Buy Or Not

Doc Technique

MARTIN Justine

BTS SIO 2 - Jean Rostand Caen

Table des matières

1 Présentation

La présente application Android a pour but la gestion des produits en utilisant une base de données semblable à celle d'Open Food Facts. En effet, si cette application est satisfaisante, on pourra alors la redévelopper en utilisant cette fois ci la base de données d'Open Food Facts tout en s'appuyant sur le code préexistant. Cette documentation a donc pour objectif de présenter le fonctionnement de l'application pour les utilisateurs afin de les accompagner dans la prise en main de celle ci.

Attention, toutes les fonctionnalités ne sont pas implémentées. L'objectif principal de l'application est respecté cependant des informations comme le pays de provenance, les catégories, etc sont volontairement omises, l'application n'étant actuellement conçue que dans un cadre d'ébauche pour un éventuel développement à plus grande échelle.

De manière générale, le présent document couvrira uniquement la partie du code de l'application. Celui-ci ne traitera pas des aspects propres à Android comme l'utilisation de Sqlite, les RecyclerView, la bibliothèque AppCompat, la bibliothèque Design, etc. Cependant voici quelques liens afin de vous aiguiller :

— RecyclerView : https://guides.codepath.com/android/Using-the-RecyclerViews
— AppCompat : https://www.captechconsulting.com/blogs/android-material-themes-made-easy-with-appcompat
— Design : https://android-developers.googleblog.com/2015/05/android-design-support-library.html

2 Gestion de la base de données

Tout le code concernant la base de données est stocké dans le package com.ppe.buyornot.bdd.

A La classe BuyOrNotDatabase

Cette classe hérite de la classe SQLiteOpenHelper. La base de données est sauvegardée dans le fichier "BurOrNot.db". La classe utilise le Pattern Singleton. Lors de la création de la base de données, on réeffectue toutes les modifications une à une. Ces modifications sont faites à l'aide de scripts SQL contrairement à ce que l'on a pu voir dans les applications précédentes.

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    onUpgrade(sqLiteDatabase, 1, DATABASE_VERSION);
}

@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
    if (oldVersion == 1) {
        execFile(sqLiteDatabase, "1_struct.sql");
        oldVersion++;
    }

    if (oldVersion == 2) {
        execFile(sqLiteDatabase, "2_data.sql");
        oldVersion++;
    }
}
```

FIGURE 2.1 – Code de mise à jour de la base de données

Ainsi à chaque nouvelle version de la base de données, on exécute un script SQL précis en fonction des versions en retard. Pour exécuter ce script, on appelle la fonction "execFile" dont voici le code :

```
private void execFile(SQLiteDatabase sqLiteDatabase, String path) {
    BufferedReader input = null;
    StringBuilder fileContent = new StringBuilder();
    try {
        input = new BufferedReader(new InputStreamReader(this.context.getAssets().open(path)));

        String line;
        while ((line = input.readLine()) != null) {
            fileContent.append(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    try {
        for (String query : fileContent.toString().split(";"))
            sqLiteDatabase.execSQL(query);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

FIGURE 2.2 – Code d'exécution d'un script

Le code n'est rien de plus qu'une simple lecture du fichier SQL fourni en paramètre. La seule subtilité

réside dans le fait que pour exécuter le script, on doit exécuter chaque instruction une à une, raison pour laquelle on sépare le contenu du fichier à chaque ";".

B Les scripts de base de données

Tous les sripts SQL de la base de données sont contenus dans le dossier suivant : "src/app/src/main/assets". Lorsque l'on veut faire évoluer la base de données, il suffit donc de faire un script SQL et d'exécuter le script correspondant. Il est préférable de respecter la nommenclature visible, à savoir le numéro de version suivi du descriptif du script.

C Les classes modèles

Avant d'expliquer comment nous manipulons la base de données afin d'y ajouter ou récupérer des informations, il faut savoir que nous avons dans le package com.ppe.buyornot.bdd.model toutes nos classes métiers. La particularité de celles-ci est qu'elles implémentent l'interface IEntity.

```
public interface IEntity {
    void createFromCursor(Cursor cursor, Context context);
}
```

