

# iScreen Documentation

Développer par Jean-Laurent DUZANT

Code source disponible sur <https://github.com/JDevs10/iScreen>

## Sommaire :


➤ <b>Dolibarr iScreen Module</b>	-----	Page 2
• Installation	-----	Page 2
• API	-----	Page 3
• Base de données	-----	Page 3
➤ <b>Application iScreen</b>	-----	Page 4
• DataBase	-----	Page 4
▪ Dao	-----	Page 4
▪ Entity	-----	Page 4
• Loading	-----	Page 5
▪ Méthodes	-----	Page 5
▪ Layouts	-----	Page 7
• HomeActivity	-----	Page 8
▪ Méthodes	-----	Page 8
▪ Menu	-----	Page 8
• Affichage	-----	Page 9
▪ Méthodes	-----	Page 9
▪ Layouts	-----	Page 10
• Paramètre	-----	Page 11
▪ Méthodes	-----	Page 11
▪ Layouts	-----	Page 12
• Remote (RESTful)	-----	Page 13
▪ Model	-----	Page 14
▪ Rest	-----	Page 15
• Task	-----	Page 15
• Utility	-----	Page 15

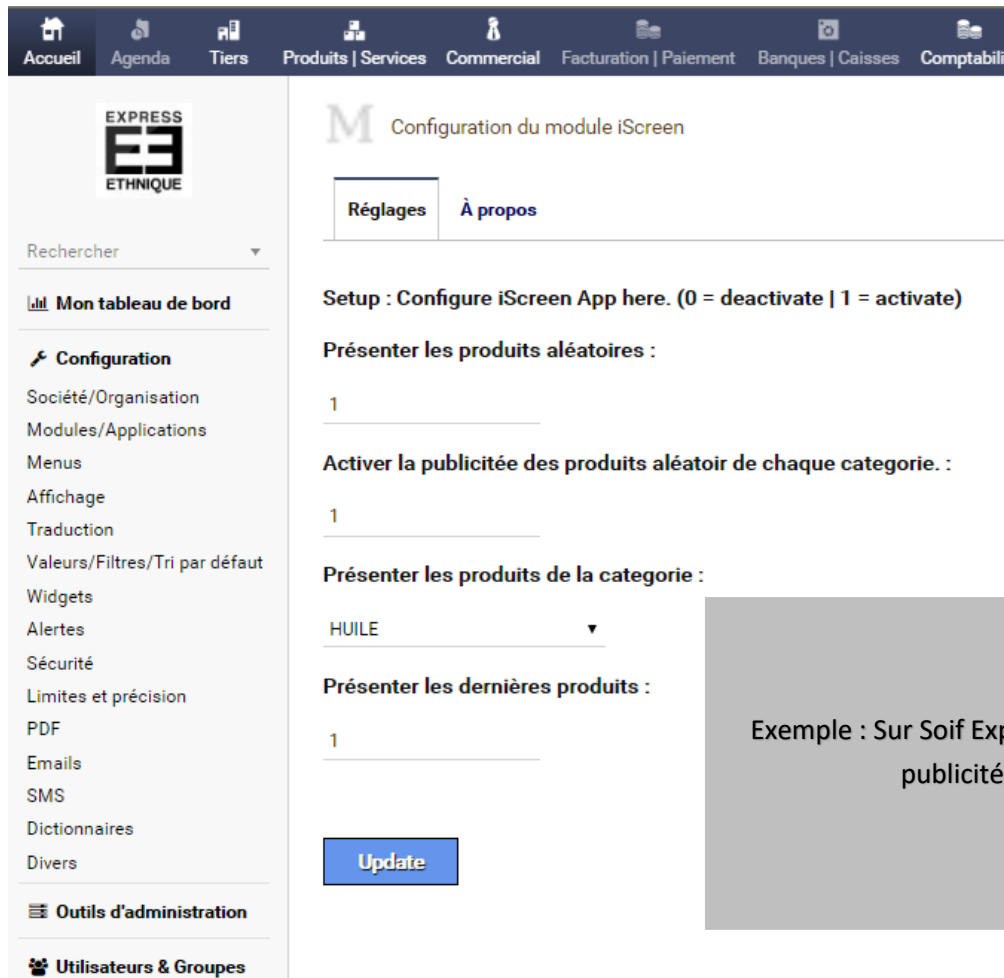
## Dolibarr iScreen Module

Ce module permettra d'enregistrer des configurations prédéfinies avant la **première exécution** (Quand l'utilisateur doit se connecter sur l'appli) de l'application. Cela permet à l'application d'avoir ces configurations de publicité avant que l'utilisateur puisse l'utiliser.

