**iScreen Documentation**

Sommaire :

* DataBase -------------------------------------------------------------------------- Page 2
  + Dao -------------------------------------------------------------------------- Page 2
  + Entity -------------------------------------------------------------------------- Page 2
* Loading -------------------------------------------------------------------------- Page 3
  + Methodes -------------------------------------------------------------------------- Page 3
  + Layouts -------------------------------------------------------------------------- Page 5
* HomeActivity -------------------------------------------------------------------------- Page 6
  + Methodes -------------------------------------------------------------------------- Page 6
  + Menu -------------------------------------------------------------------------- Page 6
* Affichage -------------------------------------------------------------------------- Page 7
  + Methodes -------------------------------------------------------------------------- Page 7
  + Layouts --------------------------------------------------------------------------
* Paramétre --------------------------------------------------------------------------
  + Methodes --------------------------------------------------------------------------
  + Layouts --------------------------------------------------------------------------
* Remote (RESTful) --------------------------------------------------------------------------
  + Model --------------------------------------------------------------------------
  + Rest --------------------------------------------------------------------------
* Task --------------------------------------------------------------------------
* Utility --------------------------------------------------------------------------

**DataBase**

La Classe AppDatabase est une classe de support qui utilise des annotations pour définir la liste des entités et la version de la base de données. Le contenu de cette classe définit la liste des DAO.

**Dao**

« Data Access Object » interface classe est annotée avec l'annotation @Dao Room (Salle) générera une implémentation de méthodes définies. Il existe quatre annotations @Query, @Insert, @Update, @Delete pour effectuer des opérations CRUD. L'annotation @Query est utilisée pour effectuer une opération de lecture sur la base de données.

**Entity**

Les classes dans le dossier « entity » définit les attributs de chaque table, il est indispensable de déclarer un champ comme clé primaire. Il a la propriété de générer automatiquement des valeurs.

**Loading**

Quand l’application démarre, elle vérifiera s’il y a des données suivantes s’il y a des données du Token, de la configuration depuis le serveur et s’il y a des produits.

Si l’une des données est manquante l’application supprime toute donnée qui existe puis affiche une fenêtre d’identification. L’utilisateur entre les informations suivantes, le nom de l’entreprise, le nom d’utilisateur et le mot de passe. Ces données seront sauvegardées dans une base de données locale.

Si les données d’identification sont correctes alors l’application sauvegardera les données motionné précédemment dans la base de données locale.

**Méthodes**

Voici les méthodes utilisé dans l’activité :

// Vérifie le token et les configurations dans la BDD.

**private boolean** getSaveTokenData()

// Affiche la fenêtre du login si l’argument est vrai sinon elle se ferme.  
**private void** dialogEnterServerInfo(**boolean** status)

// Convertir le prix du produit en décimale i.e 11.2300 en 11.2  
**private** **String** decimalPrice(String price\_str)

// Récupérer la configuration locale.  
**private void** getLocalConf()

// Ajouter les serveurs dans la BDD en dur.  
**private void** initServerUrl()

// Initialise la fenêtre du Loign.

**private void** InitServerInfo(Dialog mDialog)

// Vérifie les informations entrée par l’utilisateur.  
**private void** attemptLogin()

// Sauvegarder l’objet serveur active dans la BDD locale. **private void** saveServerurl()

// Exécute une requête d’internaute au serveur.  
**private void** executeLogin(String username, String password)

// Récupère le résultat de la requête d’internaute.  
**@Override**  
**public void** onInternauteLoginTaskComplete(LoginREST loginREST)

// Exécute une requête pour la configuration définie au serveur.  
**private void** executeFindConfiguration()

// Récupère le résultat de la requête de la configuration.  
**@Override**  
**public void** onFindConfiguration(FindConfigurationREST findConfigurationREST)

// Exécute une requête pour récupérer les catégories au serveur.  
**private void** executeFindCategorieProducts()

// Récupère le résultat de la requête des catégories.  
**@Override**  
**public void** onFindCategorieCompleted(FindCategoriesREST findCategoriesREST)

// Exécute une requête pour récupérer les produits au serveur.  
**private void** executeFindProducts()

// Récupère le résultat de la requête des produits.  
**@Override**  
**public void** onFindProductsCompleted(FindProductsREST findProductsREST)