FIGURE 2.3 – L'interface IEntity

Cette interface permet d'avoir une base commune pour toutes nos classes métiers. Cependant elle fournit aussi la méthode "createFromCursor" qui prend en paramètre des objets fournis par le SDK d'Android, à savoir un Cursor et un Context. Comme son nom l'indique elle permet de créer une entité depuis un Cursor fournis. Ainsi, appeller cette méthode sur la classe métier "Produit", par exemple, permettra d'obtenir une instance de celle-ci avec toutes ses informations suivant les valeurs du Cursor(ex : le libellé, l'URL, le taux de sucre, etc). Le Context est là pour permettre d'appeler un autre DAO (nous en parlerons dans la partie suivante).

```
@Override
public void createFromCursor(Cursor cursor, Context context) {
    this.id = cursor.getInt(cursor.getColumnIndex(ProduitDao.FIELD_ID));
    this.libelle = cursor.getString(cursor.getColumnIndex(ProduitDao.FIELD_LIBELLE));
    this.ingredient = cursor.getString(cursor.getColumnIndex(ProduitDao.FIELD_INGREDIENT));
    this.lien = cursor.getString(cursor.getColumnIndex(ProduitDao.FIELD_LIEN));
    this.quantite = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_QUANTITE));
    this.energie = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_ENERGIE));
    this.matiereGrasse = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_MATIERE_GRASSE));
    this.acideGras = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_ACIDE_GRAS));
    this.glucide = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_GLUCIDE));
    this.sucre = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_SUCRE));
    this.fibre = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_FIBRE));
    this.proteine = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_PROTEINE));
    this.sel = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_SEL));
    this.sodium = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_SODIUM));
    this.fruits = cursor.getFloat(cursor.getColumnIndex(ProduitDao.FIELD_FRUITS));

    this.nutriscore = new Nutriscore(cursor.getString(cursor.getColumnIndex(ProduitDao.FIELD_ID_NUTRISCORE)));

    CodeEmballeurDao codeEmballeurDao = new CodeEmballeurDao(context);
    this.codeEmballeur = codeEmballeurDao.get(cursor.getInt(cursor.getColumnIndex(ProduitDao.FIELD_ID_CODE_EMBALLEUR)));
    codeEmballeurDao.close();

    CategNutriscoreDao categNutriscoreDao = new CategNutriscoreDao(context);
    this.categNutriscore = categNutriscoreDao.get(cursor.getInt(cursor.getColumnIndex(ProduitDao.FIELD_ID_CATEG_NUTRISCORE)));
    categNutriscoreDao.close();

    NovaDao novaDao = new NovaDao(context);
    this.nova = novaDao.get(cursor.getInt(cursor.getColumnIndex(ProduitDao.FIELD_ID_NOVA)));
    novaDao.close();
}
```

FIGURE 2.4 – La méthode createFromCursor de la classe Produit

D Pattern DAO

Tous les DAO sont disponibles dans le package com.ppe.buyornot.bdd.dao

Passons maintenant aux explications concernant les DAO. Tous les DAO implémentent l'interface IEntityManager. Cette classe utilise la généricité sur toute classe héritant de l'interface IEntity. On à ainsi plusieurs méthodes qui permettent par exemple de récupérer une entité, d'en ajouter une, d'en supprimer une, etc, le tout uniquement pour la classe T.

```
public interface IEntityManager<T extends IEntity> {

    List<T> getAll();

    T get(int id);

    long add(T entity);

    int update(T entity);

    void delete(int id);

    public ContentValues fillContentValues(T entity);
}
```

FIGURE 2.5 – L'interface IEntityManager

Un avantage de cette méthode est l'utilisation possible du Pattern Strategy pour la création d'une factory (nous en parlerons plus tard).

Voyons maintenant comment est conçu un DAO :

```
public class ProduitDao implements IEntityManager<Produit> {
    public static final String FIELD_ID = "PRO_ID";
    public static final String FIELD_LIBELLE = "PRO_LIBELLE";
    public static final String FIELD_ID_CODE_EMBALLEUR = "COD_ID";
    public static final String FIELD_ID_NOVA = "NOVA_ID";
    public static final String FIELD_ID_NUTRISCORE = "NUT_CODE";
    public static final String FIELD_INGREDIENT = "PRO_INGREDIENT";
    public static final String FIELD_LIEN = "PRO_LIEN";
    public static final String FIELD_QUANTITE = "PRO_QUANTITE";
    public static final String FIELD_ENERGIE = "PRO_ENERGIE";
    public static final String FIELD_MATIERE_GRASSE = "PRO_MATIEREGRASSE";
    public static final String FIELD_ACIDE_GRAS = "PRO_ACIDEGRAS";
    public static final String FIELD_GLUCIDE = "PRO_GLUCIDE";
    public static final String FIELD_SUCRE = "PRO_SUCRE";
    public static final String FIELD_FIBRE = "PRO_FIBRE";
    public static final String FIELD_PROTEINE = "PRO_PROTEINE";
    public static final String FIELD_SEL = "PRO_SEL";
    public static final String FIELD_SODIUM = "PRO_SODIUM";
    public static final String FIELD_FRUITS = "PRO_FRUIT";
    public static final String FIELD_ID_CATEG_NUTRISCORE = "PROD_ID_CATEG_NUTRISCORE";
    private static final String TABLE_NAME = "PRODUIT";
```

FIGURE 2.6 – Base d'un DAO

Comme on peut le voir, un DAO utilise l'interface IEntityManager et définit des constantes publiques qui permettent de définir les différents champs SQL disponibles sur la table de l'entité.

```java
public ProduitDao(Context context) {
    buyOrNotDatabase = BuyOrNotDatabase.getInstance(context);
    this.open();

    this.context = context;
}

public void open() {
    db = buyOrNotDatabase.getWritableDatabase();
}

public void close() {
    db.close();
}
```

FIGURE 2.7 – Constructeur et gestion des flux à la base de données

Ici, on utilise le Context fourni en paramètre afin de récupérer la connexion à la base de données. On définit également 2 méthodes qui permettent de gérer correctement la mémoire en ouvrant ou fermant la connexion active à la base de données.

```java
public List<Produit> getAll() {
    return getAll(FIELD_ID);
}

public List<Produit> getAll(String orderbyClause) {
    List<Produit> produits = new ArrayList<>();

    Cursor c = db.rawQuery("SELECT * FROM " + TABLE_NAME + " ORDER BY "+orderbyClause, null);
    while (c.moveToNext()) {
        Produit p = new Produit();
        p.createFromCursor(c, this.context);

        produits.add(p);
    }

    return produits;
}

public Produit get(int id) {
    Produit p = new Produit();

    Cursor c = db.rawQuery("SELECT * FROM " + TABLE_NAME + " WHERE " + FIELD_ID + "=" + id, null);
    if (c.moveToFirst()) {
        p.createFromCursor(c, this.context);
        c.close();
    }

    return p;
}
```