## Installation

Pour installer le module iScreen veuillez suivre les étapes suivantes :

1. **Copier** le dossier « iScreen » dans `Projet_iScreen/Module_Dolibarr` et le **coller** dans `{racine}/custom`.
2. **Activer** le module dans <https://host> → Configuration → Modules/Applications → iScreen.
3. Ensuite **Click** sur l'icône  de configuration du module.
4. Ici dans « Configuration du module iScreen » les valeurs seront 0 et « sélectionner » (pour la liste) par défaut. Les valeurs par défaut sont la désactivation des publicités. Pour activer la publicité il suffit de changer le champ souhaité de **0** à **1** et de changer la sélection de la liste catégorie autre que « sélectionner ».
5. Enregistrer en cliquant sur le bouton « Update ».



Configuration du module iScreen

Réglages À propos

Setup : Configure iScreen App here. (0 = deactivate | 1 = activate)

Présenter les produits aléatoires :

1

Activer la publicité des produits aléatoires de chaque catégorie. :

1

Présenter les produits de la catégorie :

HUILE

Présenter les derniers produits :

1

Update

Exemple : Sur Soif Express, on constate que les 4 publicités sont activées

## API

L'image ci-dessous représente dans Soif Express **API Explorer** les urls pour faire les opérations C.R.U.D.

iscreenapi		Show/Hide   List Operations   Expand Operations
GET	/iscreenapi/getConfig/{id}	Get properties of a iscreenobject object 📄
POST	/iscreenapi/createConfig/{p_aleatoire_iscreen}/{a_category_iscreen}/{category_x_iscreen}/{p_recente_iscreen}	Create iscreenobject object. Setup : Configure iScreen App here. (0 = deactivate   1 = activate) 📄
PUT	/iscreenapi/updateConfig/{rowid}/{p_aleatoire_iscreen}/{a_category_iscreen}/{category_x_iscreen}/{p_recente_iscreen}	Update iscreenobject object. Setup : Configure iScreen App here. (0 = deactivate   1 = activate) 📄
DELETE	/iscreenapi/deleteConfig/{rowid}	Delete iscreenobject 📄

## Base de données

L'image ci-dessous représente la table « llx\_iscreen\_iscreenobject ».

✓ Affichage des lignes 0 - 0 (total de 1, traitement en 0.0005 seconde(s).)					
SELECT * FROM `llx_iscreen_iscreenobject`					
<input type="checkbox"/> Tout afficher   Nombre de lignes : 25 ▼   Filtrer les lignes : <input type="text" value="Chercher dans cette table"/>					
+ Options					
<input type="checkbox"/> <input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer					
	rowid	p_aleatoire	a_category	category_x	p_recente
<input type="checkbox"/>	1	1	1	129	1
<input type="checkbox"/> Tout cocher   Avec la sélection : <input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer <input type="checkbox"/> Exporter					
<input type="checkbox"/> Tout afficher   Nombre de lignes : 25 ▼   Filtrer les lignes : <input type="text" value="Chercher dans cette table"/>					