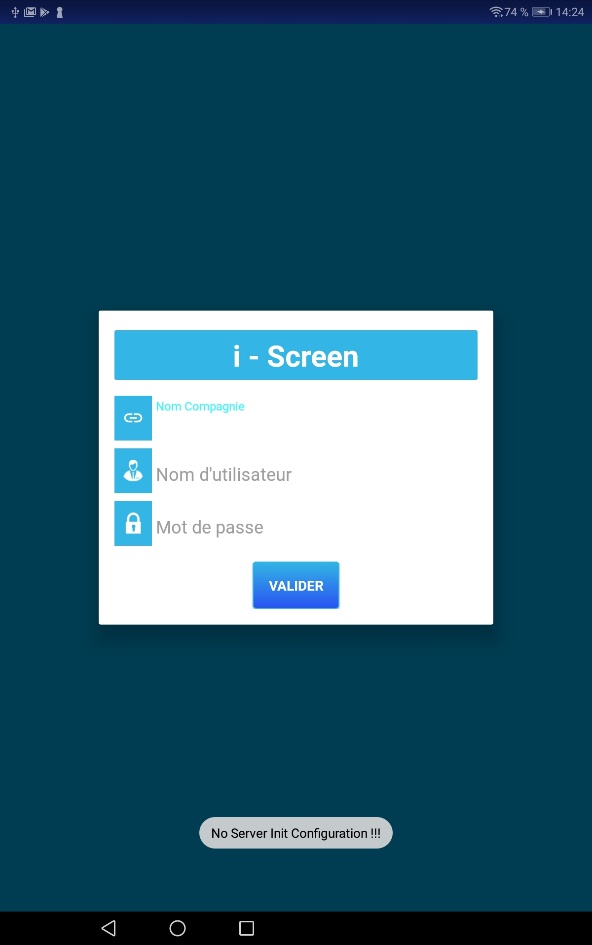
// Exécute une requête pour récupérer les images au serveur.  
**private void** executeFindImageProduct()

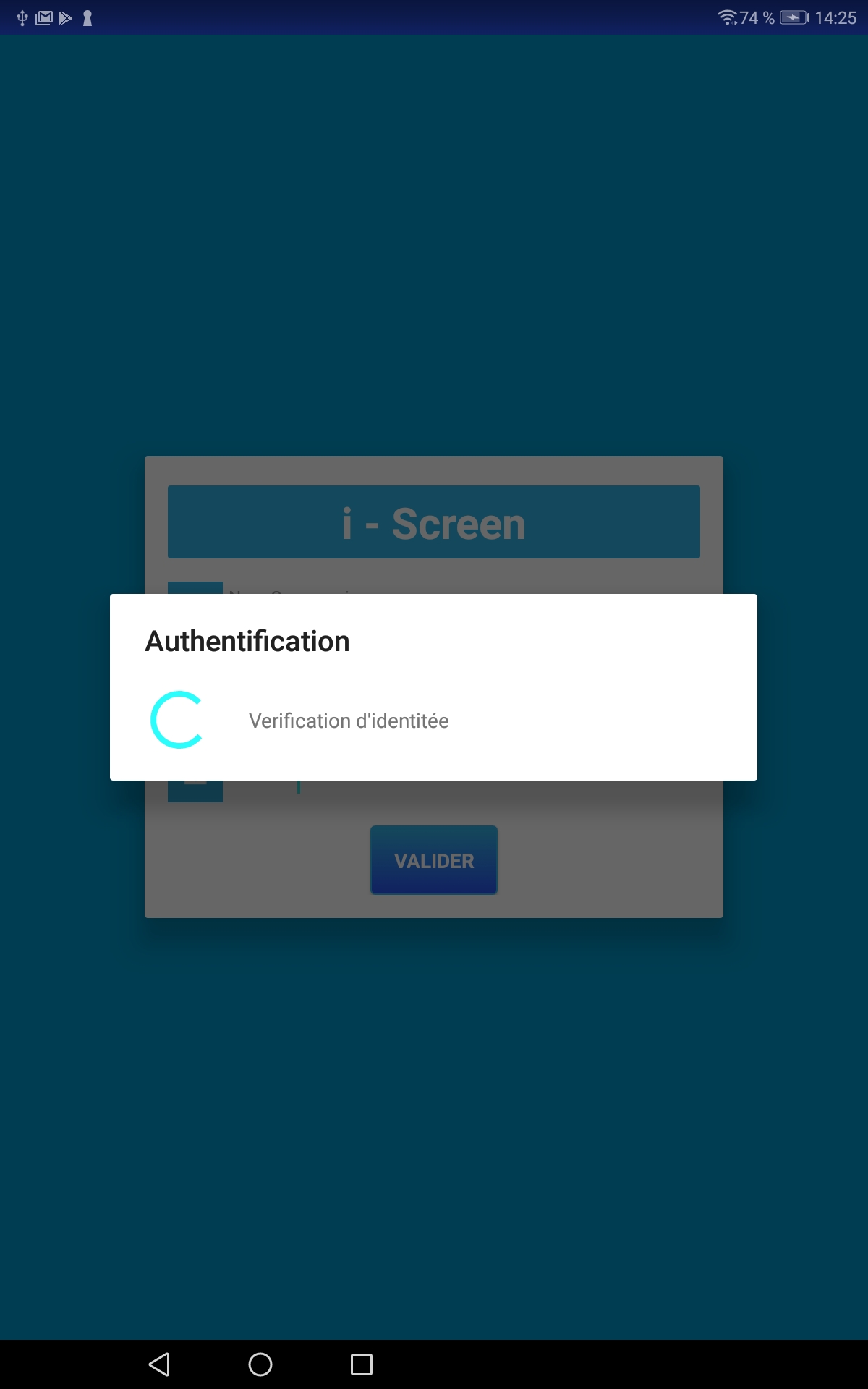
// Récupère le résultat de la requête des images.  
**@Override**  
**public void** onFindImagesProductsComplete(String pathFile)