FIGURE 2.8 – Implémentation des méthodes get(int) et getAll() de IEntityManager

Le DAO pour les produits est un peu particulier. En effet, la méthode getAll() appelle la méthode getAll(String) afin de trier les éléments dans un certains ordre. L'ordre est défini par la classe String fournie en paramètre. Ainsi, un appel à getAll() retourne les produits triés par ordre d'ajout. Concernant les méthodes get, getAll, on utilise la méthode de l'interface IEntity, à savoir createFromCursor. Ainsi, ici, la création du produit et le changement de ses différents attributs tiennent en 2 lignes seulement et permettent une factorisation du code qui est commun.

```java
public long add(Produit produit) {
    ContentValues values = fillContentValues(produit);

    return db.insert(TABLE_NAME, null, values);
}

public int update(Produit produit) {
    ContentValues values = fillContentValues(produit);

    String where = FIELD_ID + " = ?";
    String[] whereArgs = {"" + produit.getId()};

    return db.update(TABLE_NAME, values, where, whereArgs);
}

public void delete(int id) {
    String where = FIELD_ID + " = ?";
    String[] whereArgs = {"" + id};

    db.delete(TABLE_NAME, where, whereArgs);
}
```

FIGURE 2.9 – Implémentation des méthodes add(Produit), update(Produit) et delete(int) de IEntityManager

Le code de la méthode add et update utilisent donc la méthodes fillContentValues qui permet d'obtenir une instance de ContentValues contenant toutes les informations sur le produit fourni en paramètre. On a ainsi, une fois de plus, une factorisation du code qui simplifie l'ajout et la modification d'un produit. La suppression, elle se contente uniquement de l'id du produit à supprimer.

```java
public ContentValues fillContentValues(Produit produit) {
    ContentValues contentValues = new ContentValues();

    contentValues.put(FIELD_LIBELLE, produit.getLibelle());
    if (produit.getCodeEmballeur() != null)
        contentValues.put(FIELD_ID_CODE_EMBALLEUR, produit.getCodeEmballeur().getId());
    else
        contentValues.putNull(FIELD_ID_CODE_EMBALLEUR);

    if (produit.getNova() != null)
        contentValues.put(FIELD_ID_NOVA, produit.getNova().getId());
    else
        contentValues.putNull(FIELD_ID_NOVA);

    if (produit.getNutriscore() != null)
        contentValues.put(FIELD_ID_NUTRISCORE, produit.getNutriscore().getCode());
    else
        contentValues.putNull(FIELD_ID_NUTRISCORE);

    if (produit.getCategNutriscore() != null)
        contentValues.put(FIELD_ID_CATEG_NUTRISCORE, produit.getCategNutriscore().getId());
    else
        contentValues.putNull(FIELD_ID_CATEG_NUTRISCORE);

    contentValues.put(FIELD_INGREDIENT, produit.getIngredient());
    contentValues.put(FIELD_LIEN, produit.getLien());
    contentValues.put(FIELD_QUANTITE, produit.getQuantite());
    contentValues.put(FIELD_ENERGIE, produit.getEnergie());
    contentValues.put(FIELD_MATIERE_GRASSE, produit.getMatiereGrasse());
    contentValues.put(FIELD_ACIDE_GRAS, produit.getAcideGras());
    contentValues.put(FIELD_GLUCIDE, produit.getGlucide());
    contentValues.put(FIELD_SUCRE, produit.getSucre());
    contentValues.put(FIELD_FIBRE, produit.getFibre());
    contentValues.put(FIELD_PROTEINE, produit.getProteine());
    contentValues.put(FIELD_SEL, produit.getSel());
    contentValues.put(FIELD_SODIUM, produit.getSodium());
    contentValues.put(FIELD_FRUITS, produit.getFruits());

    return contentValues;
}
```

FIGURE 2.10 – Implémentation de la méthode fillContentValues

Rien de spécial ici, on instancie un ContentValues que l'on remplit en utilisant le produit en paramètre puis on le retourne.

3 Les différentes "Activity"

A MainActivity

La classe MainActivity est la 1re interface que verra l'utilisateur. Celle ci liste les produits par l'intermédiaire d'un RecyclerView en affichant leur nom ainsi que leur nutriscore. Voyons d'abord l'adapter utilisé dans le projet.

i Adapter

L'adapteur est situé dans le package com.ppe.buyornot.adapter. Pour les produits il s'agit du ProduiRecyclerAdapter. Les adapters pour les RecyclerView faisant partie des bibliothèques support d'Android, leur fonctionnement ne sera pas traité ici. Nous allons uniquement parler de quelques spécificités de la classe.

```
public class ProduitRecyclerAdapter extends RecyclerView.Adapter<ProduitRecyclerAdapter.ViewHolder> {

    private Context context;
    private List<Produit> produits;
    private View.OnClickListener rowClickLister;

    public ProduitRecyclerAdapter(List<Produit> produits, View.OnClickListener rowClickListener, Context context) {
        this.produits = produits;
        this.context = context;
        this.rowClickLister = rowClickListener;
    }
```

FIGURE 3.1 – Constructeur de l'adapter

En effet, pour construire l'adapter on doit fournir la liste des produits à afficher, un OnClickListener ainsi qu'un context. Chacun aura une utilité par la suite.

La liste des produits Elle permet de lister certains produits. On aurait pu directement faire appel au DAO ici pour récupérer tous les produits, cependant, cela permet de conserver le même code pour lister les produits peut importe que l'on souhaite tout lister ou seulement certains (comme ceux qui ont uniquement un nutriscore en A par exemple).