## Application iScreen

### 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()

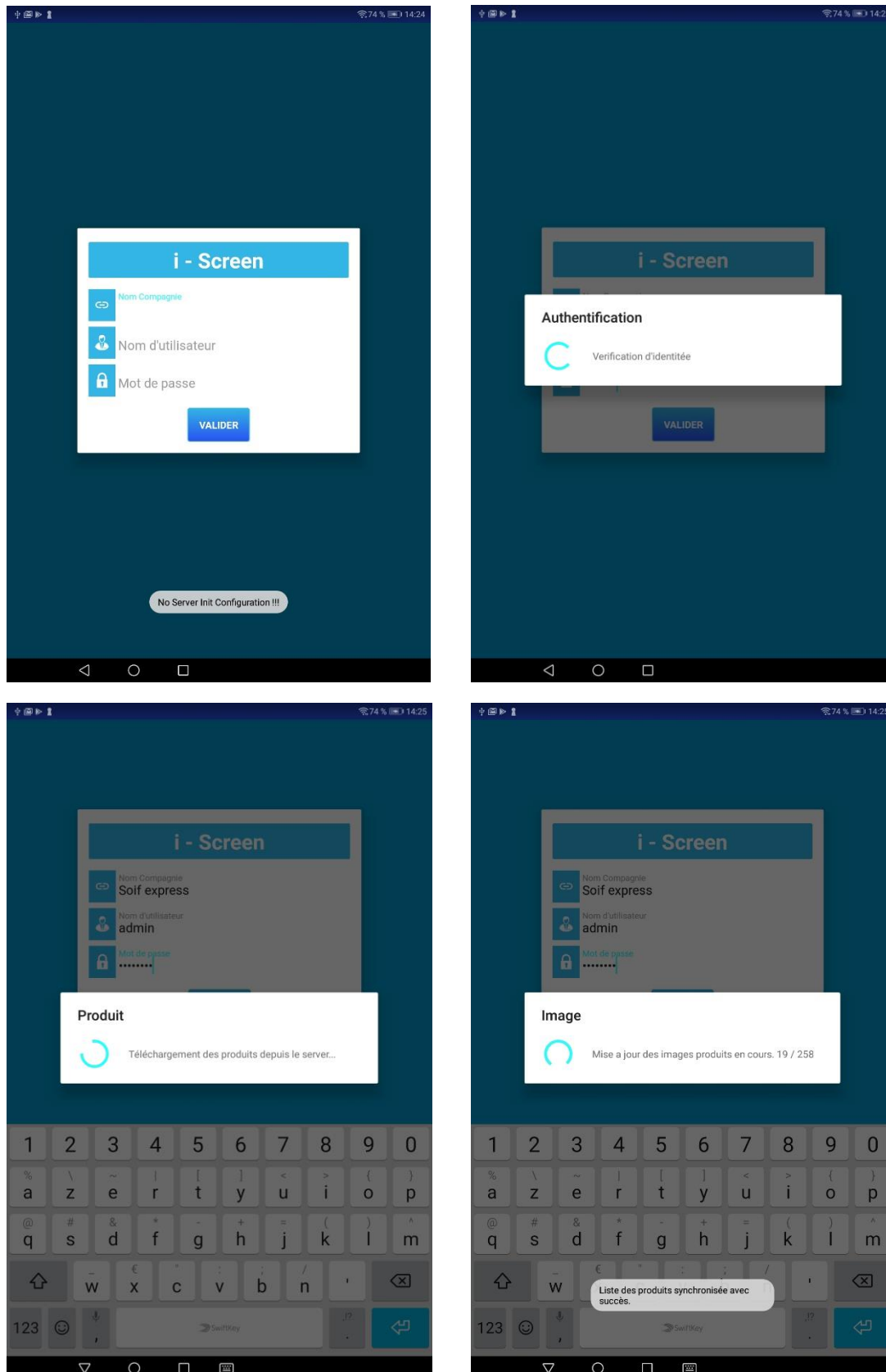
// Exécute une requête d'internaute au serveur.

**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

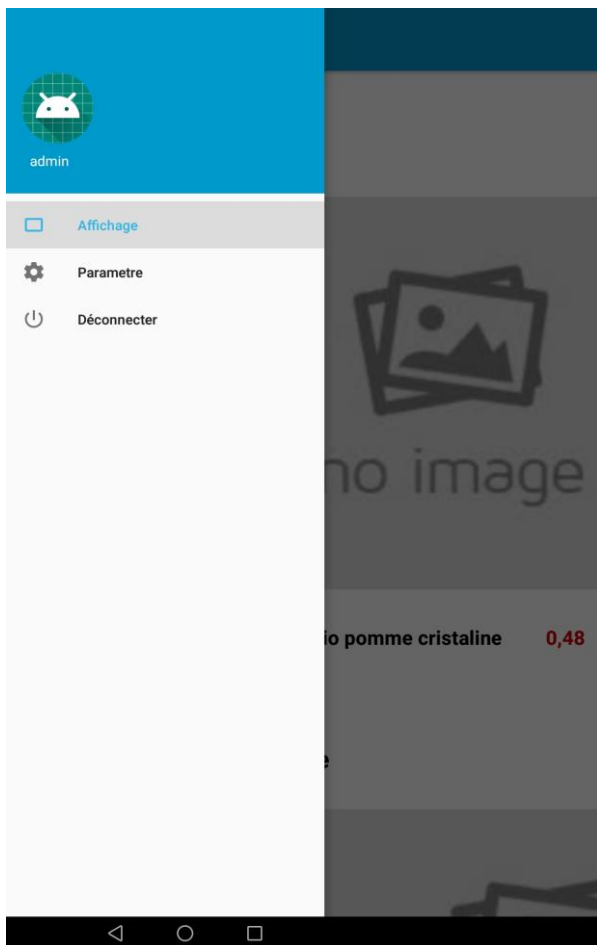
// En fonction sur l'élément choisi/appuyé dans le menu, des actions sont effectuées.