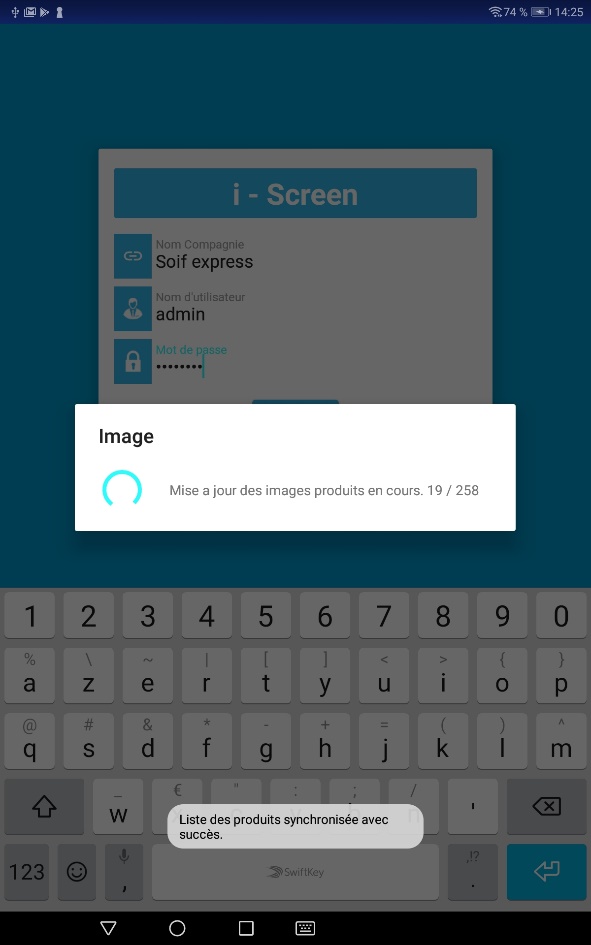
// Montrer/Retirer la bar de progression  
**private void** showProgressDialog(boolean show, String title, String message)

