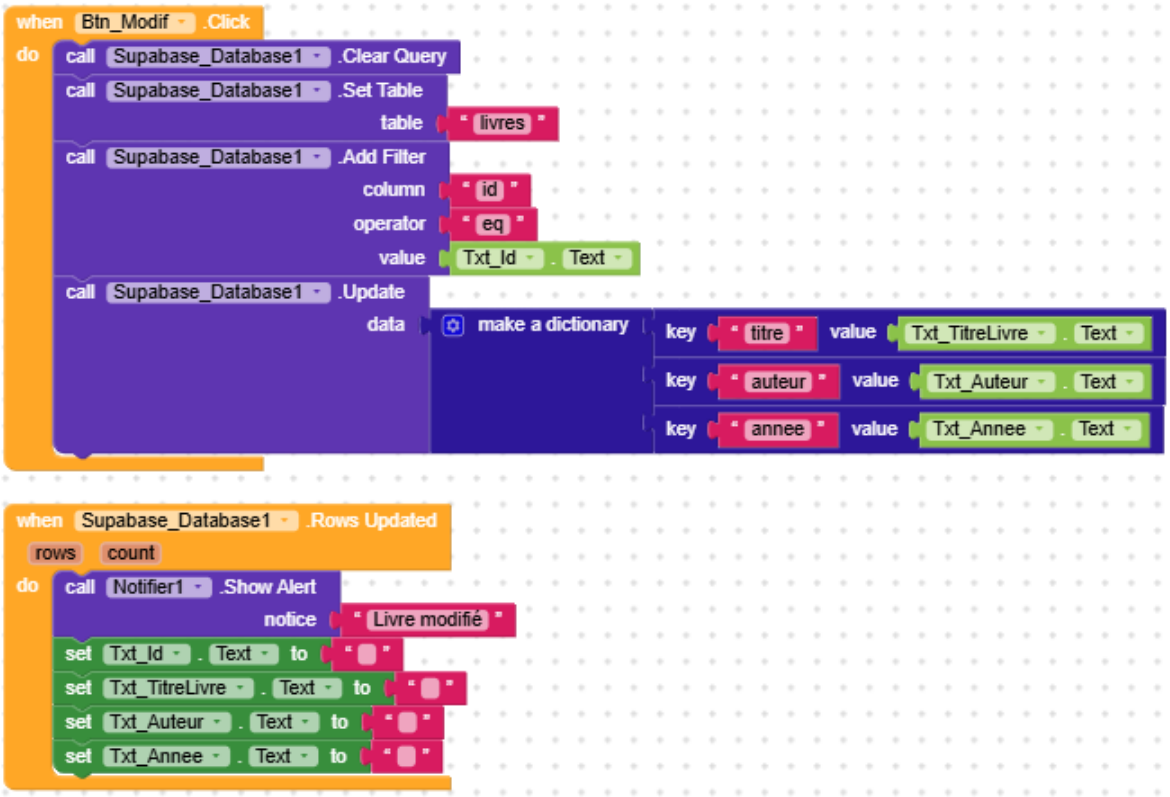
Dans un premier temps, modifie l'écran :

- Ajoute le bouton *Modifier*
- Ajoute les zones *Id* (non modifiable) et *Année* à l'écran

Ensuite, modifie les **Blocks** afin que l'utilisateur puisse sélectionner le livre à modifier au moyen de la liste des livres affichée et des zones de texte.



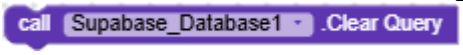
Et enfin, fais-en sorte qu'il puisse modifier l'auteur, le titre et/ou l'année.



La commande **Clear Query** permet d'effacer les anciens filtres.

Le filtre est essentiel car sans lui, tu risques de modifier plusieurs livres.

J. CE QU'IL FAUT RETENIR

- **Supabase** conserve les données en ligne.
- Les tables **SQL** organisent les livres, auteurs et catégories.
- **Kodular** fournit l'interface et les blocs.
- Le composant **Supabase Database** effectue les opérations.
- Dans certains cas, il faudra exécuter la commande **Clear Query** avant de définir une nouvelle requête → 

- Les autorisations et les règles RLS contrôlent qui peut lire ou modifier les données.

