

RIAHI Anas
3L BIS 1

Développement Mobile

Gestion de Production

<https://www.linkedin.com/in/anas-riahi-6602a8238/>



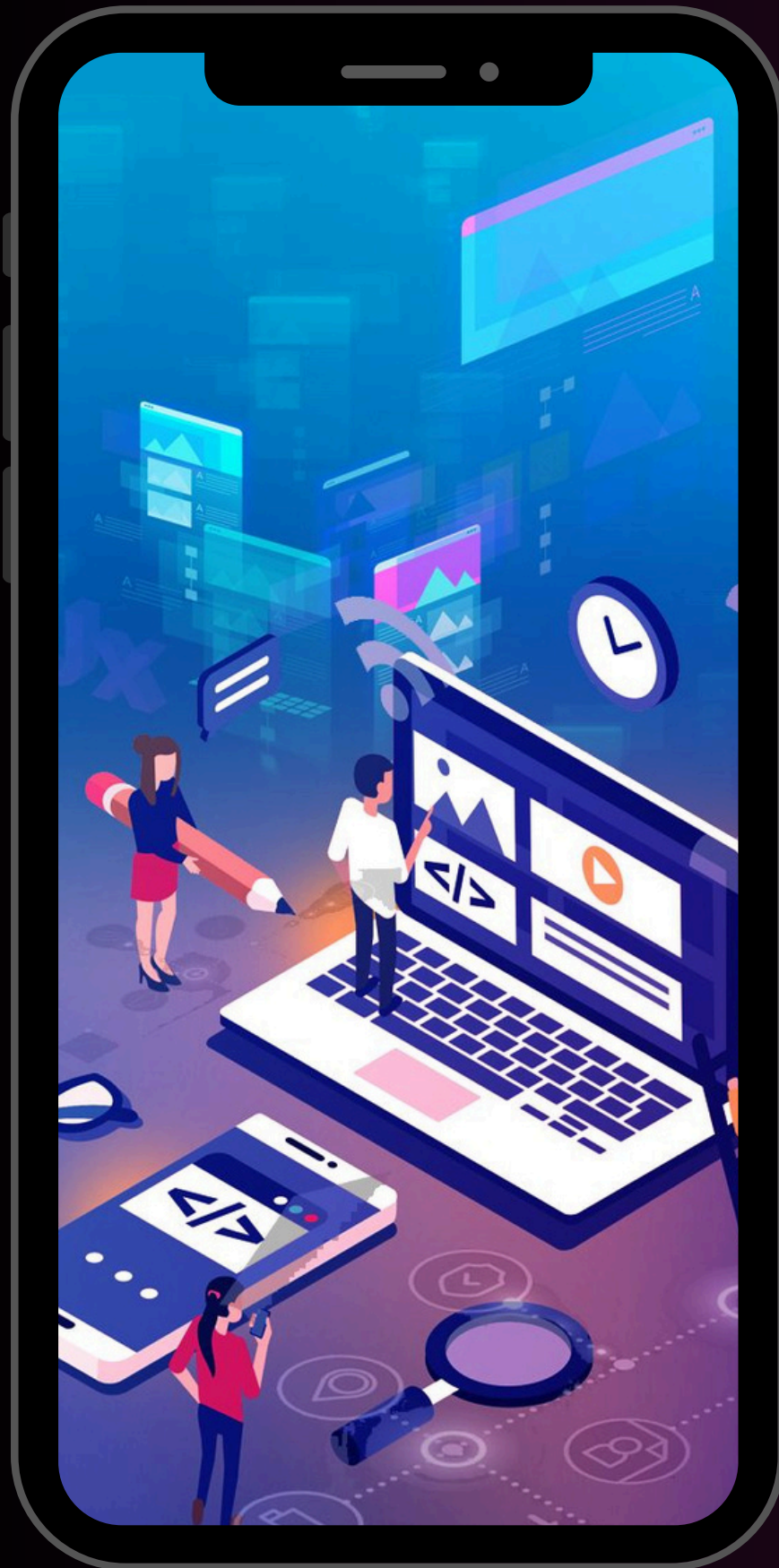
Introduction

Le présent projet consiste à développer une application mobile Android de gestion de production destinée à faciliter le suivi et l'organisation des ressources au sein d'une entreprise industrielle ou artisanale.

L'objectif principal de cette application est de permettre à l'utilisateur de gérer les matières premières, les produits, les catégories, les compositions ainsi que les opérations de production, tout en assurant un suivi clair du stock et des productions réalisées.

Ce projet s'inscrit dans une démarche d'apprentissage du développement mobile Android en utilisant des technologies modernes telles que Kotlin, Room Database, et une architecture claire séparant les données, la logique métier et l'interface utilisateur.





● Objectifs du projet

- **Centraliser la gestion des matières premières**
- **Gérer les produits finis et leurs compositions**
- **Suivre les opérations de production**
- **Détecter les stocks faibles**
- **Offrir une interface simple, ergonomique et professionnelle**
- **Sécuriser l'accès à l'application via un système d'authentification (Login / Sign Up)**

Technologies utilisées

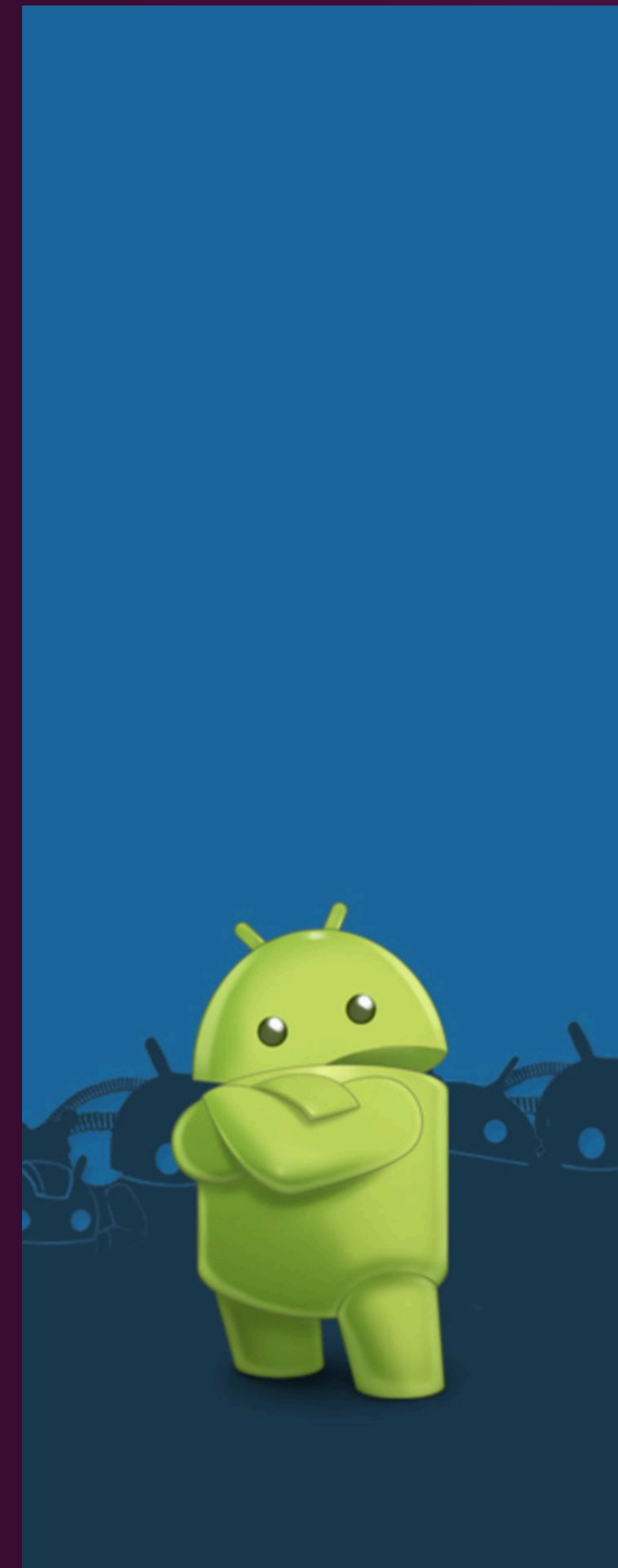
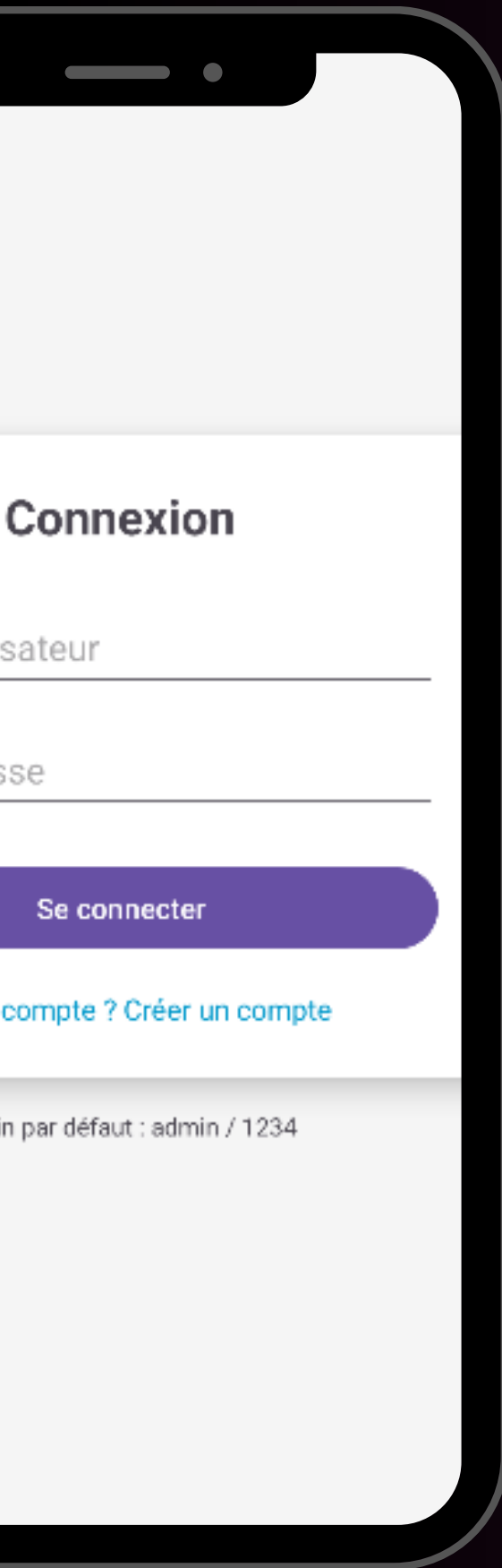
Android Studio : environnement de développement