L'OnClickListener Il s'agit du callback pour la sélection d'un item sur la liste. Le fait de le fournir en paramètre ce callback permet de gérer de différentes manières un clic sur un élément de la liste.

Le Context Il permet d'obtenir un accès aux Drawables de l'application afin de récupérer celui correspondant au nutriscore du produit que l'on bind.

ii Système de tri

Le système de tri est rendu possible par la méthode getAll(String) de ProduitDao. Dans le menu "inflaté" on peut alors sélectionner par l'intermédiaire des items la méthode de tri sélectionné.

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_add:
            Intent intent = new Intent(MainActivity.this, AddProduitActivity.class);
            startActivityForResult(intent, ACTIVITY_CODE_ADD_PRODUIT);
            return true;

        case R.id.action_sortRecent:
            if (item.isChecked())
                item.setChecked(false);
            else {
                item.setChecked(true);
                updateRecyclerView();
            }

            return true;


        case R.id.action_sortLibelle:
            if (item.isChecked())
                item.setChecked(false);
            else {
                item.setChecked(true);
                updateRecyclerView(ProduitDao.FIELD_LIBELLE);
            }

            return true;


        case R.id.action_sortNovaNutriscore:
            if (item.isChecked())
                item.setChecked(false);
            else {
                item.setChecked(true);
                updateRecyclerView(ProduitDao.FIELD_ID_NOVA + ", " + ProduitDao.FIELD_ID_NUTRISCORE);
            }

            return true;


        case R.id.action_sortNova:
            if (item.isChecked())
                item.setChecked(false);
            else {
                item.setChecked(true);
                updateRecyclerView(ProduitDao.FIELD_ID_NOVA);
            }

            return true;
```

FIGURE 3.2 – La gestion des différents tri

On doit cependant changer manuellement la méthode sélectionnée. Dans le cas où la case ne serait pas cochée, on met alors le RecyclerView à jour en fonction des champs de la base de données fournis par le DAO. Ce tri est fait sous forme de requête SQL.

```java
private void updateRecyclerView(String orderByClause) {
    ProduitDao produitDao = new ProduitDao(this);
    this.produits = produitDao.getAll(orderByClause);

    ProduitRecyclerAdapter adapter = new ProduitRecyclerAdapter(this.produits, new RowClickListener(), this);
    this.recyclerView.setAdapter(adapter);
    this.recyclerView.setLayoutManager(new LinearLayoutManager(this));

    produitDao.close();
}
```

FIGURE 3.3 – Mise à jour de la RecyclerView

Rien de spécial dans ce code ci. On utilise la classe ProduitDao afin de récupérer tous les produits triés dans l'ordre demandé. On instancie ensuite l'adapter pour le RecyclerView que l'on utilise directement.

iii Interaction avec les autres Activity

Les activity pour mettre à jour un produit ou pour en ajouter un, retourne un code de résultat. De ce fait il est important de les démarrer en utilisant startActivityForResult(). Ce code de retour permet de savoir si la manipulation du produit a été validée. Si tel est le cas, on doit alors mettre à jour la RecyclerView.

```java
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    if (requestCode == ACTIVITY_CODE_UDPATE_PRODUIT) {
        if (resultCode == RESULT_OK) {
            this.updateRecyclerView();
        }
    } else if (requestCode == ACTIVITY_CODE_ADD_PRODUIT) {
        if (resultCode == RESULT_OK) {
            this.updateRecyclerView();
        }
    }
}
```

FIGURE 3.4 – Code du OnActivityResult

B AbstractProduitActivity

Cette classe fournit une base pour le layout "activity_produit". Elle permet de récupérer toutes les vues grace à la méthode "findViews()".

```java
private void findViews() {
    this.buttonDelete = findViewById(R.id.delete);
    this.editTextLibelle = findViewById(R.id.libelle);
    this.editTextQuantite = findViewById(R.id.quantite);
    this.editTextIngredients = findViewById(R.id.ingredients);
    this.editTextLien = findViewById(R.id.lien);
    this.editTextEnergie = findViewById(R.id.energie);
    this.editTextFibre = findViewById(R.id.fibre);
    this.editTextMatieresGrasses = findViewById(R.id.matieresGrasses);
    this.editTextAcidesGras = findViewById(R.id.acidesGras);
    this.editTextGlucides = findViewById(R.id.glucides);
    this.editTextSucres = findViewById(R.id.sucres);
    this.editTextSel = findViewById(R.id.sel);
    this.editTextSodium = findViewById(R.id.sodium);
    this.editTextProteine = findViewById(R.id.proteine);
    this.editTextFruit = findViewById(R.id.fruits);
    this.spinnerCodeEmballeur = findViewById(R.id.codeEmballeur);
    this.spinnerNova = findViewById(R.id.nova);
    this.spinnerCategNutriscore = findViewById(R.id.categNutriscore);
}
```

FIGURE 3.5 – Code de la méthode findViews()

On a aussi l'initialisation de tous les spinner de fait. Chaque spinner possède sa propre méthode pour l'initialiser.