Pour ta bibliothèque définitive, nous pourrons ensuite séparer les livres, les auteurs et les catégories en plusieurs tables reliées par des identifiants. Cela évitera de répéter les mêmes informations et permettra des recherches plus riches.

K. QUE FAIRE SI LA LISTE RESTE VIDE ?

Voici les vérifications à effectuer dans l'ordre :

1. La table contient-elle bien des lignes ?
2. L'URL du projet et la clé publique sont-elles correctes ?
3. La lecture est-elle autorisée par les permissions **SQL** et les règles **RLS** ?
4. L'événement **Error Occurred** est-il déclenché ? Si oui, note **operation**, **status Code** et **message**.
5. L'événement **Got Rows** reçoit-il des données ?

Une liste vide ne signifie pas nécessairement que la connexion est en panne : il peut aussi s'agir d'une table vide ou d'un accès refusé.

L. SECURISER LES DONNEES

Toutes les données ne doivent pas être accessibles à tout le monde. Il est souvent nécessaire de définir des règles.

Par exemple :

- Seuls les utilisateurs connectés peuvent lire les données ;
- Seuls les administrateurs peuvent supprimer des informations ;
- Chaque utilisateur ne peut modifier que ses propres données.

Une bonne sécurité est indispensable pour protéger les informations.

M. BONNES PRATIQUES

- Utiliser **Supabase** uniquement lorsque le partage de données est nécessaire.
- Tester l'application avec plusieurs appareils.
- Prévoir une gestion des erreurs réseau.
- Organiser clairement les données.
- Mettre en place des règles de sécurité.
- Sauvegarder régulièrement les données importantes.

N. EXERCICE : CREER UNE LISTE DE TACHES COLLABORATIVE

Conçois une application dans laquelle plusieurs utilisateurs peuvent gérer une même liste de tâches.

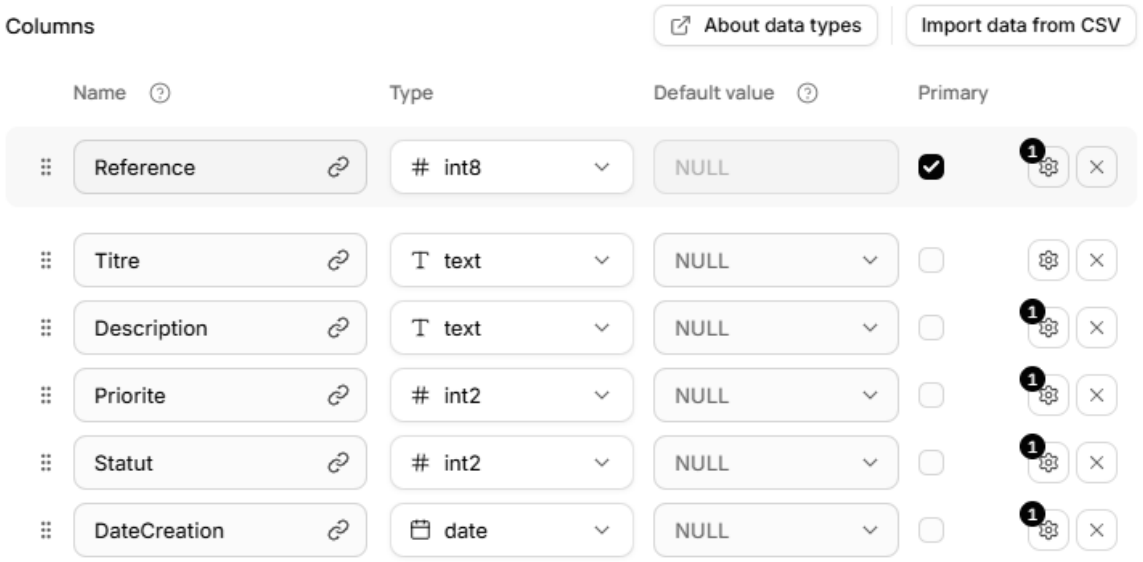
Chaque tâche comprend :

- Un titre ;
- Une description ;
- Un niveau de priorité ;
- Un état (*A faire*, *En cours* ou *Terminé*) ;
- La date de création.