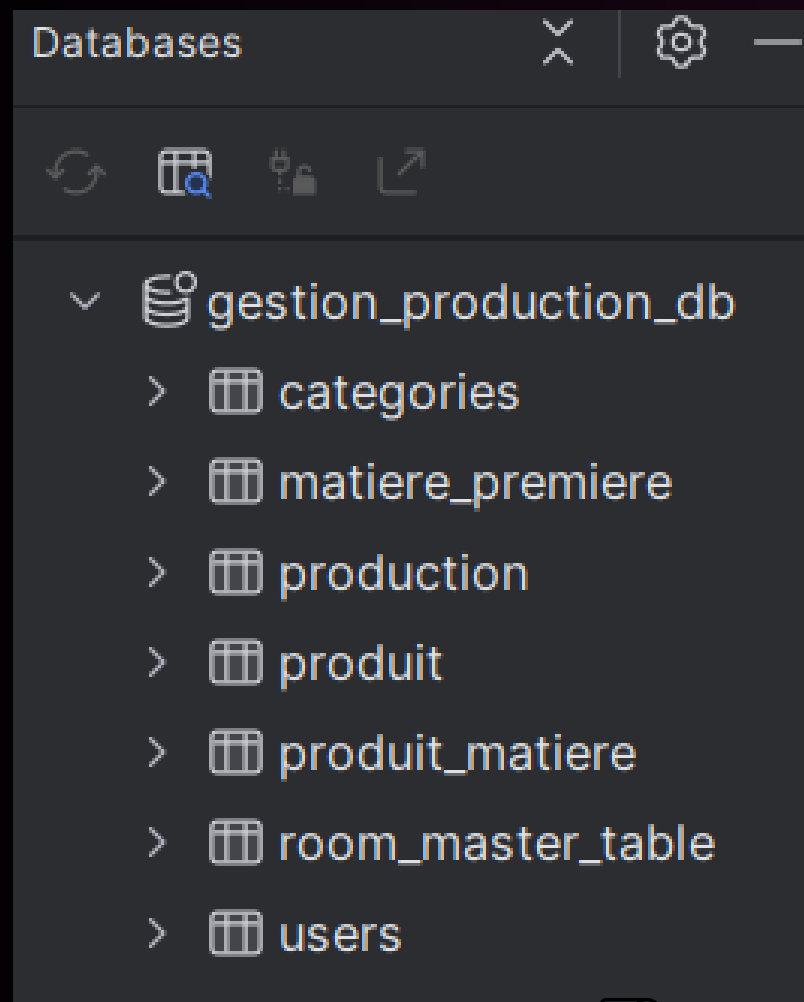
Kotlin : langage de programmation principal

Room Database (SQLite) : gestion de la base de données locale

XML : conception des interfaces graphiques

Architecture DAO / Entity : organisation des données





Architecture de l'application



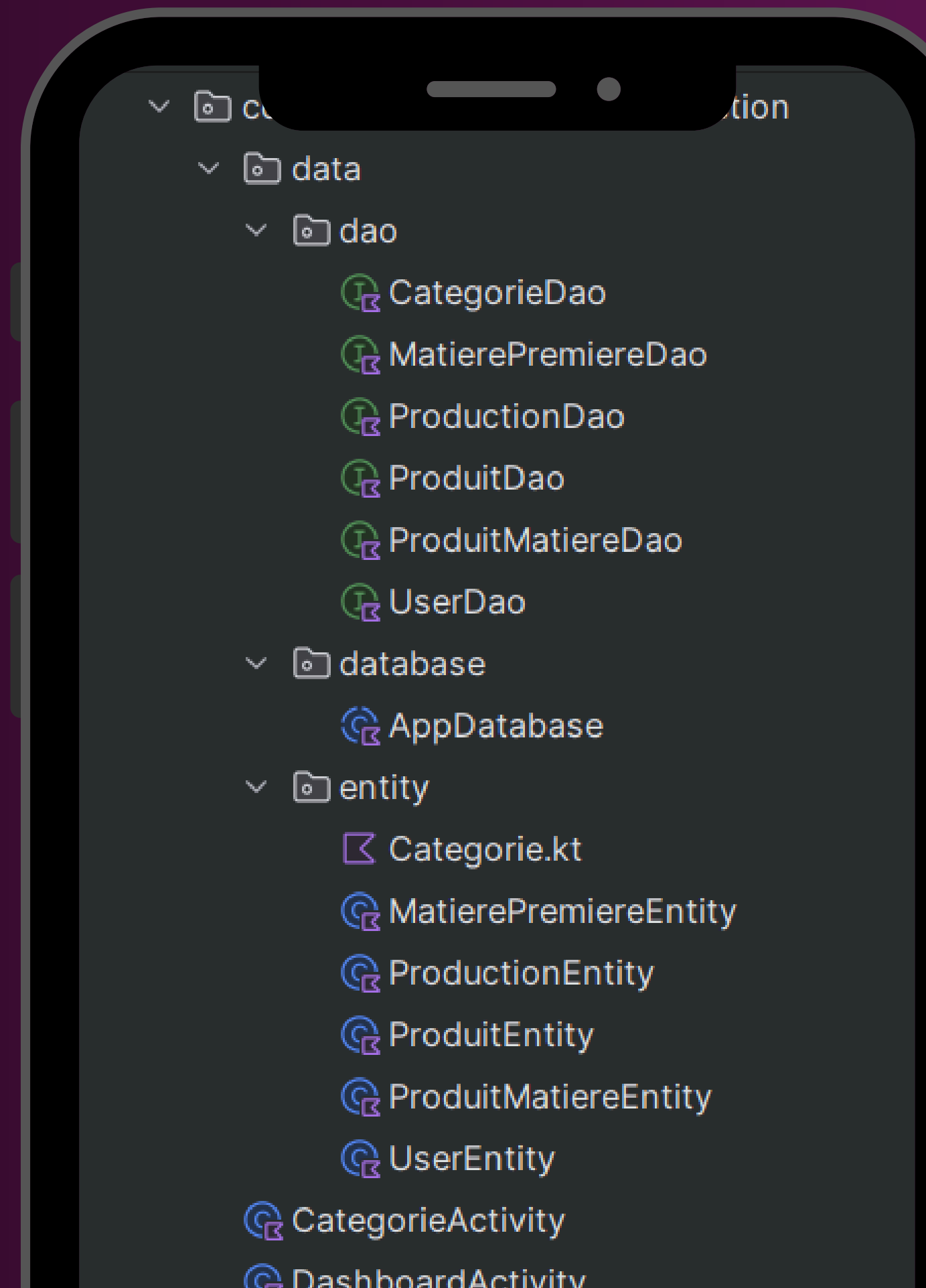
L'application repose sur une architecture bien structurée composée de :