```
private void initNovaSpinner() {
    NovaDao novaDao = new NovaDao(this);
    this.novas = novaDao.getAll();

    List<String> novasStr = new ArrayList<>();
    for(Nova n : this.novas)
        novasStr.add(n.getLibelle());

    ArrayAdapter<String> novaAdapter = new ArrayAdapter<>(this, android.R.layout.simple_spinner_dropdown_item, novasStr);
    novaDao.close();

    spinnerNova.setAdapter(novaAdapter);
}

private void initCodeEmballeurSpinner() {
    CodeEmballeurDao codeEmballeurDao = new CodeEmballeurDao(this);
    this.codeEmballeurs = codeEmballeurDao.getAll();

    List<String> codeEmballeurStr = new ArrayList<>();
    codeEmballeurStr.add(this.getString(R.string.codeEmballeurNull));
    for(CodeEmballeur ce : this.codeEmballeurs)
        codeEmballeurStr.add(ce.getLibelle());

    ArrayAdapter<String> codeEmballeurAdapter = new ArrayAdapter<>(this, android.R.layout.simple_spinner_dropdown_item, codeEmballeurStr);
    codeEmballeurDao.close();

    spinnerCodeEmballeur.setAdapter(codeEmballeurAdapter);
}

private void initCategNutriscore() {
    CategNutriscoreDao categNutriscoreDao = new CategNutriscoreDao(this);
    this.categNutriscore = categNutriscoreDao.getAll();

    List<String> categNutriscoreStr = new ArrayList<>();
    for(CategNutriscore cn : this.categNutriscore)
        categNutriscoreStr.add(cn.getLibelle());

    ArrayAdapter<String> categNutriscoreAdapter = new ArrayAdapter<>(this, android.R.layout.simple_spinner_dropdown_item, categNutriscoreStr);
    categNutriscoreDao.close();

    spinnerCategNutriscore.setAdapter(categNutriscoreAdapter);
}
```

FIGURE 3.6 – Code d'initialisation des spinner

De ce fait, les champs étant commun à l'ajout et la modification d'un produit, la classe fournit la méthode getProduit() qui permet de récupérer le produit en fonction de ce ui est actuellement saisie dans les champs. La fonction retournee "null" si un des champs n'est pas remplis, signifiant ainsi une erreur. En effet, il s'agit d'une sécurité visant à vérifier que l'on possède bien les informations du produit. Si l'on ne peut pas renseigner toutes les informations, c'est surement que l'on n'a pas le produit sous les yeux.

```
protected Produit getProduit() {
    if(this.editTextLibelle.getText().toString().isEmpty() ||
        this.editTextQuantite.getText().toString().isEmpty() ||
        this.editTextIngredients.getText().toString().isEmpty() ||
        this.editTextLien.getText().toString().isEmpty() ||
        this.editTextEnergie.getText().toString().isEmpty() ||
        this.editTextFibre.getText().toString().isEmpty() ||
        this.editTextMatieresGrasses.getText().toString().isEmpty() ||
        this.editTextAcidesGras.getText().toString().isEmpty() ||
        this.editTextGlucides.getText().toString().isEmpty() ||
        this.editTextSucres.getText().toString().isEmpty() ||
        this.editTextSel.getText().toString().isEmpty() ||
        this.editTextSodium.getText().toString().isEmpty() ||
        this.editTextProteine.getText().toString().isEmpty() ||
        this.editTextFruit.getText().toString().isEmpty()) {

        return null;
    }

    Produit produit = new Produit();

    produit.setLibelle(editTextLibelle.getText().toString());
    produit.setQuantite(Float.parseFloat(editTextQuantite.getText().toString()));
    produit.setIngredient(editTextIngredients.getText().toString());
    produit.setLien(editTextLien.getText().toString());
    produit.setEnergie(Float.parseFloat(editTextEnergie.getText().toString()));
    produit.setFibre(Float.parseFloat(editTextFibre.getText().toString()));
    produit.setMatiereGrasse(Float.parseFloat(editTextMatieresGrasses.getText().toString()));
    produit.setAcideGras(Float.parseFloat(editTextAcidesGras.getText().toString()));
    produit.setGlucide(Float.parseFloat(editTextGlucides.getText().toString()));
    produit.setSucre(Float.parseFloat(editTextSucres.getText().toString()));
    produit.setSel(Float.parseFloat(editTextSel.getText().toString()));
    produit.setSodium(Float.parseFloat(editTextSodium.getText().toString()));
    produit.setProteine(Float.parseFloat(editTextProteine.getText().toString()));
    produit.setFruits(Float.parseFloat(editTextFruit.getText().toString()));

    if(spinnerCodeEmballeur.getSelectedItemPosition() == 0)
        produit.setCodeEmballeur(null);
    else {
        produit.setCodeEmballeur(codeEmballeurs.get(spinnerCodeEmballeur.getSelectedItemPosition()-1));
    }

    produit.setCategNutriscore(categNutriscore.get(spinnerCategNutriscore.getSelectedItemPosition()));

    produit.updateNutriscore();

    produit.setNova(novas.get(spinnerNova.getSelectedItemPosition()));

    return produit;
}
```

FIGURE 3.7 – Création d'un produit depuis les champs

En plus de l'interface principale, le menu en haut à droite est le même sur les 2 écrans. De ce fait ce code fournit une méthode abstraite saveProduit() qui permet de définir le comportement à adopter lors d'un clic sur ce bouton.

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.valid, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_valid:
            this.saveProduit();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

protected abstract void saveProduit();
```