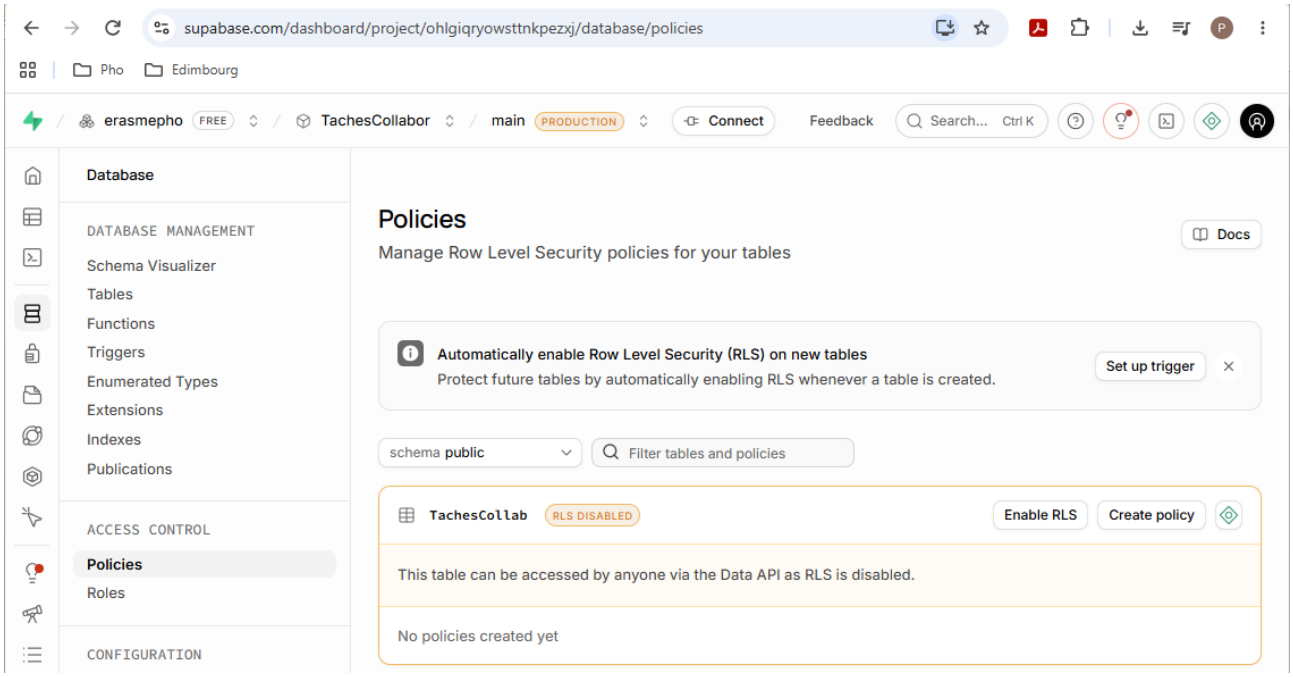
L'application doit permettre :

- D'ajouter une tâche ;
- De modifier une tâche sélectionnée via la liste ;
- De supprimer une tâche ;
- De voir les changements apparaître sur tous les appareils connectés lors de chaque demande d'actualisation de la liste.