1. Entities

Les entités représentent les tables de la base de données :

- CategorieEntity
- MatierePremiereEntity
- ProduitEntity
- ProduitMatiereEntity
- ProductionEntity
- UserEntity

Chaque entité correspond à une table stockée localement à l'aide de Room



DAO (Data Access Object)

Les DAO permettent d'interagir avec la base de données en fournissant des méthodes telles que :

- insertion
- suppression
- mise à jour
- recherche

- **Exemples**

- CategorieDao
- ProduitDao
- UserDao

```
1 package com.example.gestiondeproduction.data.dao
2
3 > import ...
4
5 6 Usages 1 Implementation
6 @Dao
7 interface CategorieDao {
8
9     1 Implementation
10    @Insert
11    fun insert(categorie: CategorieEntity)
12
13    1 Implementation
14    @Query(value = "SELECT * FROM categories")
15    fun getAll(): List<CategorieEntity>
16
17    1 Implementation
18    @Delete
19    fun delete(categorie: CategorieEntity)
20
21    1 Implementation
22    @Update
23    fun update(categorie: CategorieEntity)
24
25 }
```

CategorieDao

```

package com.example.gestiondeproduction.data.dao

import ...

6 Usages 1 Implementation
@Dao
interface ProduitDao {

    1 Implementation
    @Query(value = "SELECT * FROM produit WHERE categorieId = :categorieId")
    fun getByCategorie(categorieId: Int): List<ProduitEntity>

    1 Implementation
    @Query(value = "SELECT * FROM produit")
    fun getAll(): List<ProduitEntity>

    1 Implementation
    @Insert
    fun insert(produit: ProduitEntity)

    1 Implementation
    @Delete
    fun delete(produit: ProduitEntity)
}

```

ProduitDao

```

package com.example.gestiondeproduction.data.dao

import androidx.room.*
import com.example.gestiondeproduction.data.entity.UserEntity

6 Usages 1 Implementation
@Dao
interface UserDao {

    1 Implementation
    @Insert
    fun insert(user: UserEntity)

    2 Usages 1 Implementation
    @Query(value = """
        SELECT * FROM users
        WHERE username = :username AND password = :password
        LIMIT 1
    """)
    fun login(username: String, password: String): UserEntity?
}

```

UserDao

Base de données (Room)

La classe AppDatabase centralise toutes les entités et les DAO.

Room assure :

- la sécurité des requêtes SQL
- la cohérence des données
- une meilleure performance

```
10 @Database(
11     entities = [
12         CategorieEntity::class,
13         MatierePremiereEntity::class,
14         ProduitEntity::class,
15         ProduitMatiereEntity::class,
16         ProductionEntity::class,
17         UserEntity::class
18     ],
19     version = 7,
20     exportSchema = false
21 )
22 abstract class AppDatabase : RoomDatabase() {
23
24     7 Usages 1 Implementation
25     abstract fun categorieDao(): CategorieDao
26     7 Usages 1 Implementation
27     abstract fun matierePremiereDao(): MatierePremiereDao
28     7 Usages 1 Implementation
29     abstract fun produitDao(): ProduitDao
30     5 Usages 1 Implementation
31     abstract fun produitMatiereDao(): ProduitMatiereDao
32     4 Usages 1 Implementation
33     abstract fun productionDao(): ProductionDao
34     4 Usages 1 Implementation
```

AppDatabase

```
@Volatile
private var INSTANCE: AppDatabase? = null

fun getDatabase(context: Context): AppDatabase {
    return INSTANCE ?: synchronized(lock = this) {
        val instance = Room.databaseBuilder(
            context.applicationContext,
            AppDatabase::class.java,
            "gestion_production_db"
        )
        .fallbackToDestructiveMigration()
        .build()
        INSTANCE = instance
        instance
    }
}
```

Fonctionnalités principales

- 🔑 1. AUTHENTIFICATION (LOGIN / SIGN UP)

L'application propose un système d'authentification permettant

- L'application propose un système d'authentification permettant
- la connexion sécurisée (Login)

Un compte administrateur par défaut est disponible pour l'accès initial.

Sign Up

Créer un compte

Nom d'utilisateur

Mot de passe

Confirmer mot de passe

Créer compte

Connexion

Nom d'utilisateur

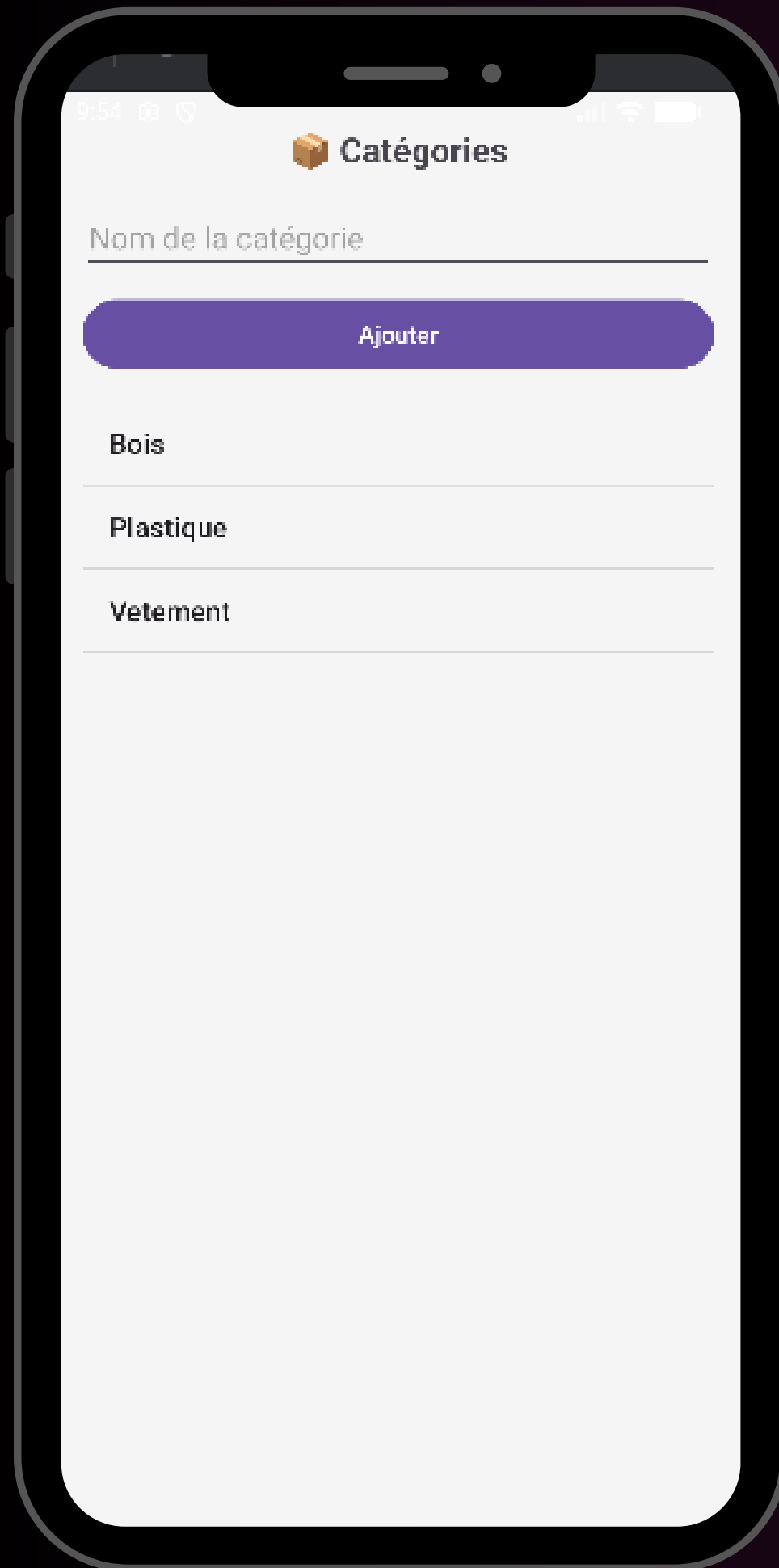
Mot de passe

Se connecter

[Pas de compte ? Créer un compte](#)