FIGURE 3.8 – Code d'initialisation des spinner

C AddProduitActivity

Cette classe hérite donc de AbstractProduitActivity. La seule spécificité du code est le fait de masquer le bouton pour supprimer. De plus, on implémente la fonction saveProduit() qui se charge uniquement d'afficher le message d'erreur si un des champs est vide ou alors, d'ajouter le produit si tous les champs sont remplis.

```java
public class AddProduitActivity extends AbstractProduitActivity {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        this.buttonDelete.setVisibility(View.GONE);
    }

    @Override
    protected void saveProduit() {
        Produit produit = this.getProduit();

        if(produit == null) {
            Toast.makeText(this, "Tous les champs doivent être remplis!", Toast.LENGTH_LONG).show();
            return;
        }

        ProduitDao produitDao = new ProduitDao(this);
        produitDao.add(produit);
        produitDao.close();

        Intent returnIntent = new Intent();
        setResult(AppCompatActivity.RESULT_OK, returnIntent);

        this.finish();
    }
```

FIGURE 3.9 – Code de la classe AddProduitActivity

D UpdateProduitActivity

Cette classe hérite donc aussi de AbstractProduitActivity. L'id du produit passé en paramètre est récupéré afin de sélectionner le produit correspondant dans la base de données. On exécute alors la

méthode updateUi(Produit).

```
private void updateUi(Produit produit) {
    editTextLibelle.setText(produit.getLibelle());
    this.editTextQuantite.setText(""+produit.getQuantite());
    this.editTextIngredients.setText((produit.getIngredient() != null) ? ""+produit.getIngredient() : "");
    this.editTextLien.setText((produit.getLien() != null) ? ""+produit.getLien() : "");
    this.editTextEnergie.setText(""+produit.getEnergie());
    this.editTextFibre.setText(""+produit.getFibre());
    this.editTextMatieresGrasses.setText(""+produit.getMatiereGrasse());
    this.editTextAcidesGras.setText(""+produit.getAcideGras());
    this.editTextGlucides.setText(""+produit.getGlucide());
    this.editTextSucres.setText(""+produit.getSucre());
    this.editTextSel.setText(""+produit.getSel());
    this.editTextSodium.setText(""+produit.getSodium());
    this.editTextProteine.setText(""+produit.getProteine());
    this.editTextFruit.setText(""+produit.getFruits());

    if(produit.getCodeEmballeur() != null) {
        for (int i = 0; i < this.codeEmballeurs.size(); i++) {
            if (this.codeEmballeurs.get(i).getId() == produit.getCodeEmballeur().getId()) {
                this.spinnerCodeEmballeur.setSelection(i+1);

                break;
            }
        }
    }

    for(int i = 0 ; i < this.novas.size() ; i++) {
        if(this.novas.get(i).getId() == produit.getNova().getId()) {
            this.spinnerNova.setSelection(i);

            break;
        }
    }

    for(int i = 0 ; i < this.categNutriscore.size() ; i++) {
        if(this.categNutriscore.get(i).getId() == produit.getCategNutriscore().getId()) {
            this.spinnerCategNutriscore.setSelection(i);

            break;
        }
    }

    this.buttonDelete.setOnClickListener(new DeleteProduitOnClickListener());
}
```

FIGURE 3.10 – Code de la méthode updateUi(Produit)

De même, le code de saveProduit(Produit) se chargera d'afficher également le message d'erreur ou de mettre à jour le produit correspondant dans la base de données.

```
@Override
protected void saveProduit() {
    ProduitDao produitDao = new ProduitDao(this);

    Produit produit = this.getProduit();
    if(produit == null) {
        Toast.makeText(this, "Tous les champs doivent être remplis!", Toast.LENGTH_LONG).show();
        return;
    }

    produit.setId(this.produitId);

    produitDao.update(produit);
    produitDao.close();

    Intent returnIntent = new Intent();
    setResult(AppCompatActivity.RESULT_OK, returnIntent);

    this.finish();
}
```

FIGURE 3.11 – Code de la méthode updateUi(Produit)

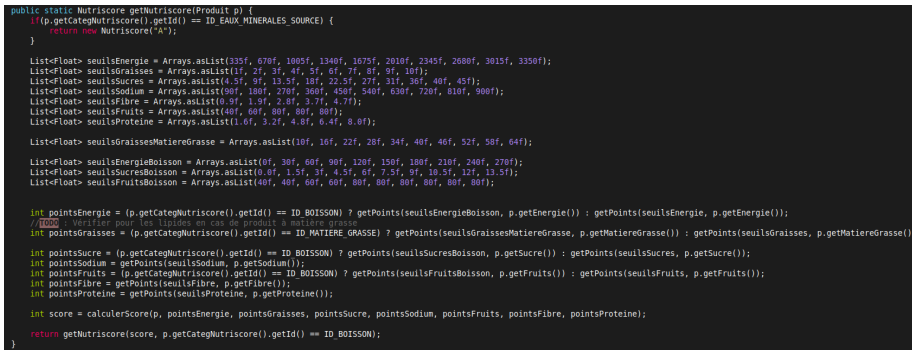
Enfin, un OnClickListener existe afin de supprimer le produit en cours de consultation lorsque l'on clique sur le bouton en bas de l'écran.

```
private class DeleteProduitOnClickListener implements View.OnClickListener {

    @Override
    public void onClick(View v) {
        ProduitDao produitDao = new ProduitDao(UpdateProduitActivity.this);
        produitDao.delete(produitId);
        produitDao.close();

        Intent returnIntent = new Intent();
        setResult(AppCompatActivity.RESULT_OK, returnIntent);

        UpdateProduitActivity.this.finish();
    }
}
```

FIGURE 3.12 – Code de la méthode updateUi(Produit)