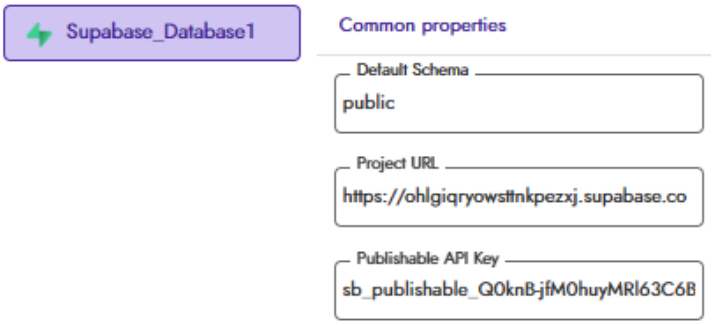
→ Création de la DB TachesCollabor et de la table TachesCollab dans Supabase



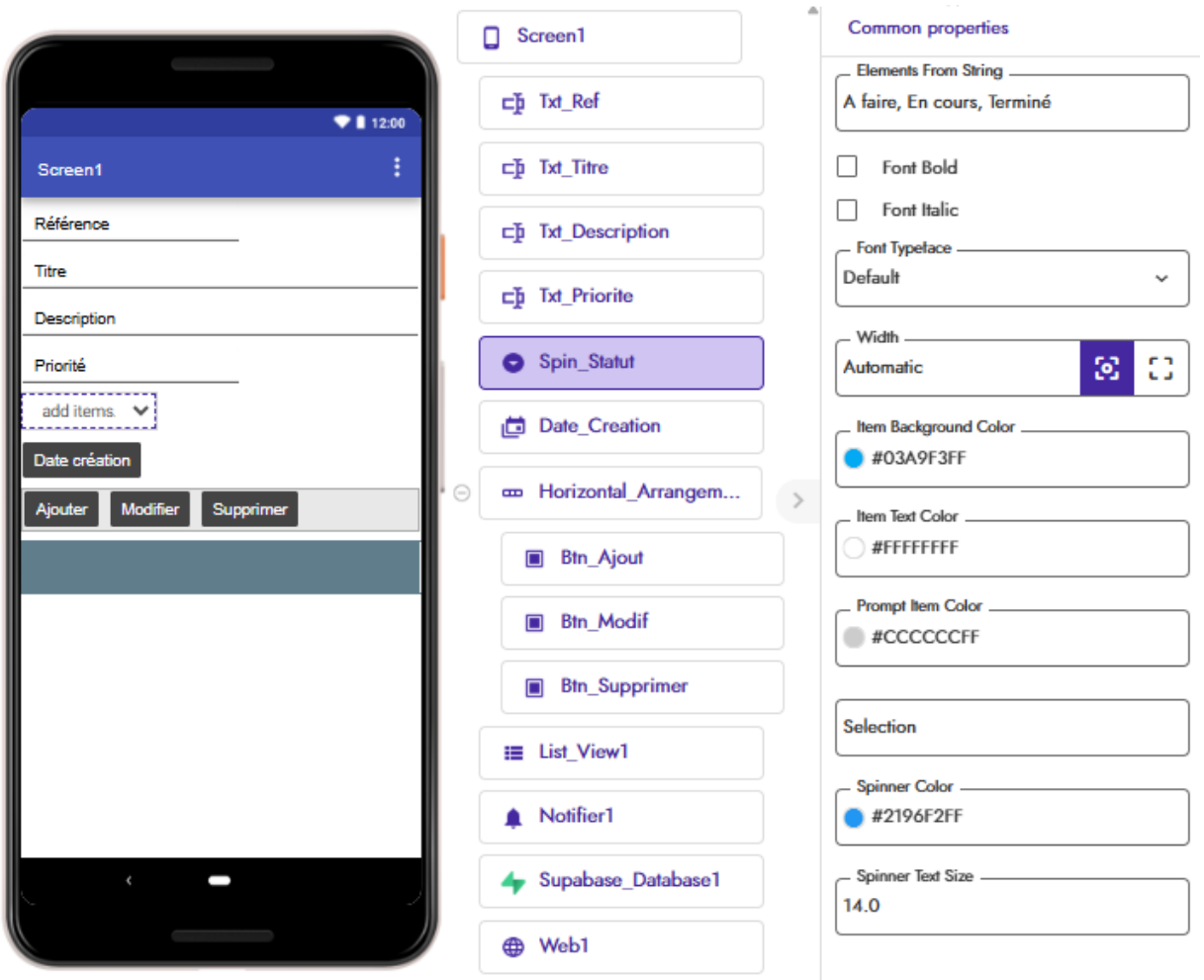
→ **Disable RLS** dans Supabase afin de pouvoir modifier la DB à partir de Kodular