Admin par défaut : admin / 1234

Login



Gestion des catégories

Cette fonctionnalité permet :

- l'ajout de catégories
- l'affichage de la liste des catégories
- la suppression ou modification si nécessaire

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_categorie)  
    db = AppDatabase.getDatabase( context = this)  
    listView = findViewById( id = R.id.listViewCategories)  
    etNom = findViewById( id = R.id.etCategorie)  
    val btnAdd = findViewById<Button>( id = R.id.btnAdd)  
  
    loadCategories()  
  
    btnAdd.setOnClickListener {  
        val nom = etNom.text.toString()  
        if (nom.isNotEmpty()) {  
            Thread {  
                db.categorieDao().insert(  
                    CategoryEntity(nom = nom)  
                )  
                runOnUiThread {  
                    etNom.text.clear()  
                    loadCategories() // 🔄  
                }  
            }.start()  
        }  
    }  
}
```

2 Usages

```
private fun loadCategories() {  
    Thread {  
        val list = db.categorieDao().getAll()  
        runOnUiThread {  
            listView.adapter = ArrayAdapter(  
                context = this,  
                resource = android.R.layout.simple_list_item_1,  
                objects = list.map { it.nom }  
            )  
        }  
    }.start()  
}
```

Categories.Activity

Gestion des matières premières

L'utilisateur peut :

- ajouter des matières premières
- définir les quantités disponibles
- consulter les stocks
- détecter les matières à stock faible



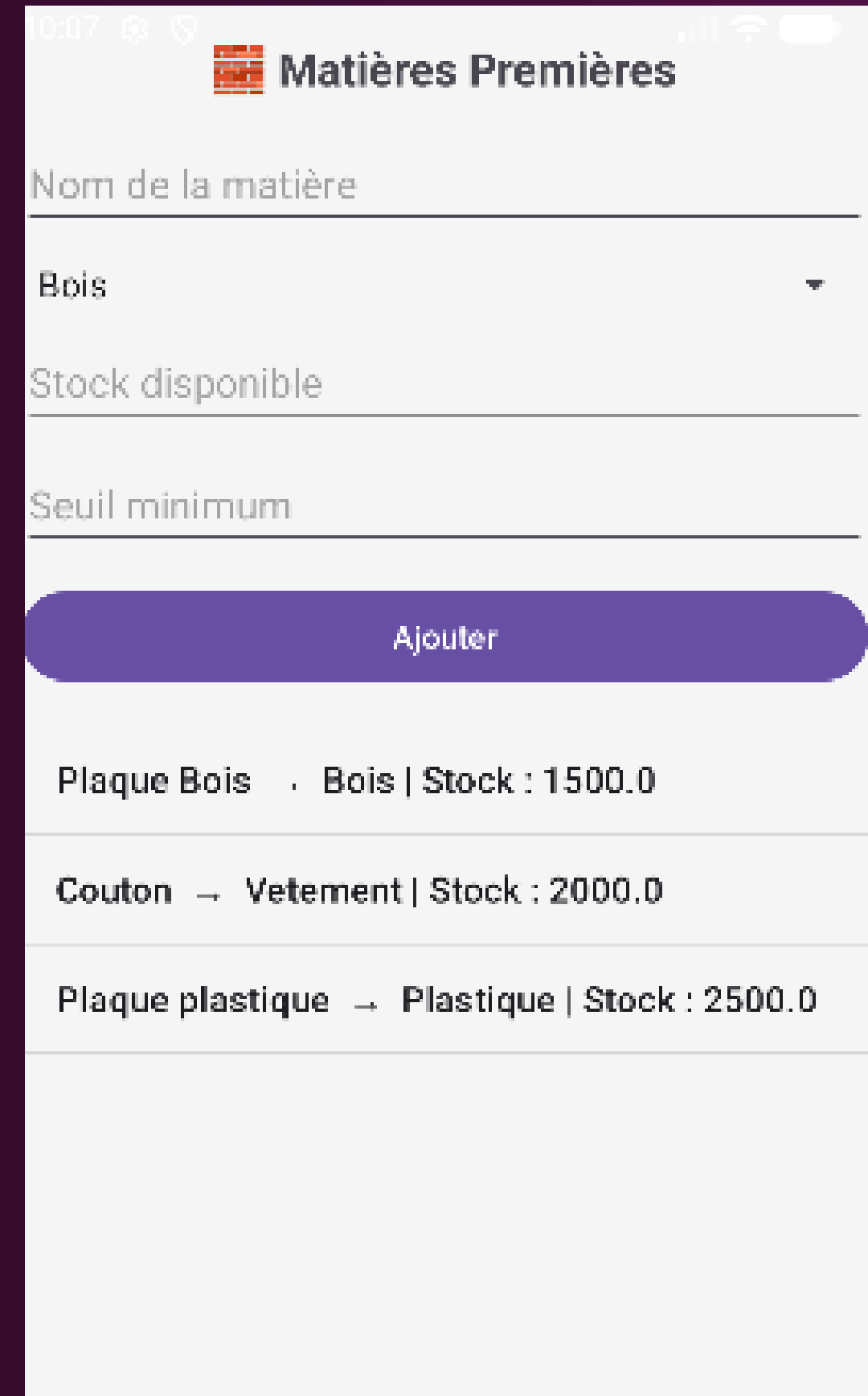
A mobile app interface for adding raw materials. It features a dropdown menu with options 'Bois', 'Plastique', and 'Vetement'. Below the menu is a text input field labeled 'Seuil minimum'. A blue 'Ajouter' button is positioned below the input field. At the bottom, a list item displays 'Plaque Bois' followed by a right arrow, 'Bois', and 'Stock : 1500.0'.

Bois
Plastique
Vetement

Seuil minimum

Ajouter

Plaque Bois → Bois | Stock : 1500.0



A mobile app interface for viewing raw materials. The title bar shows 'Matières Premières'. Below it are four input fields: 'Nom de la matière', 'Bois', 'Stock disponible', and 'Seuil minimum'. A blue 'Ajouter' button is located below these fields. At the bottom, a list displays three items: 'Plaque Bois' with a right arrow, 'Bois', and 'Stock : 1500.0'; 'Couton' with a right arrow, 'Vetement', and 'Stock : 2000.0'; and 'Plaque plastique' with a right arrow, 'Plastique', and 'Stock : 2500.0'.

Matières Premières

Nom de la matière

Bois

Stock disponible

Seuil minimum

Ajouter

Plaque Bois → Bois | Stock : 1500.0

Couton → Vetement | Stock : 2000.0

Plaque plastique → Plastique | Stock : 2500.0

```

10 class MatierePremiereActivity : AppCompatActivity() {
11
12     6 Usages
13     private lateinit var db: AppDatabase
14     4 Usages
15     private lateinit var spCategorie: Spinner
16     2 Usages
17     private lateinit var listView: ListView
18
19     6 Usages
20     private var categories = listOf<CategorieEntity>()
21
22     override fun onCreate(savedInstanceState: Bundle?) {
23         super.onCreate(savedInstanceState)
24         setContentView(R.layout.activity_matiere_premiere)
25
26         db = AppDatabase.getDatabase(context = this)
27
28         spCategorie = findViewById(id = R.id.spCategories)
29         listView = findViewById(id = R.id.listViewMatierePremiere)
30
31         val etNom = findViewById<EditText>(id = R.id.etNom)
32         val etStock = findViewById<EditText>(id = R.id.etStock)
33         val btnAdd = findViewById<Button>(id = R.id.btnAdd)
34         val etSeuil = findViewById<EditText>(id = R.id.etSeuil)
35
36         btnAdd.setOnClickListener {
37             val nom = etNom.text.toString()
38             val stock = etStock.text.toString().toDoubleOrNull() ?: 5.0
39             val seuil = etSeuil.text.toString().toDoubleOrNull() ?: 5.0
40             val categorie = spCategorie.selectedItem as CategorieEntity
41             val matierePremiereEntity = MatierePremiereEntity(
42                 nom = nom,
43                 categorieId = categorie.id,
44                 stock = stock,
45                 seuil = seuil
46             )
47             db.matierePremiereDao().insert(matierePremiereEntity)
48         }
49     }
50 }

```