4 Le calcul du Nutriscore

Le calcul du nutriscore est fait par la classe NutriscoreCalculator, qui est dans le package com.ppe.buyornot.util. Elle fournit une méthode static getNutriscore(Produit). Cette méthode est l'application du Document 4 du cahier des charges.



```
public static Nutriscore getNutriscore(Produit p) {
    if(p.getCategNutriscore().getId() == ID_EAUX_MINERALES_SOURCE) {
        return new Nutriscore("A");
    }

    List<Float> seuilsEnergie = Arrays.asList(335f, 670f, 1005f, 1340f, 1675f, 2010f, 2345f, 2680f, 3015f, 3350f);
    List<Float> seuilsGraisses = Arrays.asList(1f, 2f, 3f, 4f, 5f, 6f, 7f, 8f, 9f, 10f);
    List<Float> seuilsSucres = Arrays.asList(4.5f, 9f, 13.5f, 18f, 22.5f, 27f, 31f, 36f, 40f, 45f);
    List<Float> seuilsSodium = Arrays.asList(90f, 180f, 270f, 360f, 450f, 540f, 630f, 720f, 810f, 900f);
    List<Float> seuilsFibre = Arrays.asList(0.9f, 1.9f, 2.8f, 3.7f, 4.7f);
    List<Float> seuilsFruits = Arrays.asList(40f, 60f, 80f, 80f, 80f);
    List<Float> seuilsProteine = Arrays.asList(1.6f, 3.2f, 4.8f, 6.4f, 8.0f);

    List<Float> seuilsGraissesMatiereGrasse = Arrays.asList(10f, 16f, 22f, 28f, 34f, 40f, 46f, 52f, 58f, 64f);

    List<Float> seuilsEnergieBoisson = Arrays.asList(0f, 30f, 60f, 90f, 120f, 150f, 180f, 210f, 240f, 270f);
    List<Float> seuilsSucresBoisson = Arrays.asList(0.0f, 1.5f, 3f, 4.5f, 6f, 7.5f, 9f, 10.5f, 12f, 13.5f);
    List<Float> seuilsFruitsBoisson = Arrays.asList(40f, 40f, 60f, 60f, 80f, 80f, 80f, 80f, 80f, 80f);


    int pointsEnergie = (p.getCategNutriscore().getId() == ID_BOISSON) ? getPoints(seuilsEnergieBoisson, p.getEnergie()) : getPoints(seuilsEnergie, p.getEnergie());
    //TODO : Vérifier pour les lipides en cas de produit à matière grasse
    int pointsGraisses = (p.getCategNutriscore().getId() == ID_MATIERE_GRASSE) ? getPoints(seuilsGraissesMatiereGrasse, p.getMatiereGrasse()) : getPoints(seuilsGraisses, p.getMatiereGrasse()

    int pointsSucre = (p.getCategNutriscore().getId() == ID_BOISSON) ? getPoints(seuilsSucresBoisson, p.getSucre()) : getPoints(seuilsSucres, p.getSucre());
    int pointsSodium = getPoints(seuilsSodium, p.getSodium());
    int pointsFruits = (p.getCategNutriscore().getId() == ID_BOISSON) ? getPoints(seuilsFruitsBoisson, p.getFruits()) : getPoints(seuilsFruits, p.getFruits());
    int pointsFibre = getPoints(seuilsFibre, p.getFibre());
    int pointsProteine = getPoints(seuilsProteine, p.getProteine());

    int score = calculerScore(p, pointsEnergie, pointsGraisses, pointsSucre, pointsSodium, pointsFruits, pointsFibre, pointsProteine);

    return getNutriscore(score, p.getCategNutriscore().getId() == ID_BOISSON);
}
```

FIGURE 4.1 – Code de la méthode getNutriscore(Produit)

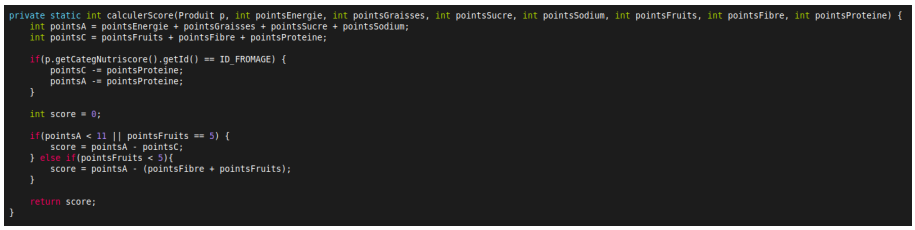
On initialise la liste des seuils pour le calcul des scores. On récupère alors le score pour un élément en récupérant sa "position" dans le tableau à l'aide de la méthode getPoints(List, float).

```
private static int getPoints(List<Float> seuils, float valeur) {
    for(int i = 0 ; i < seuils.size() ; i++) {
        if(valeur < seuils.get(i))
            return i;
    }

    return seuils.size();
}
```

FIGURE 4.2 – Code de la méthode getPoints(List, float)

On remarque l'utilisation de lignes telles que "p.getCategNutriscore().getId() == CONSTANTE". Cet id peut prendre plusieurs valeurs qui sont répertoriées sous forme de constantes dans la classe. Elles permettent de gérer les adaptations du nutriscore au système français. Ainsi, par exemple, on utilise une expression ternaire pour récupérer le nombre de points en énergie. Cela permet de gérer le cas où le produit est une boisson énergétique.