**@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 setupCarouselData()

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

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

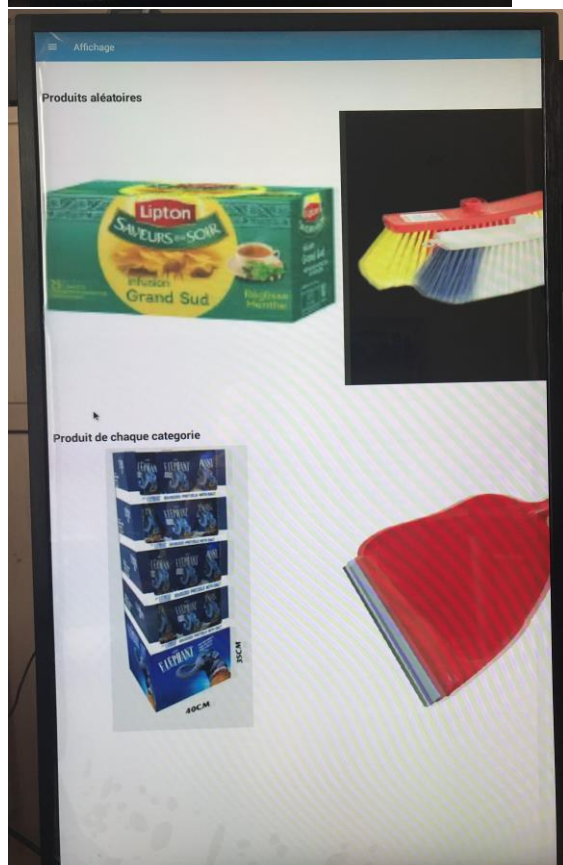
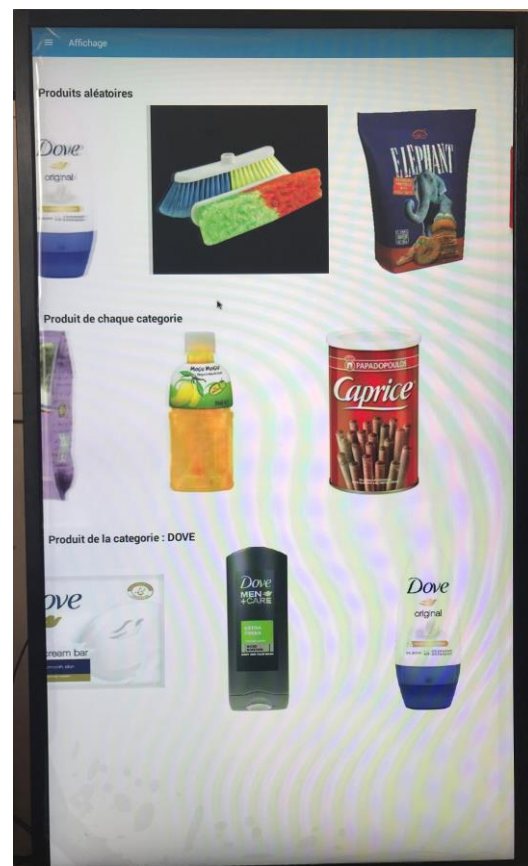
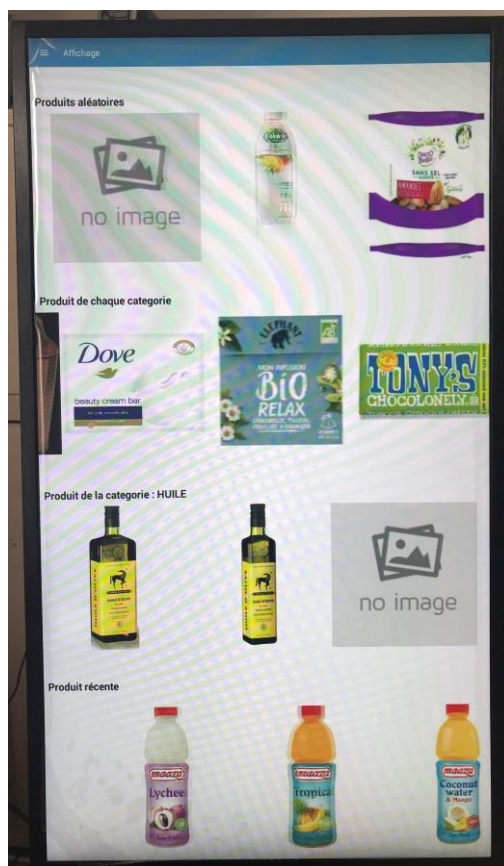
// Charger le carousel avec la liste aléatoire des produits de chaque catégorie.
private void getRandomFromEachCategory(List<ProduitEntry>
randomFromSelectedCategoryList)