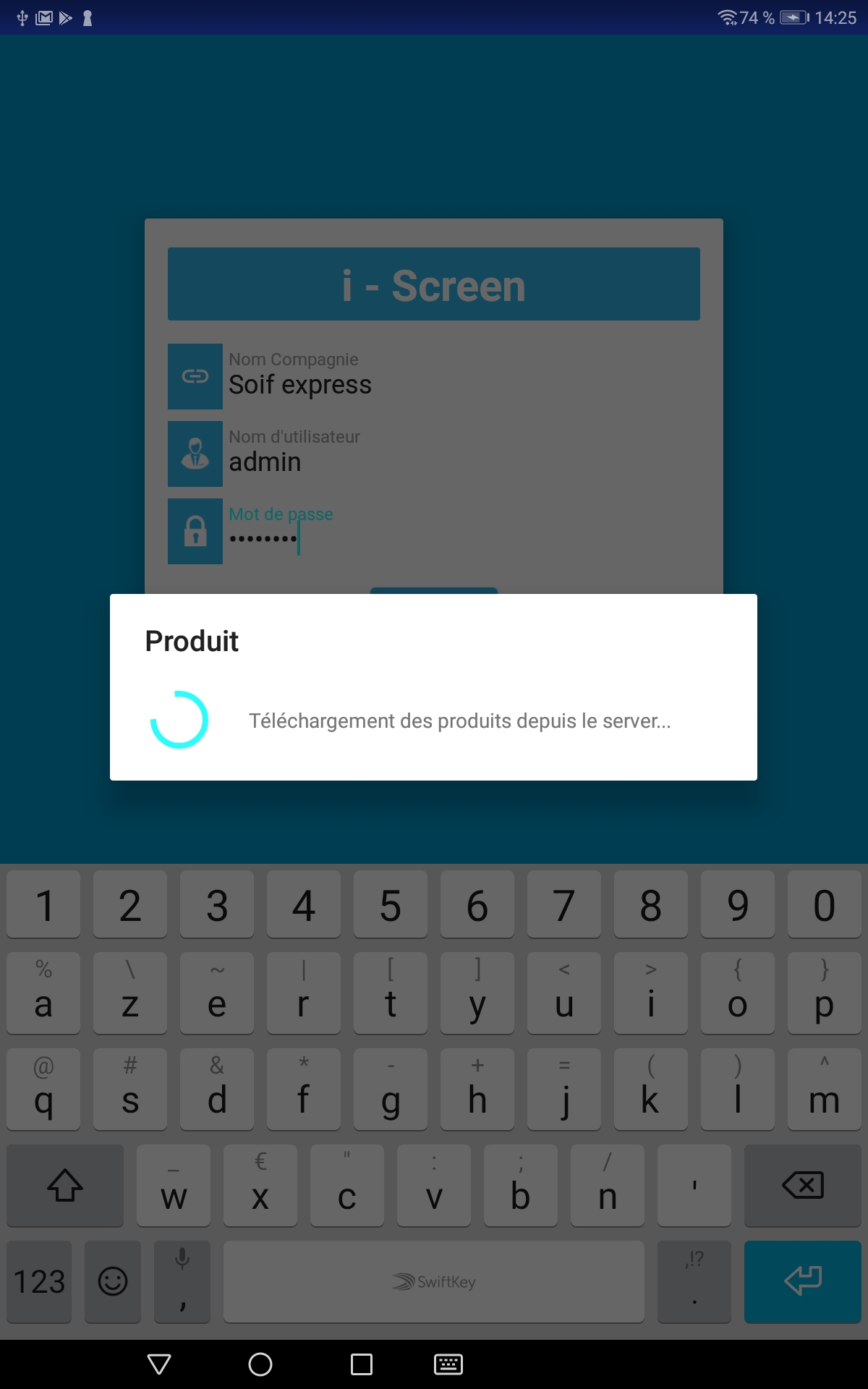
**Layouts**

Ci-dessous représente la vue de l’activité Loading sous le nom de loading\_main.xml et la fenêtre nom de dialog\_server\_info\_login.xml









**HomeActivity**

HomeActivity est une activité qui génère le menu de navigation principale et les fragments nommé « Affichage », « Paramètre » et un bouton de déconnection.  
Le bouton de déconnection supprime toutes les données dans la base de données locale avant de retourner sur l’activité « Loading » avec la fenêtre d’identification.

**Méthode**

//En fonction sur l’élément choisi/appuyé dans le menu, des actions sont effectué.