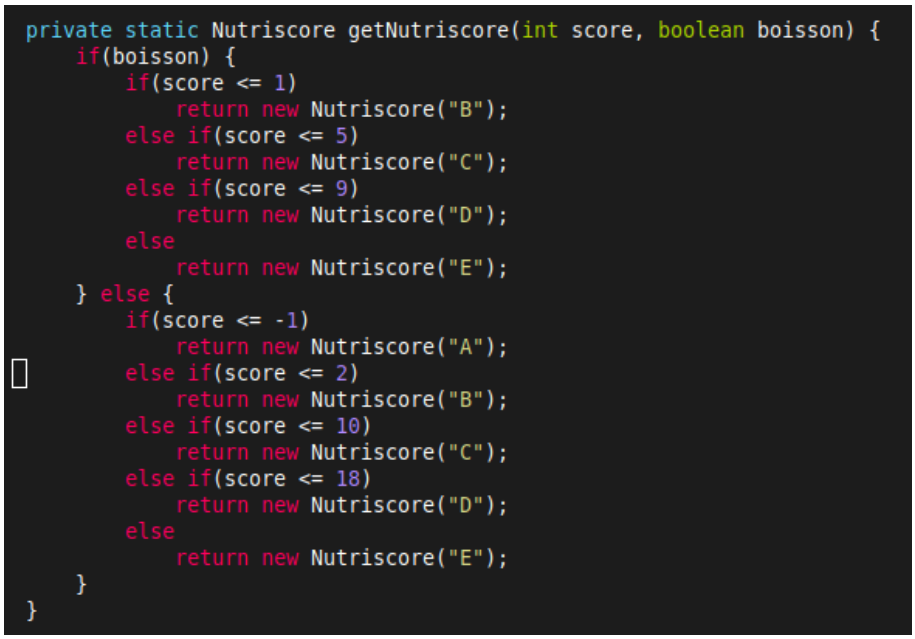
Une fois tous les points récupérés, il faut maintenant récupérer le score total en utilisant la méthode calculerScore. Cette méthode permet d'obtenir le score final d'un produit.



```
private static int calculerScore(Produit p, int pointsEnergie, int pointsGraisses, int pointsSucre, int pointsSodium, int pointsFruits, int pointsFibre, int pointsProteine) {
    int pointsA = pointsEnergie + pointsGraisses + pointsSucre + pointsSodium;
    int pointsC = pointsFruits + pointsFibre + pointsProteine;

    if(p.getCategNutriscore().getId() == ID_FROMAGE) {
        pointsC -= pointsProteine;
        pointsA -= pointsProteine;
    }

    int score = 0;

    if(pointsA < 11 || pointsFruits == 5) {
        score = pointsA - pointsC;
    } else if(pointsFruits < 5){
        score = pointsA - (pointsFibre + pointsFruits);
    }

    return score;
}
```

FIGURE 4.3 – Code de la méthode calculerScore

Une fois ce score obtenu, getNutriscore(Produit) doit maintenant revoyer le nutriscore associé en utilisant la méthode getNutriscore(int, boolean) (la boolean indiquant s'il s'agit d'une boisson ou non).



```
private static Nutriscore getNutriscore(int score, boolean boisson) {
    if(boisson) {
        if(score <= 1)
            return new Nutriscore("B");
        else if(score <= 5)
            return new Nutriscore("C");
        else if(score <= 9)
            return new Nutriscore("D");
        else
            return new Nutriscore("E");
    } else {
        if(score <= -1)
            return new Nutriscore("A");
        else if(score <= 2)
            return new Nutriscore("B");
        else if(score <= 10)
            return new Nutriscore("C");
        else if(score <= 18)
            return new Nutriscore("D");
        else
            return new Nutriscore("E");
    }
}
```

FIGURE 4.4 – Code de la méthode getNutriscore(List, float)

5 Amélioration importantes

A Pattern Factory

Comme mentionné auparavant, actuellement les fonctions "createFromCursor" fournissent en paramètre une instance de "Context". Cette méthode est relativement contraignante. Dans un développement futur, il faudrait utiliser le pattern Factory afin d'éviter ce genre de contrainte. En effet, il suffit de créer un setter permettant de changer l'instance du Context. Cette méthode serait appelée une seule fois au lancement de l'activité. Par la suite, en utilisant la méthode static "getDao" de la Factory, on pourra alors récupérer le DAO demandé, qui sera créé en utilisant le Context fourni en paramètre plus tôt dans l'activité. On permet donc d'éviter le passage du Context de méthode en méthode et l'on permet aussi une uniformisation et une abstraction de la création et de la gestion des DAO. La factory pourrait ainsi également gérer les méthodes "open" et "close" des DAO, ce qui permettrait une meilleure gestion de la mémoire.

B Implémentation des DAO manquants

L'application n'étant là que dans un but de test, tous les DAO ne sont pas implémentés. Il faudra donc tous les créer afin de pouvoir gérer l'ensemble des informations d'un produit.

C Shared Preference

Le système de tri se remet à zéro à chaque lancement de l'application. Une amélioration simple et rapide serait de permettre de sauvegarder ce choix dans les SharedPreference d'Android. Cela éviterait de fruster l'utilisateur qui doit actuellement resélectionner à chaque fois son choix pour le tri.

D Utilisation des Fragment

Il pourrait également être interessant d'utiliser des Fragment afin de permettre, notamment, une meilleure gestion des interfaces sur mobile et tablette. Voici un article présentant de manière brève leur utilisation https://mathias-seguy.developpez.com/tutoriels/android/comprendre-fragments/.

6 Ressources

Développeuse du projet : MARTIN Justine
Adresse mail : justine.martin.dev@gmail.com
Github du projet : https://github.com/jmartin-pro/BuyOrNot
Trello : https://trello.com/b/HCXYRgZp/buy-or-not