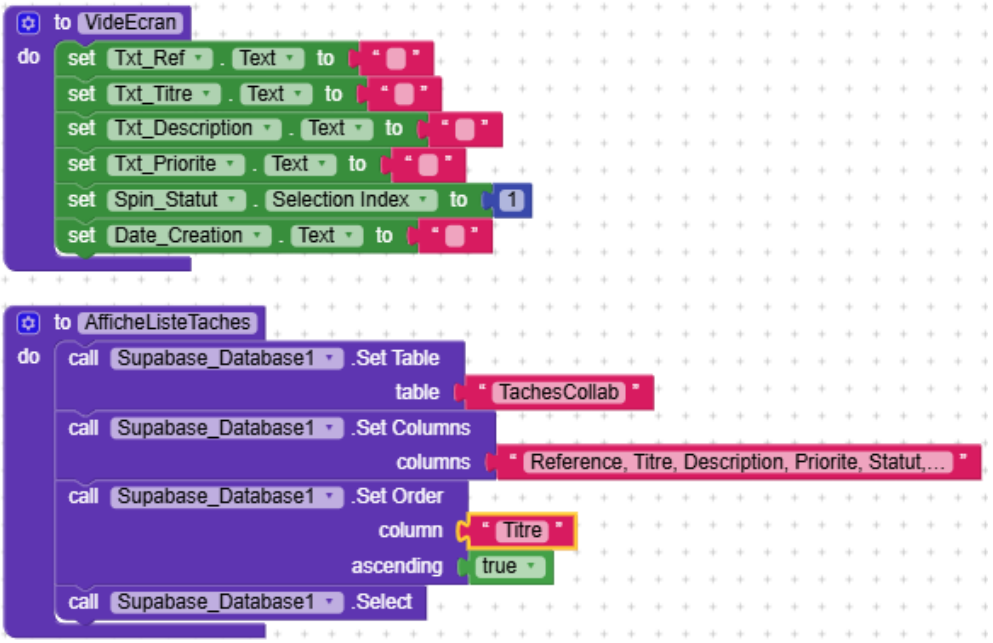
→ Ne pas oublier de copier les liens Project URL et API Key dans le designer