```

36 if (nom.isNotEmpty() && categories.isNotEmpty()) {
37     val categorie = categories[spCategorie.selectedItemPosition]
38
39     Thread {
40         db.matierePremiereDao().insert(
41             mp = MatierePremiereEntity(
42                 nom = nom,
43                 categorieId = categorie.id,
44                 stock = 0.0,
45                 seuil = seuil
46             )
47         )
48
49         runOnUiThread {
50             etNom.text.clear()
51             etSeuil.text.clear()
52             loadMatierePremieres()
53         }
54     }.start()
55 }
56
57
58
59 loadCategories()
60 loadMatierePremieres()
61

```

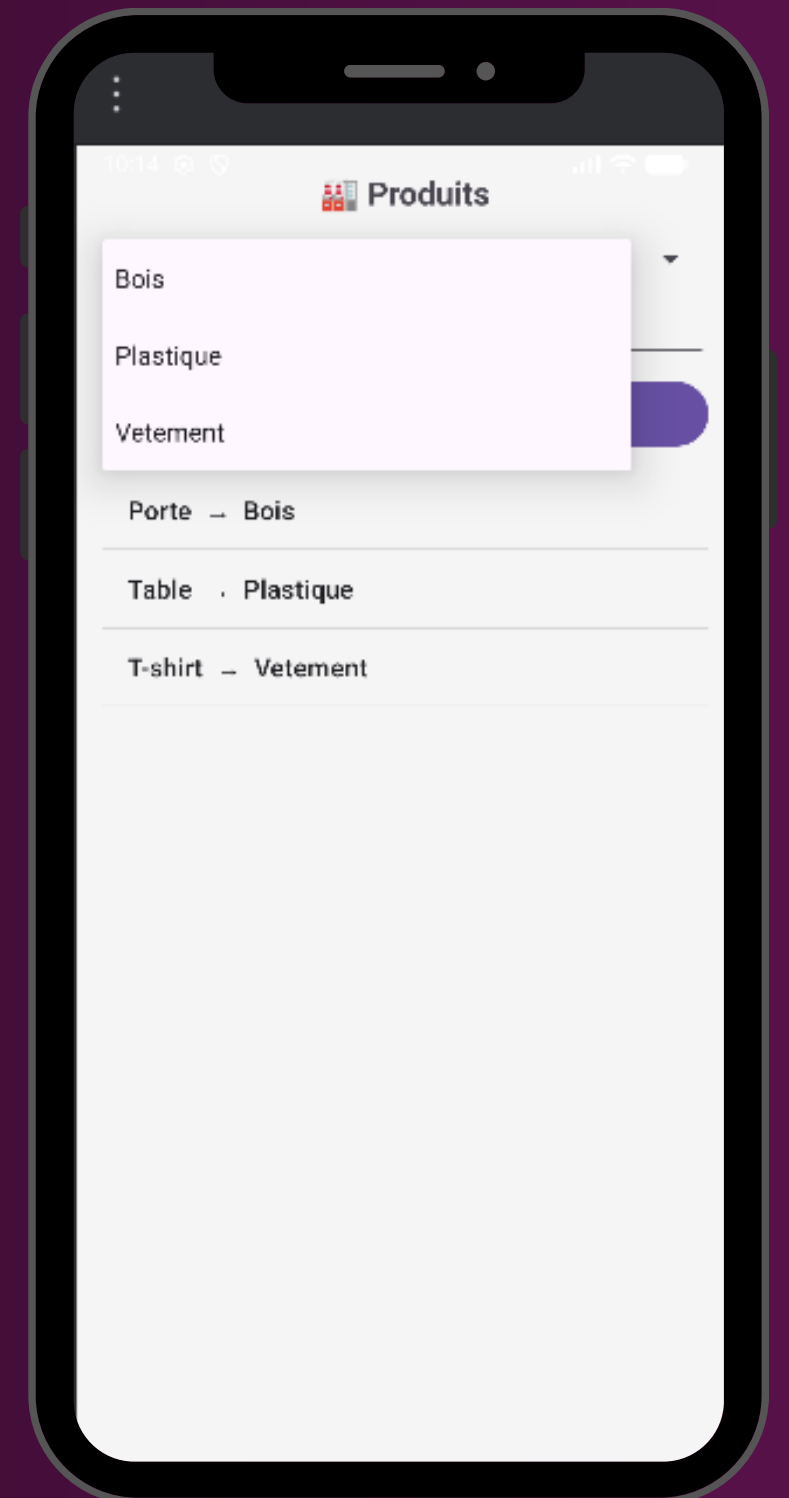
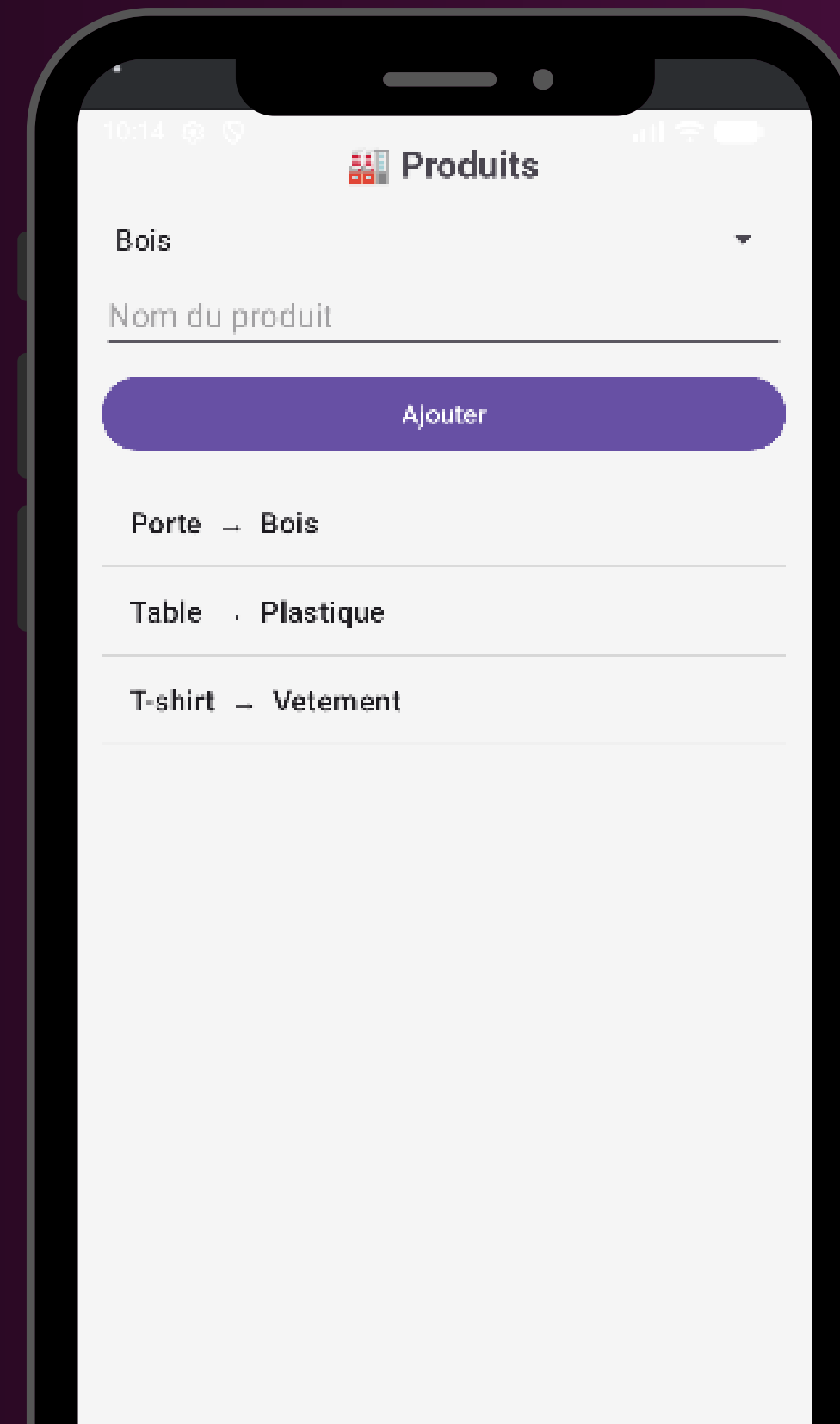
Partie de Matiere Premier Activity

Gestion des produits

- Chaque produit peut être :
 - ajouté à une catégorie
 - associé à plusieurs matières premières
 - utilisé dans les opérations de production