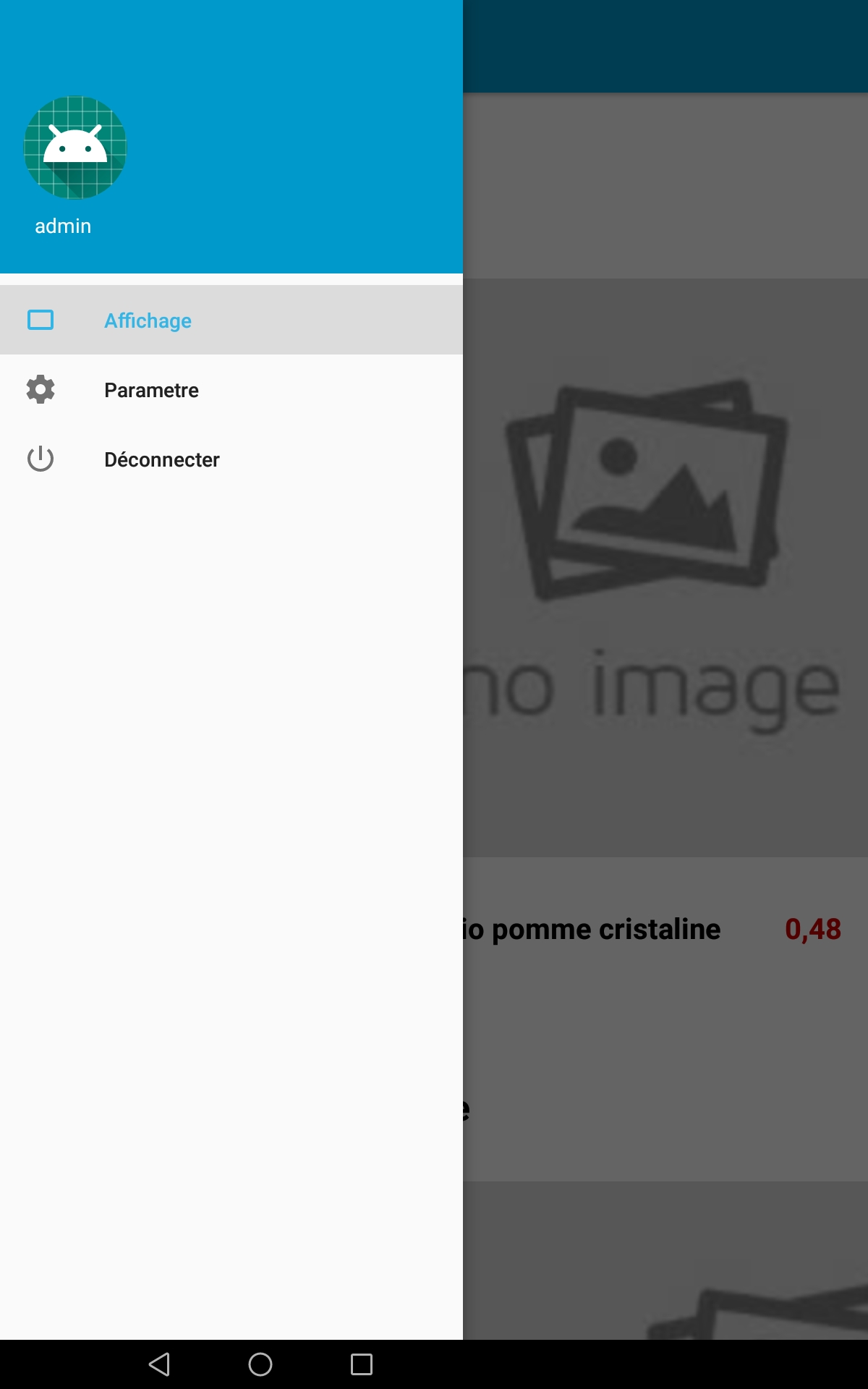
**@Override**  
**public boolean** onNavigationItemSelected(**@NonNull** MenuItem menuItem)

**Menu**

Ci-dessous est le menu de l’application.

La vue du menu est divisée en deux :

* Header (Tête du menu) avec l’image de l’utilisateur et son nom.
* Body (Le corps du menu) avec les options.



**Affichage**

Une fois les catégories, les produits et les images sauvegardé dans la BDD, ici une ou plusieurs carrousels (jusqu’à quatre) sont créé tout dépend la configuration (Initial du serveur ou par l’utilisateur dans les « Paramètre »).

**Méthode**

// Montrer/Retirer la bar de progression  
**private void** showProgressDialog(boolean show, String title, String message)

// Exécute une tâche pour activer et charger les carrousels en fonction des configurations.  
**private void** setupCarrouselData()

// Résulta des listes carrousel.  
@Override  
**public void** onLoadCarouselsData(Carrousel carrousel)

// Charger le carrousel avec la liste aléatoire des produits.  
**private void** getRandomProducts(List<ProduitEntry> randomProductList)

// Charger le carrousel avec la liste aléatoire des produits de chaque catégorie.  
**private void** getRandomFromEachCategory(List<ProduitEntry> randomFromSelectedCategoryList)

// Charger le carrousel avec la liste aléatoire des produits de la catégorie X.   
**private void** getRandomFromCategoryX(List<ProduitEntry> randomFromCategoryXList)

// Charger le carrousel avec la liste des produits recente.  
**private void** getRecentProducts(List<ProduitEntry> recentProductList)

// Cette méthode prend en charge l’animation du carrousel avec comment argument :

La list des produits, le RecyclerView, RecyclerAdapter, LinearLayoutManager et Runnable (Thread).  
**private void** recycleViewAnimation(List<ProduitEntry> productList, final RecyclerView theRecyclerView, final RecyclerView.Adapter adapter, final LinearLayoutManager llm, final Runnable runnable)

**Méthode**