→ Designer



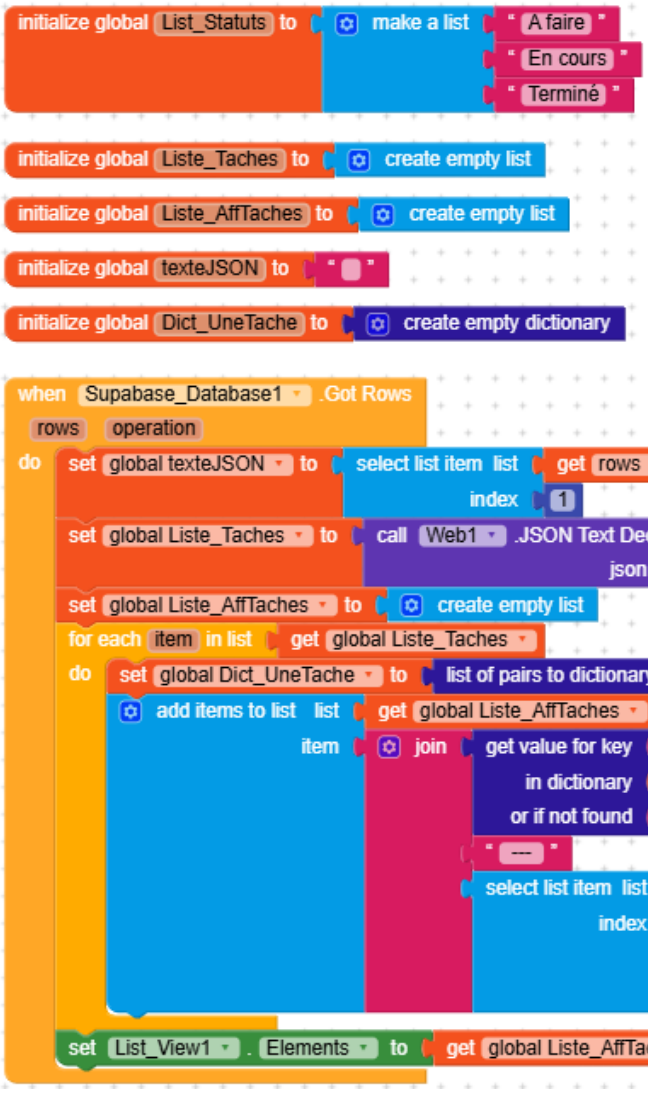
→ Blocks : Procédures



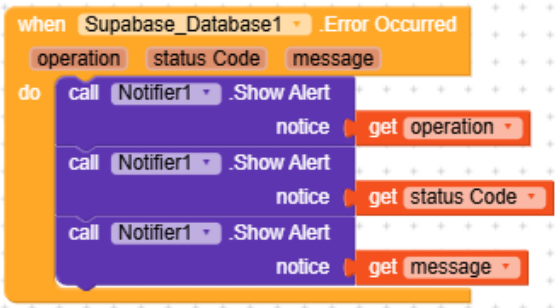
→ Blocks : Initialisation de l'écran



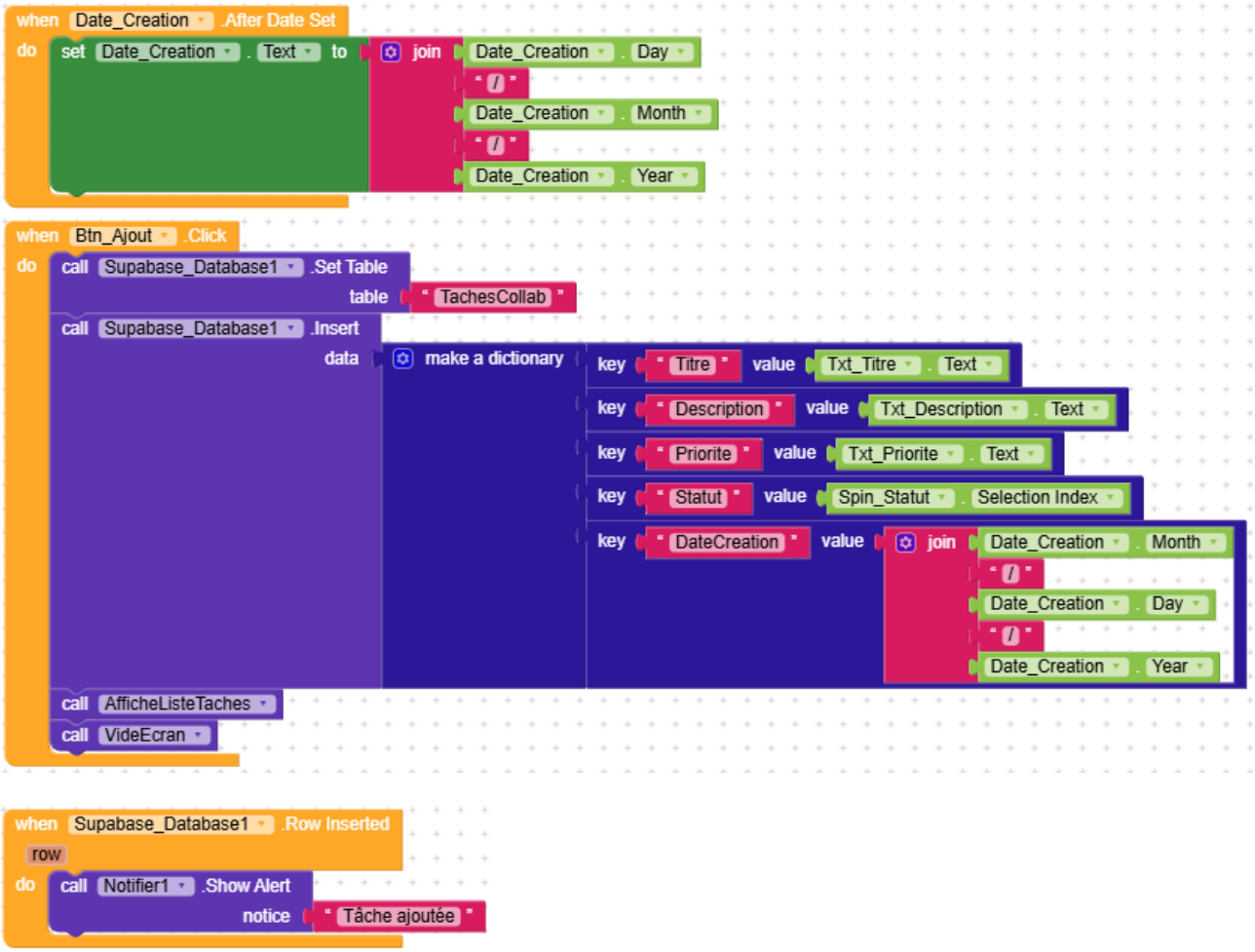
→ Blocks : Affichage de la liste complète