```
private fun loadProduits() {  
    Thread {  
        val produits = db.produitDao().getAll()  
        val cats = db.categorieDao().getAll()  
  
        val displayList = produits.map { produit ->  
            val cat = cats.find { it.id == produit.categorieId }  
            "${produit.nom} → ${cat?.nom ?: "?"}"  
        }  
  
        runOnUiThread {  
            listViewProduits.adapter = ArrayAdapter(  
                context = this,  
                resource = android.R.layout.simple_list_item_1,  
                objects = displayList  
            )  
        }  
    }.start()  
}
```

Partie de produit Activity



Composition produit – matière première

- Cette partie permet de définir
 - les matières nécessaires à la fabrication d'un produit
 - les quantités utilisées pour chaque matières

7:52 5G M Composition du Produit

Produit
Porte ▼

Matière Première
Plaque Bois ▼

Quantité utilisée

+ Ajouter à la composition

Production possible : 0

Plaque Bois : 3670.0

Composition du Produit

Produit

Porte ▼

Table

T-shirt

Plaque Bois ▼

Quantité utilisée

+ Ajouter à la composition

Composition du Produit

Produit
Porte ▼

Matière Première

Plaque Bois ▼

Couton

Plaque plastique

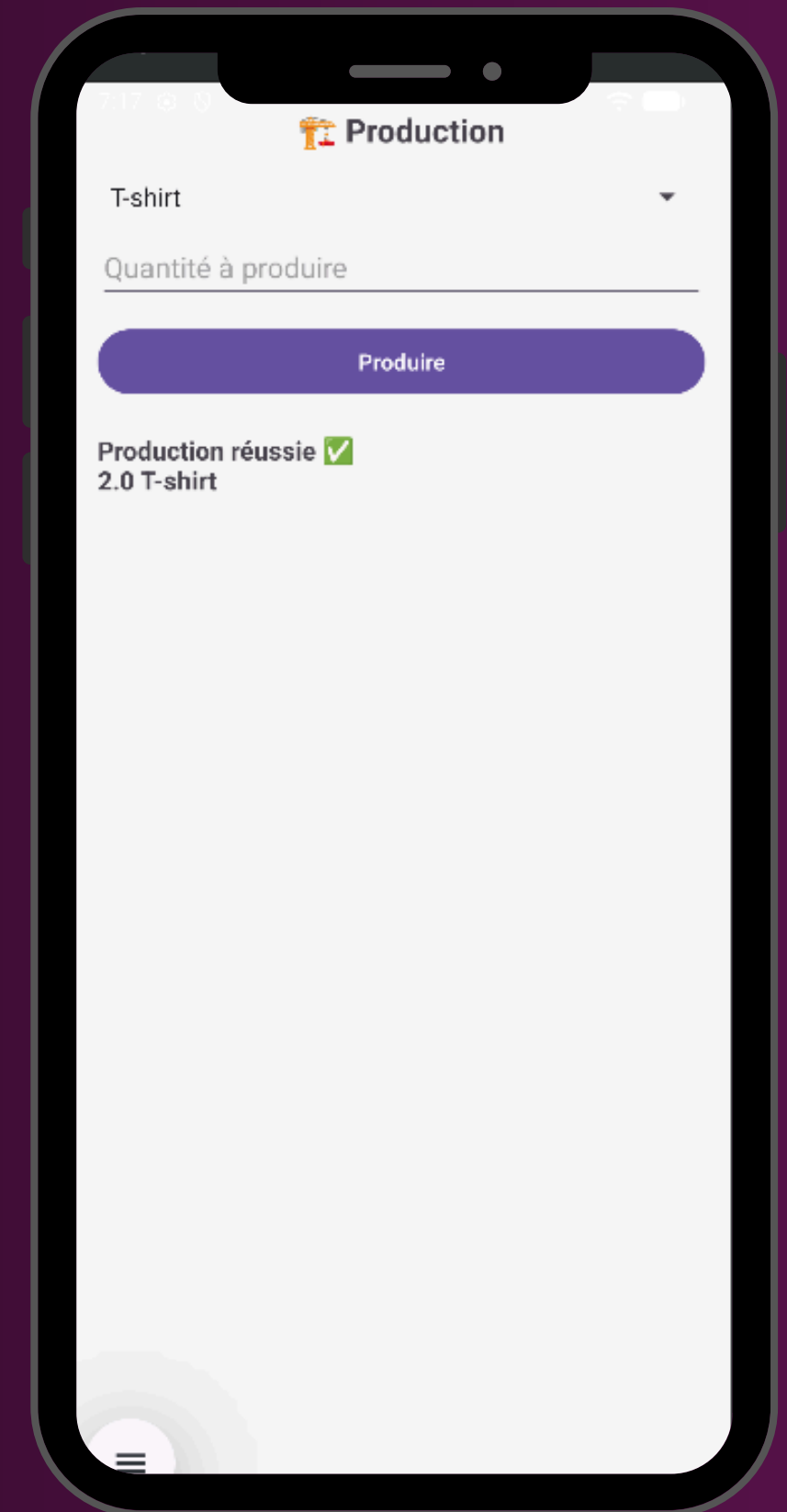
+ Ajouter à la composition

Gestion de la production

L'utilisateur peut :

- lancer une opération de production
- calculer la production possible selon le stock
- enregistrer l'historique des productions