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

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

// Cette méthode prend en charge l'animation du carousel avec comment argument :
// La list des produits, le RecyclerView, RecyclerView.Adapter, LinearLayoutManager et Runnable
// (Thread).
private void recycleViewAnimation(List<ProduitEntry> productList, final RecyclerView
theRecyclerView, final RecyclerView.Adapter adapter, final LinearLayoutManager llm, final
Runnable runnable)
```

## Layouts



## Paramètre

Comme vous avez vu récemment sur les images de l’Affichage (Catalogue). En fonction des paramètres reçus depuis Dolibarr et d’autre paramètre spécialement pour les carrousels sur l’application. L’application peut afficher un jusqu’à quatre carrousels, activer le défilement des carrousels, changer la vitesse du défilement des carrousels et donne le nombre de produit dans chaque carrousel.

Les données de configuration **locale** pour les carrousels et non les configurations (Produits, Catégorie) venant de Dolibarr sont enregistrées dans l’appareil. Une fois l’une de ces configurations changées, elles sont enregistrées automatiquement.

## Méthode

// Afficher toute la configuration sur l’écran

**private void** `displayCurrentConfig`(Configuration config)

// Cette méthode retourne la position de la catégorie dans la liste déroulent sur l’application

**private int** `setSelectedCategory`(String categoryID)

// Cette méthode prend le nom de la catégorie comme argument et retourne l’id (rowid) de cette catégorie.

**private String** `getCategoryId`(String name)

// Cette méthode retourne la taille des carrousels enregistre dans la BDD locale

**private int** `setSelectedSize`(int size)

// Cette méthode retourne la vitesse des carrousels enregistre dans la BDD locale

**private int** `setSelectedSpeed`(int speed)

// Cette méthode retourne une liste des noms (label) de catégorie

**private List<String>** `getAllCategoryLabels`()

// Cette méthode retourne une liste des tailles prédéfini

**private List<String>** `getCarouselSizes`()

// Cette méthode retourne une liste des vitesses prédéfini

**private List<String>** `getCarouselSpeeds`()

// Cette méthode sauvegarde les données d’activation des carrousels sur le serveur.

**private void** `saveToServer`()

## Layout

The screenshot shows a configuration panel for the iScreen module. It contains several settings with checkboxes and dropdown menus. Blue lines connect specific settings to callout boxes on the right. A blue 'SAUVEGARDER' button is at the bottom, with a callout box explaining its state.

☐ Activer la publicité des produits aléatoire.

☐ Activer la publicité des produits aléatoire de chaque categorie.

Activer la publicité des produits aléatoire de la categorie. ▼

☐ Activer la publicité des produits récente

Nombre de produit affiché dans le carrousel ▼

☐ Activer le défilement des produits

Vitesse du défilement des produits ▼

**SAUVEGARDER**

Ces configurations sont récupérées depuis le serveur. Elles sont définies par le **Module iScreen**.

Ces configurations sont définies par l'**application** par default.

Le bouton est désactivé par défaut, mais elle est activée si et seulement si l'utilisateur modifie l'une des configurations du serveur (config du **Module iScreen**).

## Remote (RESTful)

Dans le dossier « remote » contient les fichiers pour vérifier la connexion internet, communiqué avec le serveur (Dolibarr API) et télécharger les images des produits.

- **ApiUtils** fourni une instance des services de l'API iScreen par la fonction publique statique « getIScreenService » qui prend **un contexte comme argument et retourne un objet IScreenServiceRemote**, cela donne accès à tous les services de l'API. **ApiUtils** renvoi l'url de récupération des images de produits avec la fonction publique statique « getDownloadProductImg » qui prend **un contexte et la référence du produit comme argument et retourne l'url du téléchargement en string**.
- **IScreenServiceRemote** est une classe publique interface publique avec des méthodes qui héberger nos requêtes d'API, nous utilisons ensuite les annotations de la bibliothèque Retrofit pour annoter les méthodes d'interface avec les spécificités de chaque requête (@POST, @GET, @PUT, @DELETE).
- **ConnectionManager** vérifier la connexion internet. Elle a une fonction publique statique « isPhoneConnected » qui prend **un contexte comme argument et retourne un boolean** vrai ou faux.
- **RetrofitClient** crée le client Retrofit avec le client http, Dolibarr API key (Token de l'utilisateur généré par dolibarr et sauvegarde dans la BDD locale) et l'url de la requête. Cela gère les résultats des appels API.

## Model

Dans le dossier « model » correspond les classes de tous les objets qui circulent entre Dolibarr API et l'application. Un exemple avec la configuration pour activer/désactiver les carrousels de produit et produit de la catégorie choisi. Ci-dessous sont la classe Config et la table « llx\_iscreenapi\_myobject »