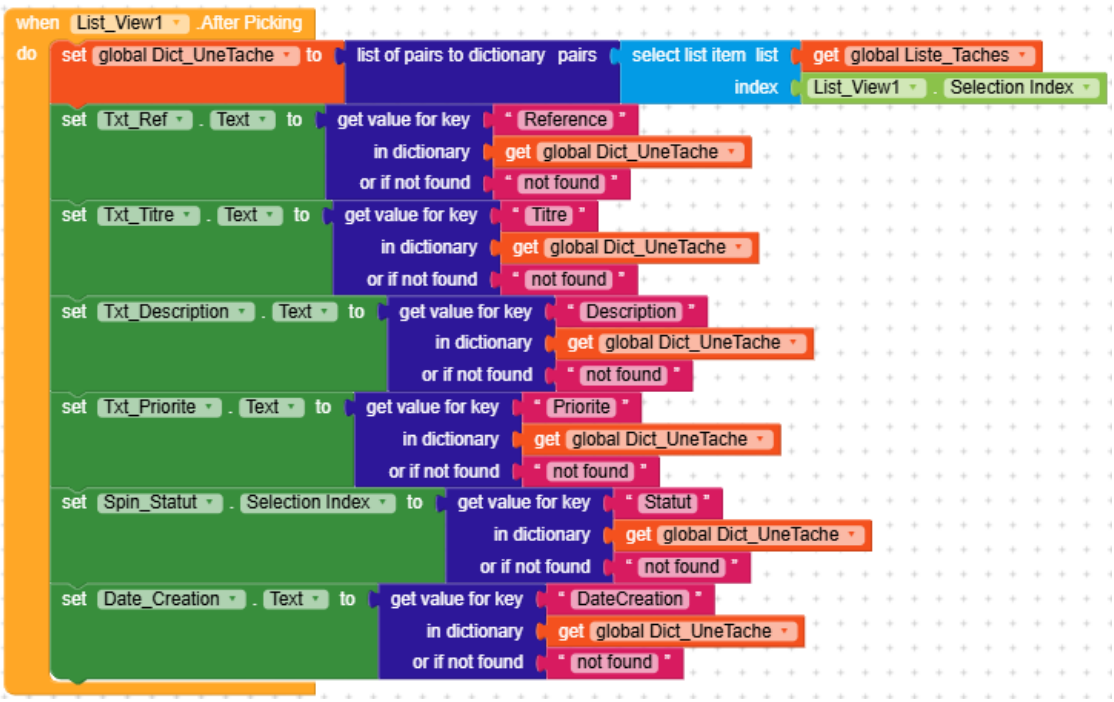
→ Blocks : Affichage d'erreur d'exécution liée à Supabase