```
18 private var produits = listOf<ProduitEntity>()
19
20 @
21 override fun onCreate(savedInstanceState: Bundle?) {
22     super.onCreate(savedInstanceState)
23     setContentView(R.layout.activity_production)
24
25     db = AppDatabase.getDatabase(context = this)
26
27     spProduits = findViewById(id = R.id.spProduits)
28     etQuantite = findViewById(id = R.id.etQuantite)
29     tvResult = findViewById(id = R.id.tvResult)
30     val btnProduire = findViewById<Button>(id = R.id.btnProduire)
31
32     loadProduits()
33
34     btnProduire.setOnClickListener {
35         produire()
36     }
37 }
```



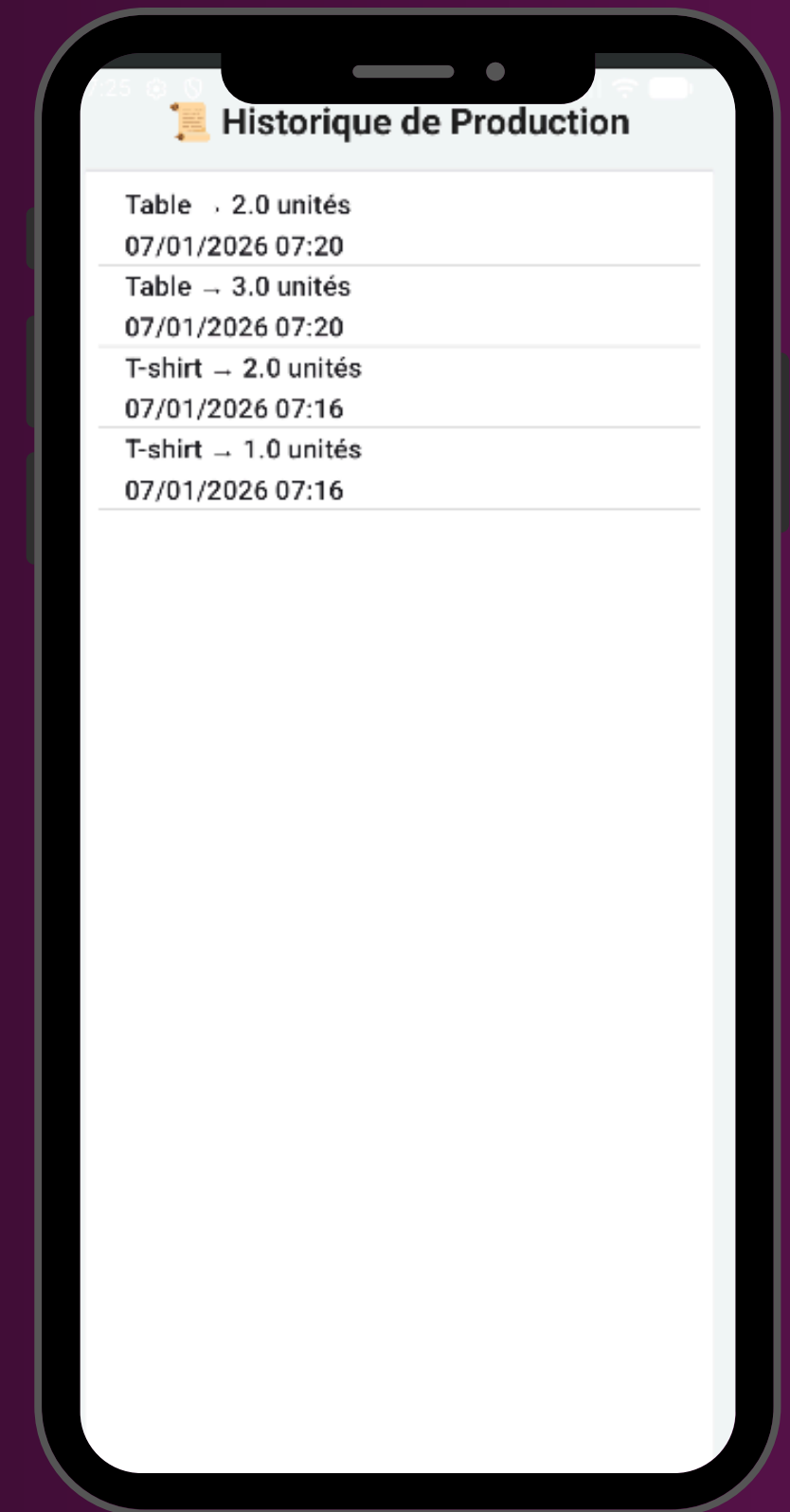
Historique de Production

Cette fonctionnalité offre à l'utilisateur la possibilité de :

- Consulter la liste des productions réalisées
- Visualiser les produits fabriqués et les quantités produites
- Suivre l'évolution de la production dans le temps
- Assurer une meilleure traçabilité des opérations

Chaque opération de production est automatiquement enregistrée dans la base de données locale à l'aide de Room Database, garantissant ainsi la fiabilité et la persistance des données même après la fermeture de l'application.

L'écran « Historique de Production » présente les informations sous forme de liste claire et lisible, facilitant l'analyse et le suivi des activités de production. Cette fonctionnalité constitue un élément essentiel pour le contrôle de la production et l'aide à la prise de décisio



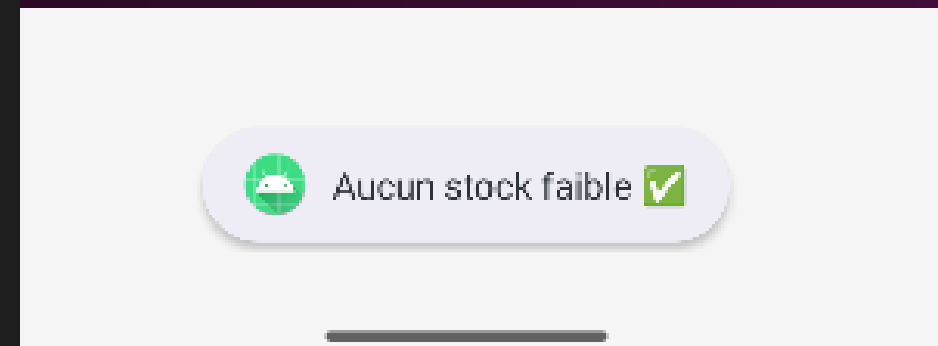
Stock faible

Une page dédiée affiche :

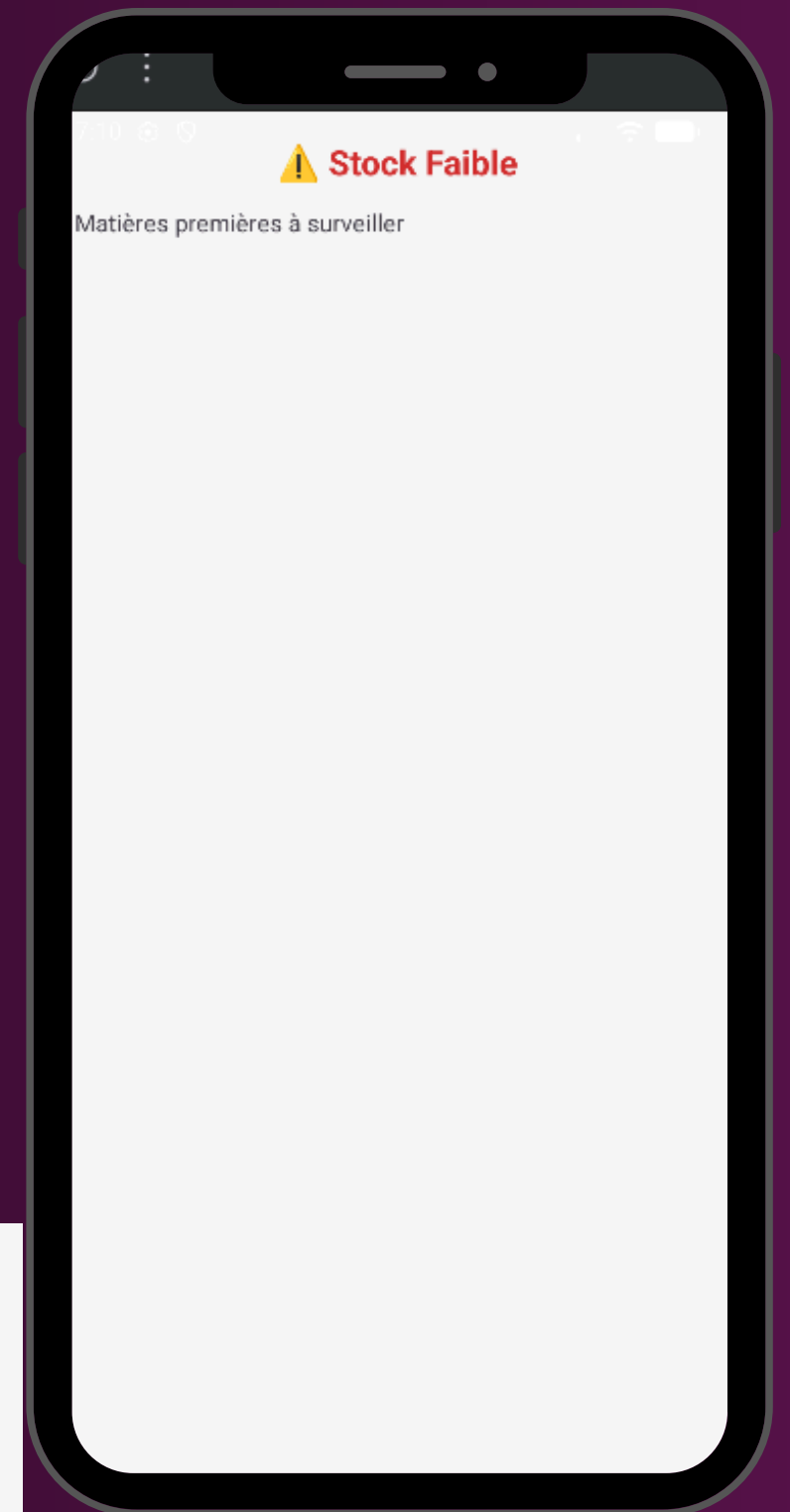
- les matières premières dont le stock est inférieur à un seuil
- des alertes visuelles pour attirer l'attention de l'utilisateur