```
package com.example.iscreen.remote.model;

public class Config {
    private String rowid;
    private String p_aleatoir;
    private String a_category;
    private String category_x;
    private String p_recente;

    public Config() {
    }

    public String getRowid() { return rowid; }

    public void setRowid(String rowid) { this.rowid = rowid; }

    public String getP_aleatoir() { return p_aleatoir; }

    public void setP_aleatoir(String p_aleatoir) { this.p_aleatoir = p_aleatoir; }

    public String getA_category() { return a_category; }

    public void setA_category(String a_category) { this.a_category = a_category; }

    public String getCategory_x() { return category_x; }

    public void setCategory_x(String category_x) { this.category_x = category_x; }

    public String getP_recente() { return p_recente; }

    public void setP_recente(String p_recente) { this.p_recente = p_recente; }
}
```

On constate que chaque colonne de la table « llx\_iscreen\_iscreenobject » correspond à un attribut dans la classe Config.

✓ Affichage des lignes 0 - 0 (total de 1, traitement en 0.0005 seconde(s).)

`SELECT * FROM `llx_iscreen_iscreenobject``

☐ Tout afficher | Nombre de lignes : 25 | Filtrer les lignes : Chercher dans cette table

+ Options

	rowid	p_aleatoir	a_category	category_x	p_recente
<input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer	1	1	1	129	1

↑ ☐ Tout cocher Avec la sélection : ☐ Éditer ☐ Copier ☐ Supprimer ☐ Exporter

☐ Tout afficher | Nombre de lignes : 25 | Filtrer les lignes : Chercher dans cette table

## Rest

Dans le dossier « rest », les classes stockent temporairement les données reçues depuis l'API Dolibarr. Ces classes sont principalement utilisées par les classes de tâche en arrière-plan de l'application.

## Task

Les classes dans le dossier « task » (tâche) sont des classes s'étendant d'une classe publique abstraite AsyncTask. Elle effectue des opérations en arrière-plan sur le thread d'arrière-plan et se met à jour sur le thread principal. Dans Android, nous ne pouvons pas toucher directement le fil de fond au fil principal du développement Android. AsyncTask nous aide à établir la communication entre le fil d'arrière-plan et le fil principal.

AsyncTask propose 4 méthodes mais on utilise 2 ou 3.

- **onPreExecute()** – Avant d'effectuer des opérations en arrière-plan, nous devons montrer à l'utilisateur un élément tel que progressbar ou toute animation. nous pouvons directement communiquer l'opération d'arrière-plan en utilisant `doInBackground()`, mais pour les meilleures pratiques, nous devrions appeler toutes les méthodes `asyncTask`.
- **doInBackground(Params)** – Dans cette méthode, nous devons effectuer une ou plusieurs opérations en arrière-plan sur le fil d'arrière-plan. Les opérations effectuées avec cette méthode ne doivent toucher aucune activité ou fragment principal.  
**Cette méthode est obligatoire pour AsyncTask.**
- **onProgressUpdate(Progress...)** – En effectuant des opérations en arrière-plan, si vous souhaitez mettre à jour des informations sur l'interface utilisateur, nous pouvons utiliser cette méthode.
- **onPostExecute(Result)** – Dans cette méthode, nous pouvons mettre à jour l'interface utilisateur du résultat de l'opération en arrière-plan.

## Utility

La classe « IScreenUtility » gère l'enregistrement du dossier « iScreen/iScreen Produits » dans l'appareil, enregistrer les produits image dans le dossier crée supprimer le dossier iScreen image.