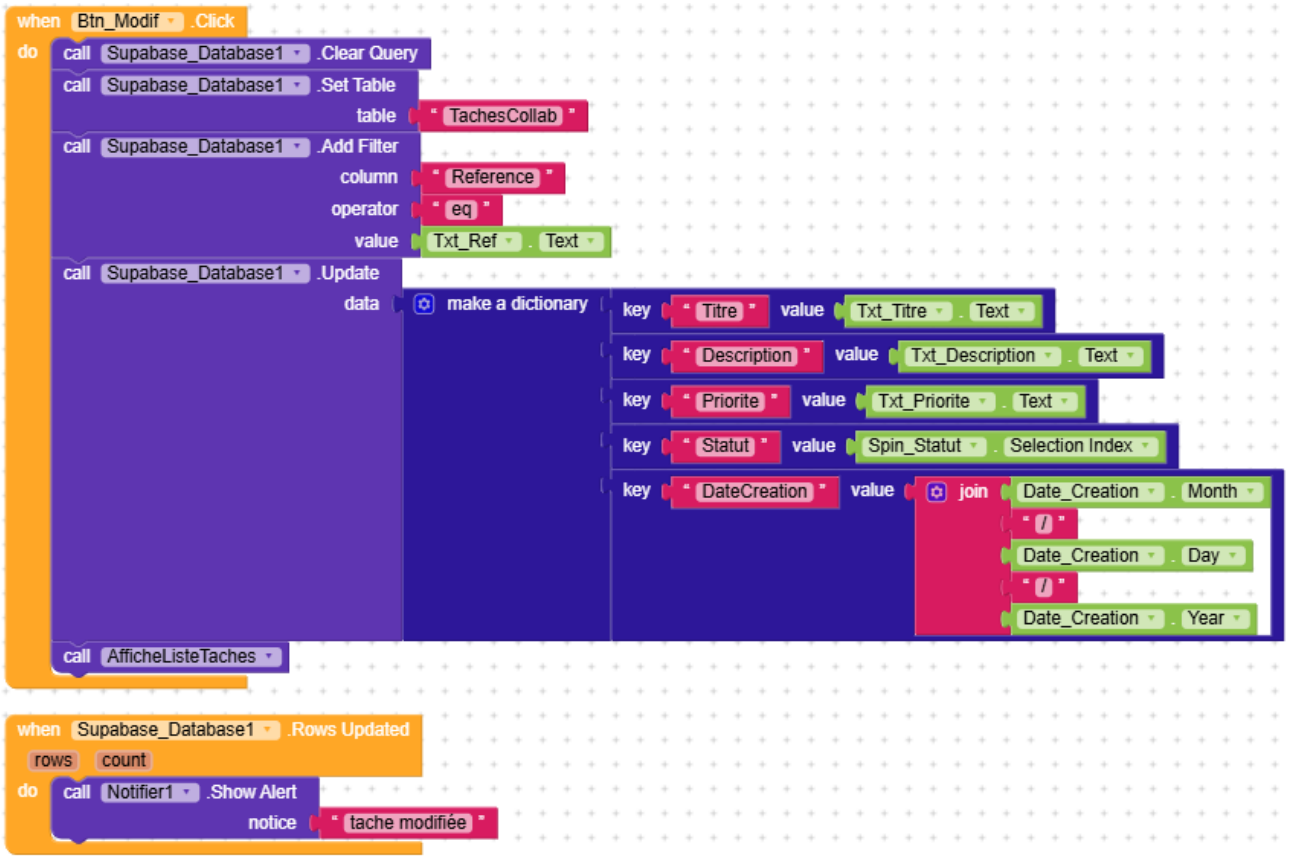
→ Blocks : Ajout d'une tâche



→ Blocks : Sélection d'une tâche dans la liste



→ Blocks : Modification d'une tâche



→ Blocks : Suppression d'une tâche