```
2 Usages
></> class StockFaibleActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_stock_faible)
        val listView = findViewById<ListView>( id = R.id.listViewStockFaible)
        val db = AppDatabase.getDatabase( context = this)
        Thread {
            val matieres = db.matierePremiereDao().getStockFaible()
            runOnUiThread {
                if (matieres.isEmpty()) {
                    Toast.makeText(
                        context = this,
                        text = "Aucun stock faible ✅",
                        duration = Toast.LENGTH_LONG
                    ).show()
                } else {
                    val display = matieres.map {
                        "⚠️ ${it.nom} | Stock = ${it.stock}"
                    }
                    listView.adapter = ArrayAdapter(
                        context = this,
                        resource = android.R.layout.simple_list_item_1,
                        objects = display
                    )
                }
            }
        }.start()
    }
}
```



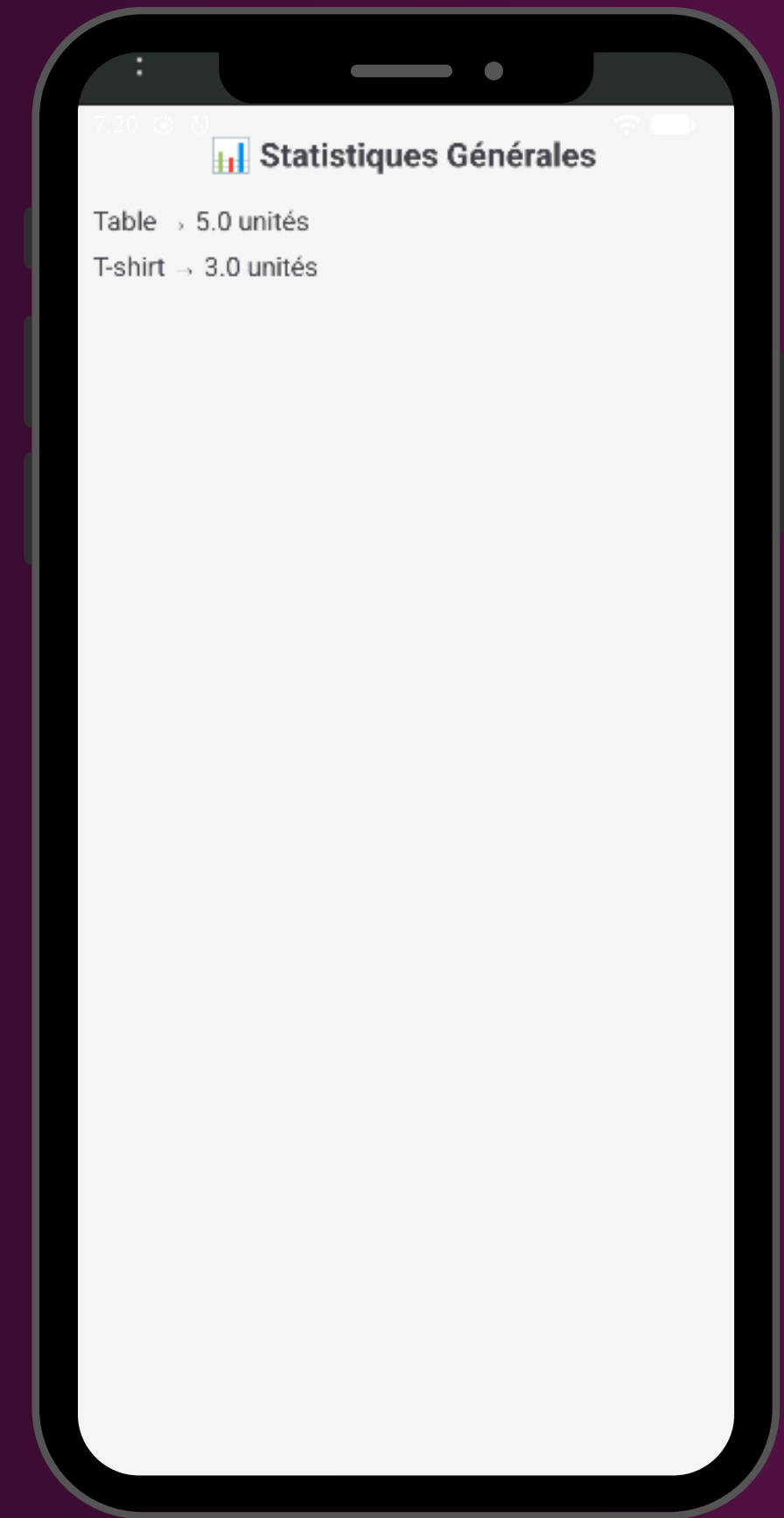
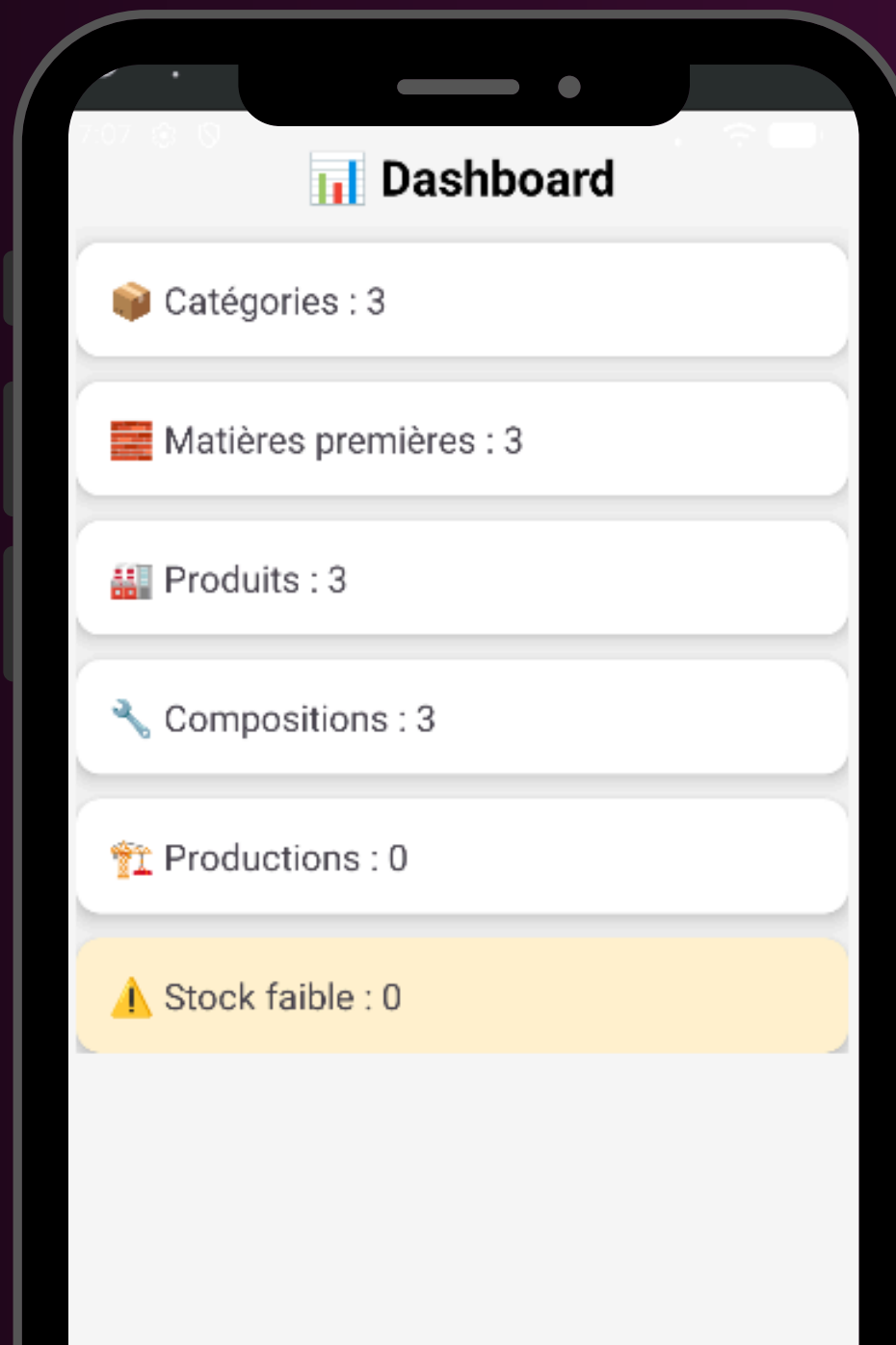
Alert



Statistiques et Dashboard

Statistiques et Dashboard:

- une vue globale sur les données
- le nombre de catégories, produits, matières et productions
- une meilleure prise de décision



Conclusion

Ce projet a permis de mettre en pratique les connaissances acquises en développement mobile Android, notamment l'utilisation de Room Database, la structuration du code et la conception d'interfaces professionnelles.

L'application développée constitue une base solide pouvant être améliorée par l'ajout de nouvelles fonctionnalités telles que :

- la synchronisation avec une base de données distante
- la gestion des rôles utilisateurs
- l'exportation des données

